

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.

¹Lorica, 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 matrix compiler on Ubuntu 16.04.

2.1.1 Installation

The following commands will install the required dependencies:

```
1  sudo apt install ocamll llvm llvm-runtime m4 build-essential
   ↳ autoconf automake opam clang
2  opam init
3  opam install llvm.3.8
4  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 matrix.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 like `/usr/bin` and insure that directory is a part of your `$PATH` so you can run `matrix.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, MATRIX 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

MATRIX 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 its 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
- `equals(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 Ocamlacc and Ocamllex extensions for compiling the scanner and parser front end. We implemented the matrix library in C.

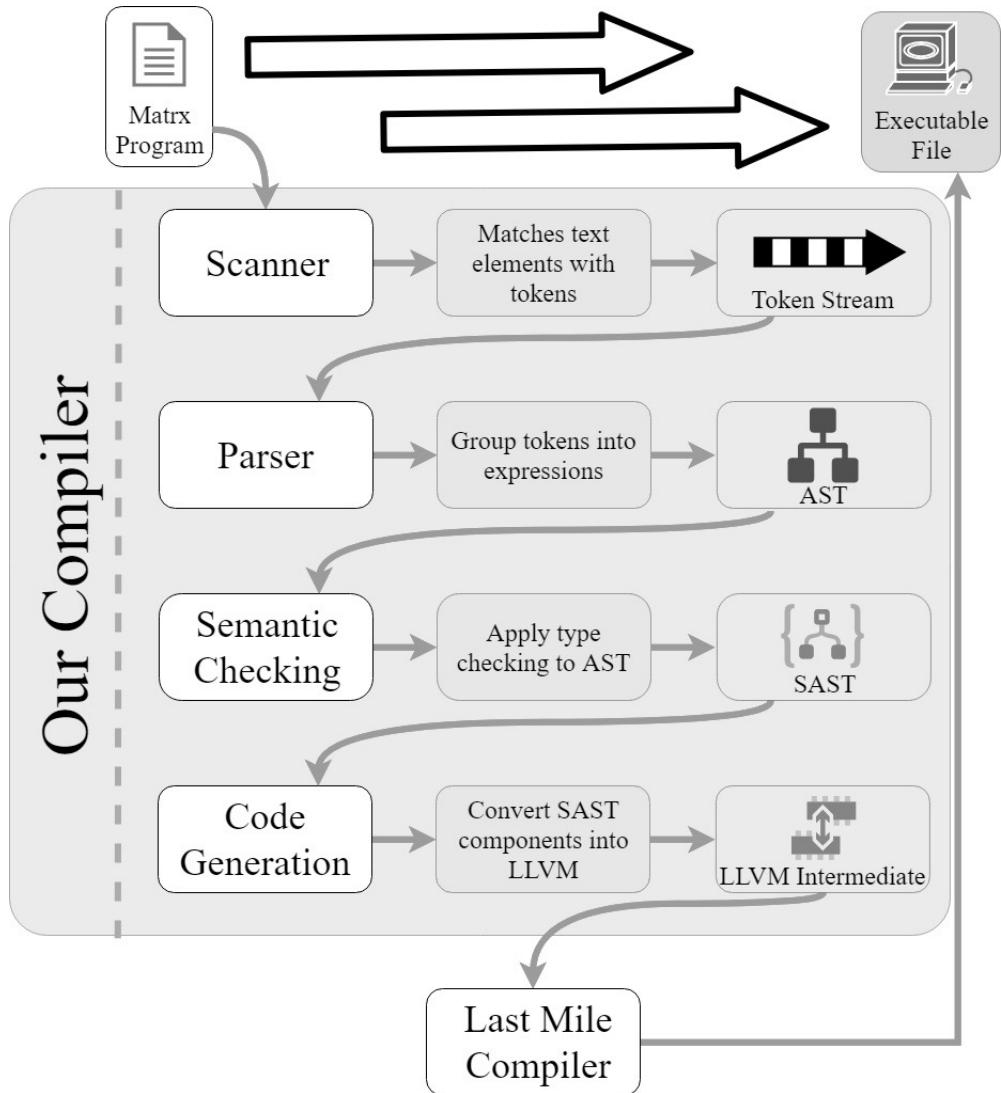
Development Environments: Different members preferred to code in different environments including: vim, Visual Studio, and Sublime Text.

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.

```
1 ; ModuleID = 'Matrx'
2 source_filename = "Matrx"
3
4 %struct.matrix = type { i32, i32, i32**, i32 }
5
6 @fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
7 @fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
8 @fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
9
10 declare i32 @printf(i8*, ...)
11
12 declare i32 @display(%struct.matrix*)
13
14 declare %struct.matrix* @initMatrix_CG(i32, i32)
15
16 declare %struct.matrix* @storeVal(%struct.matrix*, i32)
17
18 declare %struct.matrix* @transpose(%struct.matrix*)
19
20 declare %struct.matrix* @inverse(%struct.matrix*)
21
22 declare %struct.matrix* @matrixMult(%struct.matrix*, %struct.
23     → matrix*)
24
25 declare %struct.matrix* @mAdd(%struct.matrix*, %struct.matrix*)
26
27 declare %struct.matrix* @dotProduct(%struct.matrix*, %struct.
28     → matrix*)
29
30 declare %struct.matrix* @timesScalar(%struct.matrix*, i32)
31
32 declare %struct.matrix* @determinant(%struct.matrix*, i32)
```

```

31     declare i8* @string_get(i8*, i32)
32
33     declare i32 @string_length(i8*)
34
35     declare i8* @string_concat(i8*, i8*)
36
37     declare i32 @string_equals(i8*, i8*)
38
39     declare i8* @string_substr(i8*, i32, i32)
40
41     define i32 @main() {
42         entry:
43             %a = alloca %struct.matrix*
44             %matrix_init = call %struct.matrix* @initMatrix_CG(i32 2, i32
45             ↪ 2)
46             %store_val = call %struct.matrix* @storeVal(%struct.matrix* %
47             ↪ matrix_init, i32 1)
48             %store_val1 = call %struct.matrix* @storeVal(%struct.matrix*
49             ↪ %matrix_init, i32 2)
50             %store_val2 = call %struct.matrix* @storeVal(%struct.matrix*
51             ↪ %matrix_init, i32 3)
52             %store_val3 = call %struct.matrix* @storeVal(%struct.matrix*
53             ↪ %matrix_init, i32 4)
54             store %struct.matrix* %matrix_init, %struct.matrix** %a
55             %a4 = load %struct.matrix*, %struct.matrix** %a
56             %printfm = call i32 @display(%struct.matrix* %a4)
57             ret i32 0
58     }

```

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.

```

1      ; ModuleID = 'Matrx'
2      source_filename = "Matrx"
3
4      %struct.matrix = type { i32, i32, i32**, i32 }
5
6      @fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
7      @fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
8      @fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
9      @tmp = private unnamed_addr constant [19 x i8] c"This is an
   ↪ example\00"
10
11     declare i32 @printf(i8*, ...)
12
13     declare i32 @display(%struct.matrix*)
14
15     declare %struct.matrix* @initMatrix_CG(i32, i32)
16
17     declare %struct.matrix* @storeVal(%struct.matrix*, i32)
18
19     declare %struct.matrix* @transpose(%struct.matrix*)
20
21     declare %struct.matrix* @inverse(%struct.matrix*)
22
23     declare %struct.matrix* @matrixMult(%struct.matrix*, %struct.
   ↪ matrix*)
24
25     declare %struct.matrix* @mAdd(%struct.matrix*, %struct.matrix*)
26
27     declare %struct.matrix* @dotProduct(%struct.matrix*, %struct.
   ↪ matrix*)
28
29     declare %struct.matrix* @timesScalar(%struct.matrix*, i32)
30
31     declare %struct.matrix* @determinant(%struct.matrix*, i32)
32
33     declare i8* @string_get(i8*, i32)
34
35     declare i32 @string_length(i8*)
36
37     declare i8* @string_concat(i8*, i8*)
38
39     declare i32 @string_equals(i8*, i8*)
40
41     declare i8* @string_substr(i8*, i32, i32)
42
43     define i32 @main() {
44       entry:
45         %s1 = alloca i8*
46         store i8* getelementptr inbounds ([19 x i8], [19 x i8]* @tmp,
   ↪ i32 0, i32 0), i8** %s1
47         %s11 = load i8*, i8** %s1
48         %string_get = call i8* @string_get(i8* %s11, i32 0)
49         %printf = call i32 (i8*, ...) @printf(i8* getelementptr
   ↪ inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0),
   ↪ i8* %string_get)
50         ret i32 0
51     }
```

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.

```

1 ; ModuleID = 'Matrx'
2 source_filename = "Matrx"
3
4 %struct.matrix = type { i32, i32, i32**, i32 }
5
6 @fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
7 @fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
8 @fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
9 @tmp = private unnamed_addr constant [6 x i8] c"MATRX\00"
10 @tmp.3 = private unnamed_addr constant [6 x i8] c"MATRX\00"
11 @tmp.4 = private unnamed_addr constant [22 x i8] c"of a string
   ↪ in MATRX!\00"
12 @tmp.5 = private unnamed_addr constant [20 x i8] c"This is an
   ↪ example \00"
13 @tmp.6 = private unnamed_addr constant [41 x i8] c"This is an
   ↪ example of a string in MATRX!\00"
14
15 declare i32 @printf(i8*, ...)
16
17 declare i32 @display(%struct.matrix*)
18
19 declare %struct.matrix* @initMatrix_CG(i32, i32)
20
21 declare %struct.matrix* @storeVal(%struct.matrix*, i32)
22
23 declare %struct.matrix* @transpose(%struct.matrix*)
24
25 declare %struct.matrix* @inverse(%struct.matrix*)
26
27 declare %struct.matrix* @matrixMult(%struct.matrix*, %struct.
   ↪ matrix*)
28
29 declare %struct.matrix* @mAdd(%struct.matrix*, %struct.matrix*)
30
31 declare %struct.matrix* @dotProduct(%struct.matrix*, %struct.
   ↪ matrix*)
32
33 declare %struct.matrix* @timesScalar(%struct.matrix*, i32)
34
35 declare %struct.matrix* @determinant(%struct.matrix*, i32)
36
37 declare i8* @string_get(i8*, i32)
38
39 declare i32 @string_length(i8*)
40

```

```

41     declare i8* @string_concat(i8*, i8*)
42
43     declare i32 @string_equals(i8*, i8*)
44
45     declare i8* @string_substr(i8*, i32, i32)
46
47     define i32 @main() {
48         entry:
49             %string_equals = call i32 @string_equals(i8* getelementptr
50             ↪ inbounds ([6 x i8], [6 x i8]* @tmp.3, i32 0, i32 0),
51             ↪ i8* getelementptr inbounds ([6 x i8], [6 x i8]* @tmp,
52             ↪ i32 0, i32 0))
53             %printf = call i32 (i8*, ...) @printf(i8* getelementptr
54             ↪ inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
55             ↪ %string_equals)
56             %string_concat = call i8* @string_concat(i8* getelementptr
57             ↪ inbounds ([20 x i8], [20 x i8]* @tmp.5, i32 0, i32 0),
58             ↪ i8* getelementptr inbounds ([22 x i8], [22 x i8]* @tmp.4, i32 0, i32 0))
59             %printf1 = call i32 (i8*, ...) @printf(i8* getelementptr
60             ↪ inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0),
61             ↪ i8* %string_concat)
62             %string_substr = call i8* @string_substr(i8* getelementptr
63             ↪ inbounds ([41 x i8], [41 x i8]* @tmp.6, i32 0, i32 0),
64             ↪ i32 3, i32 7)
65             %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr
66             ↪ inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0),
67             ↪ i8* %string_substr)
68         ret i32 0
69     }

```

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

```
1 commit 316d083e7498d1c9c1f1fd2b981f5f6fcf89b077
2 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
3     ↪ com>
4 Date:   Tue Dec 18 22:03:21 2018 -0500
5
6     removed inverse function
7
8 commit 3536ec24e958051860ff62308b395908275832c0
9 Author: Nicholas Sparks <>
10 Date:  Mon Dec 17 15:59:27 2018 -0500
11
12     demo update
13
14 commit b09442cd2c76b8da9b4fd6d13f3e0e9b39ccd02b
15 Author: Nicholas Sparks <>
16 Date:  Mon Dec 17 15:43:11 2018 -0500
17
18     Updated Dot
19
20 commit 3779751d4b3c9624ff5bc020722561f2fa13f453
21 Author: Nicholas Sparks <>
22 Date:  Mon Dec 17 15:42:55 2018 -0500
23
24     Updated Dot
25
26 commit e7421491d46a9b760fc687d677a5b8b4c569fe64
27 Author: Nicholas Sparks <>
28 Date:  Mon Dec 17 01:31:49 2018 -0500
29
30     Cleanup
31
32 commit 79ab5423738ea18cbfa578781303a8b1caa5d80b
33 Author: Nicholas Sparks <>
34 Date:  Mon Dec 17 01:25:52 2018 -0500
35
36     ignore test compile files
37
38 commit e047f21a61776da0d29095ecf79056b55ad5ca95
39 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
40     ↪ com>
41 Date:  Mon Dec 17 01:21:52 2018 -0500
42
43     fixed dimention mismatch issue
44
45 commit 68ba1bc674b8fbc25359ae572574f5ff1f91dbf2
46 Merge: 7dfd897 e25bd6c
47 Author: Nicholas Sparks <>
48 Date:  Mon Dec 17 00:48:41 2018 -0500
49
50     Dim Fix
51
52 commit 7dfd8971096702106fa04fc5dcd357b6e8f04023
53 Author: Nicholas Sparks <>
54 Date:  Mon Dec 17 00:44:53 2018 -0500
55
56     Fixed Dim Check
```

```

55
56 commit e25bd6ca6c697aacbale69279764a22c6f5f65d1
57 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
      ↢ com>
58 Date: Sun Dec 16 23:22:22 2018 -0500
59     cleanup and added matrix fail tests
60
61 commit b753cc6e838978f2eec8ade8fa0d0994e8fdc335
62 Author: Nicholas Sparks <>
63 Date: Sun Dec 16 13:09:18 2018 -0500
64
65     Demo
66
67 commit ad20a47f54d167abfd2ce01a2bf50b87a2de3ced
68 Merge: cbf57a8 9fa2dc
69 Author: Nicholas Sparks <>
70 Date: Sun Dec 16 12:51:26 2018 -0500
71
72     Merge branch 'master' of https://github.com/PearceKieser/CS4115
73
74 commit cbf57a81f38f01e6778d47accc4215da66e7db8b
75 Author: Nicholas Sparks <>
76 Date: Sun Dec 16 12:51:14 2018 -0500
77
78     Invalid Dimensions catch
79
80 commit 9fa2dc3361abd7f4a6b44689ca3321ad094a0f9
81 Author: Katie Pfleger <kjp2157@columbia.edu>
82 Date: Sun Dec 16 11:41:19 2018 -0500
83
84     Updated string tests
85
86 commit 9c6bc9f791f10ae2f935025de26fe9f6b7a7dafa
87 Author: Nicholas Sparks <>
88 Date: Sun Dec 16 11:13:42 2018 -0500
89
90     Proper naming
91
92 commit 2acffc69751f7c456db594e634f539c5506445d7
93 Author: Nicholas Sparks <>
94 Date: Sun Dec 16 11:10:35 2018 -0500
95
96     Change file extension to mx
97
98 commit e00f81080a71b20393b13439e756a0d0c3f36d16
99 Author: Nicholas Sparks <>
100 Date: Sun Dec 16 02:55:11 2018 -0500
101
102     fix
103
104 commit ea451a3599e5c9b355167515074d27d64547ce73
105 Author: Nicholas Sparks <>
106 Date: Sun Dec 16 02:52:51 2018 -0500
107
108     corrupt test
109
110 commit 68d526acaba0ae3c1899dbcd48f08ea9e0b56e84
111 Author: Nicholas Sparks <>
112 Date: Sun Dec 16 02:45:28 2018 -0500
113
114     Declare and set Vars
115

```

```

116
117 commit b7743bcb5f599ce540e6c49e06de31032daad722
118 Author: Nicholas Sparks <>
119 Date:   Sat Dec 15 22:44:29 2018 -0500
120
121     Error free make
122
123 commit 31e36bf275c74e8b0e0176e6e483a7860f7a0ce5
124 Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
125     ↪ com>
126 Date:   Sat Dec 15 19:50:48 2018 -0500
127
128     cleaning codebase
129
130 commit 1ad9672ab5fc1f659fa9dfcc9ddef4dedf2051e8
131 Author: Nicholas Sparks <>
132 Date:   Sat Dec 15 16:44:37 2018 -0500
133
134     more tests
135
136 commit db04049edf774cdcb1da5ad6a04f56a336a54dd9
137 Author: Nicholas Sparks <>
138 Date:   Sat Dec 15 16:38:22 2018 -0500
139
140     tests
141
142 commit 5c66a7a915f8e52c180af04f182b882a435ae91e
143 Author: Nicholas Sparks <>
144 Date:   Sat Dec 15 16:37:39 2018 -0500
145
146     added tests
147
148 commit 35e719ddf4b7d54a38758447ffbee3022cd3eb2a
149 Author: Nicholas Sparks <>
150 Date:   Sat Dec 15 16:24:53 2018 -0500
151
152     Added matrix.c functions
153
154 commit 9a0c5d72f8403767c643a7d1cff5d4dc2f4eebc8
155 Author: Nicholas Sparks <>
156 Date:   Sat Dec 15 14:45:53 2018 -0500
157
158     printm working
159
160 commit 99d35cf20c9909978f9588aac16da02474c76c9b
161 Author: Nicholas Sparks <>
162 Date:   Sat Dec 15 14:05:30 2018 -0500
163
164     Codegen update
165
166 commit 750ae5efeca8160f4324b78eb4f50e9307c5407
167 Author: Nicholas Sparks <>
168 Date:   Sat Dec 15 12:45:57 2018 -0500
169
170     Checks if all types are equal in Matrix
171
172 commit 932bce9933dc60a217474877b4f9738a28880449
173 Author: Nicholas Sparks <>
174 Date:   Fri Dec 14 14:32:32 2018 -0500
175
176     Some more updates

```

```

177 commit 154ebd139429b17ce3e6e876d185ba008b2d49de
178 Author: Nicholas Sparks <>
179 Date:   Fri Dec 14 12:04:45 2018 -0500
180
181     fixed multi-dimensional Matrix Parsing , & corrupt testall.sh
182
183 commit 0faa21e200d4ab9d20723927c05c58cb41871d76
184 Author: Katie Pfleger <kjp2157@columbia.edu>
185 Date:   Thu Dec 13 23:31:24 2018 -0500
186
187     String implementation/functions working and cleaned up string
188     ↪ tests
189
190 commit 48ae589cb112fdc67ffbf97377dd98b2641d485
191 Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
192     ↪ com>
193 Date:   Thu Dec 13 20:31:28 2018 -0500
194
195     removed old OurParser files
196
197 commit 367dce3a7fc065bff994b025c203de53936d8598
198 Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
199     ↪ com>
200 Date:   Thu Dec 13 20:23:59 2018 -0500
201
202     fixed matrix.c linking problem
203
204 commit c3621943cfb54392a99b7985fa836924890cc4db
205 Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
206     ↪ com>
207 Date:   Thu Dec 13 19:58:10 2018 -0500
208
209     some directory cleanup
210
211 commit 7dd3b44a84cf0dbac4c17f48a1cbd68fce754a4
212 Merge: ea6c5da c6fa85c
213 Author: Katie Pfleger <kjp2157@columbia.edu>
214 Date:   Thu Dec 13 16:46:38 2018 -0500
215
216     Fixed merge conflicts
217
218 commit ea6c5da6dd2854b36a0b0a98ad2da4d42e9b8687
219 Author: Katie Pfleger <kjp2157@columbia.edu>
220 Date:   Thu Dec 13 16:38:13 2018 -0500
221
222     WIP string implementation
223
224 commit c811c1495e13ce7253b465272221dccde2785e01
225 Author: Katie Pfleger <kjp2157@columbia.edu>
226 Date:   Thu Dec 13 16:36:33 2018 -0500
227
228     Updated semant to accept more than one argument for function
229     ↪ decls
230
231 commit c6fa85c63b92d5f5d40aea16ee2b0342e2a97304
232 Merge: f707525 4b284a8
233 Author: Nicholas Sparks <>
234 Date:   Thu Dec 13 13:54:32 2018 -0500
235
236     update
237
238 commit f7075257fa119f7954a0f9bf67b9ae66d1a4dd43

```

```

234 | Author: Nicholas Sparks <>
235 | Date: Thu Dec 13 13:52:12 2018 -0500
236 |
237 |     Converts MatrixLit 2D to 1D Expr List
238 |
239 | commit 4b284a878d7aaf67b07af14fd33b4b955ebb1153
240 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
241 |     ↪ com>
241 | Date: Thu Dec 13 11:48:01 2018 -0500
242 |
243 |     ast print of MatrixLit
244 |
245 | commit 2f08766fb73c7e9f912a7e8afea8782bc0613787
246 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
246 |     ↪ com>
247 | Date: Thu Dec 13 11:41:15 2018 -0500
248 |
249 |     Fixed complie
250 |
251 | commit 42c132a209c4c1dc3be29162c450df2e5e18e779
252 | Author: Nicholas Sparks <>
253 | Date: Thu Dec 13 16:36:05 2018 +0000
254 |
255 |     update
256 |
257 | commit 8f7124d480ed29ecc409584b1a07f3a62b332523
258 | Author: Nicholas Sparks <>
259 | Date: Thu Dec 13 11:34:47 2018 -0500
260 |
261 |     update
262 |
263 | commit aa8e4dd9ba539cf314346768267ccf10b1b02968
264 | Author: Katie Pfleger <kjp2157@columbia.edu>
265 | Date: Wed Dec 12 20:53:02 2018 -0500
266 |
267 |     String declaration and printing now works, tests included
268 |
269 | commit 98ace61839555c4f334378d228f65ab04c2e31ba
270 | Author: Nicholas Sparks <>
271 | Date: Wed Dec 12 17:22:30 2018 -0500
272 |
273 |     Fixed printing order of functions
274 |
275 | commit 02424f271e8eb86058f497f22160283e5158ad92
276 | Author: Nicholas Sparks <>
277 | Date: Wed Dec 12 16:43:52 2018 -0500
278 |
279 |     updated parser
280 |
281 | commit 1c6febfb31e8b71d9528d30fe87ebcf79ef52a830
282 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
282 |     ↪ com>
283 | Date: Tue Dec 11 13:13:40 2018 -0500
284 |
285 |     matrix passes scanning
286 |
287 | commit e8edc019a559921652c6bcc78fdee00b1682f598
288 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
288 |     ↪ com>
289 | Date: Sun Dec 9 16:08:41 2018 -0500
290 |
291 |     fixed: compiler now passes microC tests

```

```

292
293 commit 989288cc876c24a2eef5d4d6ff3f06ba475ab3ad
294 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
295     ↳ com>
296 Date:   Tue Dec 4 20:41:31 2018 -0500
297
298     now builds
299
300 commit 4c893ad7fe017ace53d22a9d9ea88842335360d4
301 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
302     ↳ com>
303 Date:   Sun Dec 2 23:31:24 2018 -0500
304
305     Almost builds on make
306
307 commit 248f5f4fb0b2c1fde1f1d40eb561e93ddb8aee2e
308 Author: Nicholas Sparks <>
309 Date:   Sat Dec 1 20:02:56 2018 -0500
310
311     Issues with parser, but progress on codegen
312
313 commit e7ee115f65356ec5b66e488dabbe3bcd579e6eec
314 Author: Nicholas Sparks <>
315 Date:   Fri Nov 30 22:11:41 2018 -0500
316
317     Updated sast, semant. codegen. unbound 'm' Line 146:codegen.ml
318
319 commit 12c7590ba13f2b80c1ec544c05d576e9ce39a8a0
320 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
321     ↳ com>
322 Date:   Thu Nov 29 22:33:02 2018 -0500
323
324     Added SMatrixLit processing to codegen expr-builder
325
326 commit 2d94ca0f587a9b8bf7536feffd181cad32516775
327 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
328     ↳ com>
329 Date:   Tue Nov 27 11:01:26 2018 -0500
330
331     added printm for matrix to codegen
332
333 commit ddfea1e7eaa53891f0365dad251b9f124bf8cbd7
334 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
335     ↳ com>
336 Date:   Mon Nov 26 17:55:51 2018 -0500
337
338     Started linking C lib to codegen
339
340 commit 2380b2ffafb03a2ecc5593062950ee3e57462e8
341 Author: Nicholas Sparks <>
342 Date:   Thu Nov 22 01:10:15 2018 -0500
343
344     AST/SAST/SEMANT Updated
345
346 commit c58cc6b578b2437d50349830a1bfc5f90d728307
347 Merge: ddad05a 405896e
348 Author: Nicholas Sparks <>
349 Date:   Sun Nov 18 05:48:05 2018 -0500
350
351     Merge branch 'master' of https://github.com/PearceKieser/CS4115
352
353 commit ddad05a41d7e3266a445a7baf60bd41c718e43b7

```

```

349 Author: Nicholas Sparks <>
350 Date: Sun Nov 18 05:47:03 2018 -0500
351
352     Not working , but getting closer .
353
354 commit 405896ea46a2e8e539c5ba72e833944667ccd09a
355 Merge: 3d32fb3 08044f6
356 Author: Julia Sheth <jnsheth@me.com>
357 Date: Sat Nov 17 16:57:17 2018 -0500
358
359     Fixed merge conflicts in matrix.c
360
361 commit 3d32fb312581ab263bb57a416cc61262c8b112f1
362 Author: Julia Sheth <jnsheth@me.com>
363 Date: Sat Nov 17 16:55:18 2018 -0500
364
365     Finished writing inverse function for matrices
366
367 commit 08044f6c5c57c51d05a1120a28f31664f68b527b
368 Author: Katie Pfleger <kjp2157@columbia.edu>
369 Date: Sat Nov 17 16:39:44 2018 -0500
370
371     Completed matrix mult and scalar mult functions for matrix
372
373 commit 995a9f31e837b801af72c1ff68a0043029b15f6d
374 Author: Katie Pfleger <kjp2157@columbia.edu>
375 Date: Sat Nov 17 16:27:20 2018 -0500
376
377     Completed add() function for matrix
378
379 commit 9703570bd0c19f1f68e0e9904eabd75522e6282a
380 Merge: b478e2b e2c3297
381 Author: Katie Pfleger <kjp2157@columbia.edu>
382 Date: Sat Nov 17 16:08:05 2018 -0500
383
384     Fixed merge conflicts for matrix.c
385
386 commit b478e2bf1ddd8bf2d6c547a03b500e71ef21db77
387 Author: Katie Pfleger <kjp2157@columbia.edu>
388 Date: Sat Nov 17 16:01:33 2018 -0500
389
390     Finished determinant function and tests
391
392 commit e2c3297ddd1e83c75389d22f691a725940579516
393 Author: Julia Sheth <jnsheth@me.com>
394 Date: Sat Nov 17 16:01:20 2018 -0500
395
396     Finished writing transpose function for matrices
397
398 commit 2ca95562988371f81c6694fb0c2e401dc36bf60d
399 Merge: 8f58e1b defb6ff
400 Author: Julia Sheth <jnsheth@me.com>
401 Date: Sat Nov 17 15:52:40 2018 -0500
402
403     Fixed merge conflicts in matrix.c file
404
405 commit 8f58e1bffd811fcfd6134d52228256b8385f206b1
406 Author: Julia Sheth <jnsheth@me.com>
407 Date: Sat Nov 17 15:50:35 2018 -0500
408
409     Wrote tranpose function for matrices
410

```

```

411 commit defb6ffbafc517f1b0f21056c96727222662b827
412 Merge: bd2ae8e 7e4dfd5
413 Author: Katie Pfleger <kjp2157@columbia.edu>
414 Date:   Sat Nov 17 15:49:47 2018 -0500
415
416     Added print function for matrix
417
418 commit bd2ae8e4562dcfad02bf037cbf0c8ec382f60cf1
419 Author: Katie Pfleger <kjp2157@columbia.edu>
420 Date:   Sat Nov 17 15:41:24 2018 -0500
421
422     Continued work on determinant
423
424 commit 7e4dfd53d08351f5a980e82aabe6dfe16860cd92
425 Author: Julia Sheth <jnsheth@me.com>
426 Date:   Sat Nov 17 15:36:53 2018 -0500
427
428     Finished dot product function and added check for null matrices
429     ↪ in constructor
430
431 commit c09e7801fa633e49cf175642b7e98dc6495ee793
432 Merge: aa30f87 5b213c2
433 Author: Julia Sheth <jnsheth@me.com>
434 Date:   Sat Nov 17 15:01:00 2018 -0500
435
436     Merge branch 'master' of https://github.com/PearceKieser/CS4115
437
438 commit aa30f87db4a193c54a40e831a4c0f72748772fa4
439 Author: Julia Sheth <jnsheth@me.com>
440 Date:   Sat Nov 17 14:59:43 2018 -0500
441
442     Writing dot product function for matrices
443
444 commit 5b213c2048b35e2f613bd998fded80362bfd8283
445 Merge: 1904472 58e1754
446 Author: Katie Pfleger <kjp2157@columbia.edu>
447 Date:   Sat Nov 17 14:58:44 2018 -0500
448
449     Fixed merge conflicts
450
451 commit 19044722fcb65152a382d9168030886da2fcfd02
452 Author: Katie Pfleger <kjp2157@columbia.edu>
453 Date:   Sat Nov 17 14:57:33 2018 -0500
454
455     Fixed initMatrx and started determinant
456
457 commit 58e17547ed9aeba6fe3ae003dac2842ed7d342eb
458 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
        ↪ com>
459 Date:   Sat Nov 17 14:46:33 2018 -0500
460
461     added matrix.c/die()
462
463 commit 304cfe04f65981f849a42f766e63d0fb5eb588cb
464 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
        ↪ com>
465 Date:   Sat Nov 17 14:32:03 2018 -0500
466
467     Added more functions to matrix.c
468
469 commit e4faeeb4bc95fd59000b923b0ffed20979843bf7

```

```

469 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
470 |   ↳ com>
470 | Date:   Wed Nov 14 20:11:00 2018 -0500
471 |
472 |     added matrix.c
473 |
474 | commit f44b2e1ffed885caf113fc0b3b9871230d0db39d
475 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
475 |   ↳ com>
476 | Date:   Wed Nov 14 06:36:18 2018 -0500
477 |
478 |     added helloWorldSolution
479 |
480 | commit 4077f83600191c36c28dd142a1dcacbbda252a7b
481 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
481 |   ↳ com>
482 | Date:   Tue Nov 13 18:37:59 2018 -0500
483 |
484 |     missed a file
485 |
486 | commit 7f5d86ac73aee037dc240a3a303b69faf1f30456
487 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
487 |   ↳ com>
488 | Date:   Tue Nov 13 18:36:26 2018 -0500
489 |
490 |     more ast matrix fixes
491 |
492 | commit 530f0fca305b72cc5ab9519c32959f331fb7c569
493 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
493 |   ↳ com>
494 | Date:   Tue Nov 13 18:31:29 2018 -0500
495 |
496 |     fix Matrix type constructor
497 |
498 | commit da114fc17c4762cfedd63e1dbfb69590470a7b95
499 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
499 |   ↳ com>
500 | Date:   Tue Nov 13 18:23:51 2018 -0500
501 |
502 |     Working on building code
503 |
504 | commit c74b35cfedb0b0b239723a6d1925c7b4e7c271f5
505 | Author: alana anderson <afa2132@columbia.edu>
506 | Date:   Tue Nov 13 18:11:09 2018 -0500
507 |
508 |     added string to function decls and expressions
509 |
510 | commit 46297bf266d692e770b3604b83abd6c1e34730ef
511 | Author: alana anderson <afa2132@columbia.edu>
512 | Date:   Tue Nov 13 18:00:18 2018 -0500
513 |
514 |     added string types
515 |
516 | commit a39d41aebf5c80f004b8346426d21201541b6022
517 | Author: Nicholas Sparks <43015596+ns3284@users.noreply.github.com>
518 | Date:   Tue Nov 13 04:28:17 2018 +0000
519 |
520 |     Updated ast, sast, semant
521 |
522 | commit 52d658edd9d27175e7536aceb6e1d1d900c2cec3
523 | Author: Pearce Kieser <5055971+Pearcekieser@users.noreply.github.
523 |   ↳ com>
```

```

524 Date: Mon Nov 12 20:47:51 2018 -0500
525     added Matrix to AST
527
528 commit 6e3321cb660be303c58d6c86ebc398e50f950740
529 Author: alana anderson <afa2132@columbia.edu>
530 Date: Mon Nov 12 20:47:23 2018 -0500
531
532     updating gitignor
533
534 commit 143dd92460eb6cfc39d46e6fe3a068a5c7ee6db
535 Author: ns3284@columbia.edu <43015596+ns3284@users.noreply.github.
      ↪ com>
536 Date: Tue Nov 13 00:09:25 2018 +0000
537
538     Updated Parser
539
540 commit aab164a9320b49ae525386d232b9d9bd6a9d2f15
541 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
      ↪ com>
542 Date: Sun Nov 11 17:04:49 2018 -0500
543
544     codgen with matrix
545
546 commit a510db3e7948f7106ed4dbaf1906a389a5d1380c
547 Author: Nicholas Sparks <ns3284@columbia.edu>
548 Date: Sat Nov 10 02:55:31 2018 -0500
549
550     Updating Tokenizer, Parser, AST, SAST
551
552 commit d50075abc51fabdb312582cf865e4bfe5d02fce9
553 Author: alana anderson <afa2132@columbia.edu>
554 Date: Fri Nov 9 08:17:38 2018 -0500
555
556     took string out of parser
557
558 commit ec252eb258c583c4494e5063c9bb18b4a40971f1
559 Author: Julia Sheth <jnsheth@me.com>
560 Date: Thu Nov 8 16:58:03 2018 -0500
561
562     Rewriting parser so that it conforms more to MicroC and
      ↪ starting to include matrix functionality
563
564 commit 003aa4c9f9c6b1819f6bfebe42513872220125be
565 Author: Pearce Kieser <5055971+PearceKieser@users.noreply.github.
      ↪ com>
566 Date: Thu Nov 8 10:48:07 2018 -0500
567
568     reorganize code and added todo list
569
570 commit 086b5a305d53cdcaa6091ec4b746a099e8666b746
571 Author: alana anderson <afa2132@columbia.edu>
572 Date: Wed Nov 7 18:12:50 2018 -0500
573
574     wrote basic top-level
575
576 commit 047fb1089892322c5450c035c49f3913c5f8080a
577 Author: Nicholas Sparks <ns3284@columbia.edu>
578 Date: Mon Oct 15 14:21:05 2018 -0400
579
580     forgot ID.method() and nested bool_seqs
581

```

```

582 commit 5a9ed02444ebacb8795b774b35b267fefab1fe3d
583 Author: Nicholas Sparks <ns3284@columbia.edu>
584 Date:   Sat Oct 13 23:52:37 2018 -0400
585     int -> ref
587
588 commit a7e4d7a3d4c2419d8f130e1175456a040c3cce4c
589 Author: Nicholas Sparks <ns3284@columbia.edu>
590 Date:   Sat Oct 13 23:47:48 2018 -0400
591
592     Can't forgot conditionals
593
594 commit 8753f9620167291e807ab54709217130259405ba
595 Author: Nicholas Sparks <ns3284@columbia.edu>
596 Date:   Sat Oct 13 23:15:15 2018 -0400
597
598     Corrected the type of MATRIX
599
600 commit b2fdccb25c2f35912c332019a4e00bcd5ecd991
601 Author: Nicholas Sparks <ns3284@columbia.edu>
602 Date:   Sat Oct 13 23:11:01 2018 -0400
603
604     Parser compiles without warning. Untested.
605
606 commit 293362985ffa2bec765583dbb4af06d061e748f5
607 Author: Pearce kieser <5055971+Pearcekieser@users.noreply.github.
       ↳ com>
608 Date:   Fri Oct 12 13:08:16 2018 -0400
609
610     Added MicroC
611
612 commit 09c39facf88a8584bfe8d2d1fbfffc62a8bd4300
613 Author: Nicholas Sparks <ns3284@columbia.edu>
614 Date:   Mon Oct 8 00:32:42 2018 -0400
615
616     Updated tokenizer , parser.mly
617
618 commit dc28500c9edf0eb904b648cb232113eaa9d49552
619 Author: Pearce kieser <5055971+Pearcekieser@users.noreply.github.
       ↳ com>
620 Date:   Sat Oct 6 19:09:33 2018 -0400
621
622     tokenizer added
623
624 commit 979f256397eee25bd4d245bc64453f54a79bb739
625 Author: ns3284@columbia.edu <ns3284@columbia.edu>
626 Date:   Sun Oct 7 06:04:42 2018 -0400
627
628     Added conditions , and scope
629
630 commit b1d613d76048b83ce6f579e810547ba818c41d30
631 Author: Pearce kieser <5055971+Pearcekieser@users.noreply.github.
       ↳ com>
632 Date:   Sat Oct 6 13:30:45 2018 -0400
633
634     added calc.ml
635
636 commit 3a5d9a9ae33a81ce3f742bc14d5298e37982b471
637 Author: Pearcekieser <5055971+Pearcekieser@users.noreply.github.com
       ↳ >
638 Date:   Sat Oct 6 13:14:40 2018 -0400
639

```

```

640     Add files via upload
641
642 commit 6fb63efd49502f0cc5cb4bd455ae91e9d2ad7219
643 Author: Pearcekieser <5055971+Pearcekieser@users.noreply.github.com
644     ↪ >
644 Date:   Sat Sep 15 17:39:10 2018 -0400
645
646     Update README.md
647
648 commit 95b95fe80e609ae999428fe280855b924a4eb7d7
649 Author: Pearcekieser <5055971+Pearcekieser@users.noreply.github.com
649     ↪ >
650 Date:   Sat Sep 15 17:39:00 2018 -0400
651
652     Initial commit

```

8.1.2 scanner.mll

```

1 (* Ocamlex scanner for Matrx *)
2 {
3     module Lex = Lexing
4     module Buf = Buffer
5
6 let sprintf = Printf sprintf
7
8 let position lexbuf =
9     let p = lexbuf.Lex.lex_curr_p in
10    sprintf "%s:%d:%d"
11    p.Lex.pos_fname p.Lex.pos_lnum (p.Lex.pos_cnum - p.
12        ↪ Lex.pos_bol)
13
14 let set_filename (fname:string) (lexbuf:Lex.lexbuf) =
15     ( lexbuf.Lex.lex_curr_p <-
16         { lexbuf.Lex.lex_curr_p with Lex.pos_fname = fname }
17     ; lexbuf
18 )
19
20 open Parser
21
22 let digit = ['0' - '9']
23 let digits = digit+
24
25 rule token = parse
26   [ ' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
27 | /*/* { comment lexbuf } (* Comments *)
28 | '(' { LPAREN }
29 | ')' { RPAREN }
30 | '{' { LBRACE }
31 | '}' { RBRACE }
32 | '[' { LBRACK }
33 | ']' { RBRACK }
34 | ';' { SEMI }
35 | ',' { COMMA }
36 | '+' { PLUS }

```

```

37 |   '-'      { MINUS }
38 |   '*'      { TIMES }
39 |   '/'      { DIVIDE }
40 |   '='      { ASSIGN }
41 |   "=="     { EQ }
42 |   "!="     { NEQ }
43 |   '<'      { LT }
44 |   "<="     { LEQ }
45 |   ">"      { GT }
46 |   ">="     { GEQ }
47 |   "&&"     { AND }
48 |   "||"      { OR }
49 |   "!"      { NOT }
50 |   "if"      { IF }
51 |   "else"    { ELSE }
52 |   "for"     { FOR }
53 |   "while"   { WHILE }
54 |   "return"  { RETURN }
55 |   "int"     { INT }
56 |   "bool"    { BOOL }
57 |   "float"   { FLOAT }
58 |   "string"  { STRING }
59 |   "void"    { VOID }
60 |   "matrix"  { MATRIX }
61 |   ""        { STRINGLIT ( string (Buf.create 100) lexbuf ) }
62 |   '\'       { CHARLIT ( char (Buf.create 100) lexbuf) }
63 |   "true"    { BLIT(true)  }
64 |   "false"   { BLIT(false) }
65 |   digits as lxm { LITERAL(int_of_string lxm) }
66 |   digits '.' digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm
   ↪ { FLOATLIT(lxm) }
67 |   ['a'-'z' 'A'-'Z'][ 'a'-'z' 'A'-'Z' '0'-'9' '_']*      as lxm
   ↪ { ID(lxm) }
68 |   eof { EOF }
69 |   _ as char { raise (Failure("illegal character " ^ Char.
   ↪ escaped char)) }
70
71 and string buf = parse
72 | [^'', '\n', '\\']+ as content { Buf.add_string buf content
   ↪ ; string buf lexbuf }
73 | '\n'          { Buf.add_string buf "\n"; Lex.new_line lexbuf;
   ↪ string buf lexbuf }
74 | '\\', '"'    { Buf.add_char buf"'; string buf lexbuf }
75 | '\\', '\''   { Buf.add_char buf'\''; string buf lexbuf }
76 | '"'          { Buf.contents buf } (* return *)
77
78 and char buf = parse
79 | [^'\\'] as content '\\' { content }
80 | '\\', '\\', '\\' { '\\' }
81 | '\\', '?', '\\' { '?' }
82 | '\\', '\\', '\\' { '\\' }
83 | '\\', '\\', '\\' { '\\' }
84 | '\\', 'a', '\\' { Char.chr 7 }
85 | '\\', 'b', '\\' { Char.chr 8 }

```

```

86 | '\\', 'e', '\\', { Char.chr 27 }
87 | '\\', 'f', '\\', { Char.chr 12 }
88 | '\\', 'n', '\\', { Char.chr 10 }
89 | '\\', 'r', '\\', { Char.chr 13 }
90 | '\\', 't', '\\', { Char.chr 9 }
91 | '\\', 'v', '\\', { Char.chr 11 }

92
93
94 and comment = parse
95   /* { token lexbuf }
96 | _ { comment lexbuf }

```

8.1.3 parser.mly

```

1  %{
2    open Ast
3  %}
4
5  %token MATRIX STRING CHAR LBRACK RBRACK
6  %token FLOAT /* STRING */
7  %token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
8  %token PLUS MINUS TIMES DIVIDE ASSIGN NOT
9  %token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
10 %token RETURN IF ELSE FOR WHILE INT BOOL VOID
11 %token <int> LITERAL
12 %token <bool> BLIT
13 %token <string> ID FLOATLIT
14 %token <string> STRINGLIT
15 %token <char> CHARLIT
16 %token EOF
17
18 %start program
19 %type <Ast.program> program
20
21 %nonassoc NOELSE
22 %nonassoc ELSE
23 %right ASSIGN
24 %left OR
25 %left AND
26 %left EQ NEQ
27 %left LT GT LEQ GEQ
28 %left PLUS MINUS
29 %left TIMES DIVIDE
30 %right NOT
31
32 %%
33
34
35 program:
36   decls EOF { 1 decls/* nothing */ [] , [] | decls vdecl ((2 :: fst1),
            snd 1)|decls fdecl(fst1, (2 :: snd1)) fdecl:typ ID LPAREN
            formals,opt RPAREN LBRACE vdecl:liststmt1,ist RBRACE typ =1; fname =
            2; formals = List.rev4;locals = List.rev 7;body = List.rev8

```

```

formalsopt : /* nothing */ []|formal1list1 formallist :
typID[(1,2,Noexpr)]|formal1listCOMMA typID(3,4,Noexpr) ::1 typ:INT
Int | BOOL Bool | FLOAT Float | STRING String | CHAR Char |
MATRIX Matrix | VOID Void vdecl1list : /* nothing */ []|vdecl1listvdecl2
:: 1vdecl : typIDSEMI(1, 2, Noexpr)|typIDASSIGNexprSEMI(1,
2, Asn(2,4))stmt1list : /* nothing */ []|stmt1liststmt2 :::
1stmt : exprSEMIExpr(1) | RETURN exproptSEMIReturn(2) | LBRACE
stmt1listRBRACEBlock(List.rev2) | IF LPAREN expr RPAREN stmt | IF
LPAREN expr RPAREN stmt ELSE stmt If(3,5,
7)|FORLPARENexproptSEMIexprSEMIexproptRPARENstmtFor(3,
5,7, 9)|WHILELPARENexprRPARENstmtWhile(3,
5)expropt : /* nothing */ /Noexpr|expr1 expr:LITERAL
Literal(1)|FLOATLITFliteral(1) | BLIT
BoolLit(1)|CHARLITCharLit(1) | STRINGLIT
StringLit(1)|TRUEBoolLit(true)|FALSEBoolLit(false)|IDId(1) /* |
ID matrixaddrGetMatrixVal(1,2) */ /*
|IDmatrixaddrASSIGNexprSetMatrixVal(1,2,4) */ | LBRACK
matrixvalueRBRACKMatrixLit(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)/* handles assignment */ |IDLPARENseqoptRPARENCall(1,
3)|LPARENexprRPAREN2 seqopt : /* nothing */ []|seqList.rev1
seq:expr [1]|seqCOMMAexpr3 :::
1matrixvalue:LBRACKseqRBRACK[MatrixLit(List.rev2)] | LBRACK
seq RBRACK COMMA
matrixvalueMatrixLit(List.rev2)::5|LBRACKseqRBRACKmatrixvalueMatrixLit(List.rev2)::4

```

8.1.4 ast.ml

```

1 (* Abstract Syntax Tree and functions for printing it *)
2
3 type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq
4   ↪ | Greater | Geq |
5     And | Or
6
7 type uop = Neg | Not
8
9 type expr =
10   Literal of int
11   | Fliteral of string
12   | BoolLit of bool
13   | CharLit of char
14   | StringLit of string
15   | Id of string
16   | Binop of expr * op * expr
17   | Unop of uop * expr
18   | Asn of string * expr
19   | Call of string * expr list
20   | Noexpr
21   | MatrixLit of expr list
22
23 type typ = Int | Bool | Float | Void | String | Char |
   ↪ Matrix

```

```

24 type bind = typ * string * expr
25
26 type stmt =
27     Block of stmt list
28 | Expr of expr
29 | Return of expr
30 | If of expr * stmt * stmt
31 | For of expr * expr * expr * stmt
32 | While of expr * stmt
33
34 type func_decl = {
35     typ : typ;
36     fname : string;
37     formals : bind list;
38     locals : bind list;
39     body : stmt list;
40 }
41
42 type program = bind list * func_decl list
43
44 (* Pretty-printing functions *)
45
46 let string_of_op = function
47     Add -> "+"
48 | Sub -> "-"
49 | Mult -> "*"
50 | Div -> "/"
51 | Equal -> "=="
52 | Neq -> "!="
53 | Less -> "<"
54 | Leq -> "<="
55 | Greater -> ">"
56 | Geq -> ">="
57 | And -> "&&"
58 | Or -> "||"
59
60 let string_of_uop = function
61     Neg -> "-"
62 | Not -> "!"
63
64 let rec string_of_expr = function
65     Literal(l) -> string_of_int l
66 | Fliteral(l) -> l
67 | BoolLit(true) -> "true"
68 | BoolLit(false) -> "false"
69 | CharLit(l) -> Char.escaped l
70 | StringLit(l) -> l
71 | MatrixLit(l) -> "matrixLit[" ^ String.concat ", " (List.
72     ↪ map string_of_expr l) ^ "]"
73 | Id(s) -> s
74 | Binop(e1, o, e2) ->
75     string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
    ↪ string_of_expr e2
    ↪ string_of_expr e2
75 | Unop(o, e) -> string_of_uop o ^ string_of_expr e

```

```

76 | Asn(v, e) -> v ^ " = " ^ string_of_expr e
77 | Call(f, el) ->
78   f ^ "(" ^ String.concat ", " (List.map string_of_expr
79     ↪ el) ^ ")"
80 | Noexpr -> ""
81
81 let rec string_of_stmt = function
82   Block(stmts) ->
83     "{\n" ^ String.concat "" (List.map string_of_stmt
84       ↪ stmts) ^ "}\n"
85   | Expr(expr) -> string_of_expr expr ^ ";\n";
86   | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
87   | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n"
88     ↪ ^ string_of_stmt s
89   | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n"
90     ↪ string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
91   | For(e1, e2, e3, s) ->
92     "for (" ^ string_of_expr e1 ^ " ; " ^ string_of_expr
93       ↪ e2 ^ " ; " ^
94         string_of_expr e3 ^ ") " ^ string_of_stmt s
95   | While(e, s) -> "while (" ^ string_of_expr e ^ " ) " ^
96     ↪ string_of_stmt s
97
98 let string_of_typ = function
99   Int -> "int"
100  | Bool -> "bool"
101  | Float -> "float"
102  | String -> "string"
103  | Char -> "char"
104  | Matrix -> "matrix"
105  | Void -> "void"
106
107 let string_of_vdecl (t, id, _) = string_of_typ t ^ " " ^ id
108   ↪ ^ ";\n"
109
110 let string_of_fdecl fdecl =
111   string_of_typ fdecl.typ ^ " " ^
112   fdecl.fname ^ "(" ^ String.concat ", " (List.map (fun (_,
113     ↪ vName, _) -> vName) fdecl.formals) ^
114     "){\n" ^
115     String.concat "" (List.map string_of_vdecl fdecl.locals) ^
116     String.concat "" (List.map string_of_stmt fdecl.body) ^
117     "}\n"
118
119 let string_of_program (vars, funcs) =
120   let f' = List.rev funcs in
121   String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
122   String.concat "\n" (List.map string_of_fdecl f')

```

8.1.5 sast.ml

```

1 (* Semantically-checked Abstract Syntax Tree and functions
2    ↪ for printing it *)

```

```

2  open Ast
3
4
5  type sexpr = typ * sx
6  and sx =
7      SLiteral of int
8      | SFLiteral of string
9      | SBoolLit of bool
10     | SCharLit of char
11     | SStringLit of string
12     | SId of string
13     (* list of contents, number of rows then cols *)
14     | SMatrixLit of sexpr list * int * int
15     | SBinop of sexpr * op * sexpr
16     | SUNop of uop * sexpr
17     | SAsn of string * sexpr
18     | SCall of string * sexpr list
19     | SNoexpr
20
21 type sbind = typ * string * sexpr
22
23 type sstmt =
24     SBlock of sstmt list
25     | SExpr of sexpr
26     | SReturn of sexpr
27     | SIf of sexpr * sstmt * sstmt
28     | SFor of sexpr * sexpr * sexpr * sstmt
29     | SWhile of sexpr * sstmt
30
31 type sfunc_decl = {
32     styp : typ;
33     sfname : string;
34     sformals : sbind list;
35     slocals : sbind list;
36     sbody : sstmt list;
37 }
38
39 type sprogram = bind list * sfunc_decl list
40
41 (* Pretty-printing functions *)
42
43 let rec string_of_expr (t, e) =
44   "(" ^ string_of_typ t ^ " : " ^ (match e with
45     SLiteral(l) -> string_of_int l
46     | SBoolLit(true) -> "true"
47     | SBoolLit(false) -> "false"
48     | SFLiteral(l) -> l
49     | SCharLit(l) -> Char.escaped l
50     | SStringLit(l) -> l
51     | SMatrixLit(l, r, c) -> "rows: " ^ string_of_int r ^ ", "
52       ↪ cols: " ^ string_of_int c ^ " : [" ^ String.concat "
53       ↪ , " (List.map string_of_expr l) ^ "]"
54     | SId(s) -> s
55     | SBinop(e1, o, e2) ->

```

```

54     string_of_sexp e1 ^ " " ^ string_of_op o ^ " " ^
55         ↪ string_of_sexp e2
56 | SUNop(o, e) -> string_of_uop o ^ string_of_sexp e
57 | SAsn(v, e) -> v ^ " = " ^ string_of_sexp e
58 | SCall(f, el) ->
59     f ^ "(" ^ String.concat ", " (List.map string_of_sexp
60         ↪ el) ^ ")"
61 | SNoexpr -> ""
62     ) ^ ")"
63
64 let rec string_of_sstmt = function
65   SBlock(stmts) ->
66     "{\n" ^ String.concat "" (List.map string_of_sstmt
67         ↪ stmts) ^ "}\n"
68 | SExpr(expr) -> string_of_sexp expr ^ ";\n";
69 | SReturn(expr) -> "return " ^ string_of_sexp expr ^ ";\n
70     ↪ ";
71 | SIf(e, s, SBlock([])) ->
72     "if (" ^ string_of_sexp e ^ ")\n" ^ string_of_sstmt s
73 | SIf(e, s1, s2) -> "if (" ^ string_of_sexp e ^ ")\n" ^
74     string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
75 | SFor(e1, e2, e3, s) ->
76     "for (" ^ string_of_sexp e1 ^ " ; " ^
77         ↪ string_of_sexp e2 ^ " ; " ^
78         string_of_sexp e3 ^ ") " ^ string_of_sstmt s
79 | SWhile(e, s) -> "while (" ^ string_of_sexp e ^ ") " ^
80         ↪ string_of_sstmt s
81
82 let string_of_sfdecl fdecl =
83   string_of_typ fdecl.styp ^ " " ^
84   fdecl.sfname ^ "(" ^ String.concat ", " (List.map (fun (_,
85         ↪ vName, _) -> vName) fdecl.sformals) ^
86   ")\n{\n" ^
87   String.concat "" (List.map string_of_vdecl fdecl.slocals)
88   ↪ ^
89   String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
90   "}\n"
91
92 let string_of_sprogram (vars, funcs) =
93   let f' = List.rev funcs in
94   String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
95   String.concat "\n" (List.map string_of_sfdecl f')

```

8.1.6 semant.ml

```

1 (* Semantic checking for the Matrx compiler *)
2
3 open Ast
4 open Sast
5
6 module StringMap = Map.Make(String)
7
8

```

```

9  (* Semantic checking of the AST. Returns an SAST if
10   ↪ successful,
11   throws an exception if something is wrong.
12
13   Check each global variable, then check each function *)
14 let check (globals, functions) =
15
16   (* Verify a list of bindings has no void types or
17    ↪ duplicate names *)
18   let check_binds (kind : string) (binds : bind list) =
19     List.iter (function
20       (Void, b, _) -> raise (Failure ("illegal void " ^ kind ^ "
21         ↪ " ^ b))
22       | _ -> () ) binds;
23     let rec dups = function
24       [] -> ()
25       | ((_,n1,_): (_ ,n2,_) :: _) when n1 = n2 ->
26         raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
27       | _ :: t -> dups t
28       in dups (List.sort (fun (_,a,_) (_ ,b,_) -> compare a b)
29         ↪ binds)
30     in
31
32   (**** Check global variables ****)
33
34   (**** Check functions ****)
35
36   (* Collect function declarations for built-in functions:
37    ↪ no bodies *)
38   let built_in_decls =
39     let add_bind map (name, ty, ret) = StringMap.add name {
40       typ = ret;
41       fname = name;
42       formals =
43         (let rec create_ty_list = (function
44           [] -> []
45           | hd::tl -> (hd, "x", Noexpr)::(create_ty_list tl))
46           in create_ty_list ty);
47       locals = []; body = [] } map
48     in List.fold_left add_bind StringMap.empty [
49       ("print", [ Int ], Void),
50         ↪ ("printstr", [ String ], Void);
51         ("printm", [ Matrix ], Void);
52         ("printb", [ Bool ], Void);
53         ("printf", [ Float ], Void);
54         ("length", [ String ], Int);
55         ("transpose", [ Matrix ], Matrix
56           ↪ );
57         ("matmult", [ Matrix; Matrix ],
58           ↪ Matrix);
```

```

54          ("matadd", [Matrix; Matrix],
55           ↪ Matrix);
56          ("dot", [Matrix; Matrix],
57           ↪ Matrix);
58          ("matscale", [Matrix; Int],
59           ↪ Matrix);
60          ("det", [Matrix; Int], Int);
61          ("get_char", [String; Int],
62           ↪ String);
63          ("sconcat", [String; String],
64           ↪ String);
65          ("sequals", [String; String],
66           ↪ Bool);
67          ("substring", [String; Int;
68           ↪ Int], String)]
69      in
70
71      (* Add function name to symbol table *)
72      let add_func map fd =
73        let built_in_err = "function " ^ fd.fname ^ " may not be
74          ↪ defined"
75        and dup_err = "duplicate function " ^ fd.fname
76        and make_err er = raise (Failure er)
77        and n = fd.fname (* Name of the function *)
78        in match fd with (* No duplicate functions or
79          ↪ redefinitions of built-ins *)
80          _ when StringMap.mem n built_in_decls -> make_err
81            ↪ built_in_err
82          | _ when StringMap.mem n map -> make_err dup_err
83          | _ -> StringMap.add n fd map
84      in
85
86      (* Collect all function names into one symbol table *)
87      let function_decls = List.fold_left add_func
88        ↪ built_in_decls functions
89      in
90
91      let _ = find_func "main" in (* Ensure "main" is defined *)
92
93      let check_function func =
94        (* Make sure no formals or locals are void or duplicates
95         ↪ *)
96        check_binds "formal" func.formals;
97        check_binds "local" func.locals;
98
99      (* Raise an exception if the given rvalue type cannot be
100         ↪ assigned to

```

```

94      the given lvalue type *)  

95 let check_assign lvaluet rvaluet err =  

96     if lvaluet = rvaluet then lvaluet else raise (Failure  

97         ↪ err)  

98 in  

99  

100    let rec get_dims = function  

101        MatrixLit l -> List.length l :: get_dims (List.hd l)  

102        | _ -> []  

103 in  

104    (* Raise an exception if dimensions of Matrix are not  

105       ↪ balanced *)  

106    let rec flatten d = function  

107        [] -> []  

108        | MatrixLit hd::tl -> if List.length hd != List.hd d  

109            ↪ then raise (Failure("Invalid dims")) else List.  

110            ↪ append (flatten (List.tl d) hd) (flatten d tl)  

111        | a -> a  

112 in  

113    (* Build local symbol table of variables for this  

114       ↪ function *)  

115    let symbols = List.fold_left (fun m (ty, name, _) ->  

116        ↪ StringMap.add name ty m)  

117            StringMap.empty (globals @ func.formals @  

118                ↪ func.locals )  

119 in  

120  

121    (* Return a variable from our local symbol table *)  

122    let type_of_identifier s =  

123        try StringMap.find s symbols  

124        with Not_found -> raise (Failure ("undeclared  

125            ↪ identifier " ^ s))  

126 in  

127  

128    (* Return a semantically-checked expression, i.e., with  

129       ↪ a type *)  

130    let rec expr = function  

131        Literal l -> (Int, SLiteral l)  

132        | Fliteral l -> (Float, SFliteral l)  

133        | BoolLit l -> (Bool, SBoolLit l)  

134        | CharLit l -> (Char, SCharLit l)  

135        | StringLit l -> (String, SStringLit l)  

136        | MatrixLit l ->  

137            let d = get_dims (MatrixLit l) in  

138            let rec all_match = function  

139                [] -> ignore()  

140                | hd::tl -> if tl != [] then  

141                    let (t1, _) = expr hd in let (t2,  

142                        ↪ _) = expr (List.hd tl) in  

143                        if t1 = t2 then all_match tl else  

144                            ↪ raise (Failure ("Data  

145                                ↪ Mismatch in MatrixLit: " ^  

146                                ↪ string_of_typ t1 ^ " does

```

```

135                               ↪ not match " ^ string_of_typ
136                               ↪ t2))
137                           else ignore()
138                           in
139                           all_match l;
140                           if List.length d > 2 then (Matrix, SMatrixLit ((  

141                               ↪ List.map expr l), List.hd d, List.hd (List.  

142                               ↪ tl d)))
143                           else if List.length d = 2 then (Matrix, SMatrixLit  

144                               ↪ ( (List.map expr (flatten (List.tl d) l)),  

145                               ↪ List.hd d, List.hd (List.tl d)))
146                           else if List.length d = 1 then (Matrix, SMatrixLit  

147                               ↪ ( (List.map expr (flatten (List.tl d) l)),  

148                               ↪ List.hd d, 1))
149                           else (Matrix, SMatrixLit ( (List.map expr l), 0,0)
150                               ↪ )
151 | Noexpr      -> (Void, SNoexpr)
152 | Id s        -> (type_of_identifier s, SId s)
153 | Asn(var, e) as ex ->
154     let lt = type_of_identifier var
155     and (rt, e') = expr e in
156     let err = "illegal assignment " ^ string_of_typ lt
157         ↪ " = " ^
158         string_of_typ rt ^ " in " ^ string_of_expr ex
159         in (check_assign lt rt err, SAsn(var, (rt, e')))
160 | Unop(op, e) as ex ->
161     let (t, e') = expr e in
162     let ty = match op with
163         Neg when t = Int || t = Float -> t
164         | Not when t = Bool -> Bool
165         | _ -> raise (Failure ("illegal unary operator " ^
166             string_of_uop op ^
167             ↪ string_of_typ t ^
168             " in " ^ string_of_expr ex)
169             ↪ )
170             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
            ↪ -> Bool
        | Less | Leq | Greater | Geq
            when same && (t1 = Int || t1 = Float)
            ↪ -> Bool

```

```

171      | And | Or when same && t1 = Bool -> Bool
172      | _ -> raise (
173        Failure ("illegal binary operator " ^
174          string_of_typ t1 ^ " " ^ string_of_op
175            ↪ op ^ " " ^
176              string_of_typ t2 ^ " in " ^
177                ↪ string_of_expr e))
178      in (ty, SBinop((t1, e1'), op, (t2, e2')))
179    | Call(fname, args) as call ->
180      let fd = find_func fname in
181      let param_length = List.length fd.formals in
182      if List.length args != param_length then
183        raise (Failure ("expecting " ^ string_of_int
184          ↪ param_length ^
185            " arguments in " ^
186              ↪ string_of_expr call))
187      else let check_call (ft, _) e =
188        let (et, e') = expr e in
189        let err = "illegal argument found " ^
190          ↪ string_of_typ et ^
191            " expected " ^ string_of_typ ft ^ " in " ^
192              ↪ string_of_expr e
193        in (check_assign ft et err, e')
194      in
195        let fdFormals = List.map (fun (tp, vName, _) -> (
196          ↪ tp, vName)) fd.formals in
197        let args' = List.map2 check_call fdFormals args
198        in (fd.typ, SCall(fname, args'))
199      in
200      let check_bool_expr e =
201        let (t', e') = expr e
202        and err = "expected Boolean expression in " ^
203          ↪ string_of_expr e
204        in if t' != Bool then raise (Failure err) else (t', e')
205      in
206      (* Return a semantically-checked statement i.e.
207       ↪ containing sexprs *)
208      let rec check_stmt = function
209        Expr e -> SExpr (expr e)
210        | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt
211          ↪ b1, check_stmt b2)
212        | For(e1, e2, e3, st) ->
213          SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
214        | While(p, s) -> SWhile(check_bool_expr p, check_stmt
215          ↪ s)
216        | Return e -> let (t, e') = expr e in
217          if t = func.typ then SReturn (t, e')
218          else raise (
219            Failure ("return gives " ^ string_of_typ t ^ " expected "
220              ↪ ^
221                string_of_typ func.typ ^ " in " ^ string_of_expr e))

```

```

212
213      (* A block is correct if each statement is correct and
214         ↪ nothing
215         follows any Return statement. Nested blocks are
216         ↪ flattened. *)
217     | Block sl ->
218       let rec check_stmt_list = function
219         [Return _ as s] -> [check_stmt s]
220         | Return _ :: _ -> raise (Failure "nothing may
221           ↪ follow a return")
222         | Block sl :: ss -> check_stmt_list (sl @ ss)
223           ↪ (* Flatten blocks *)
224         | s :: ss           -> check_stmt s :: ss
225           ↪ check_stmt_list ss
226         | []                 -> []
227       in SBlock(check_stmt_list sl)
228
229   in (* body of check_function *)
230   { styp = func.typ;
231     sfname = func.fname;
232     sformals = List.map (fun (t,n,v) -> (t,n, expr v))
233       ↪ func.formals;
234     slocals = List.map (fun (t,n,v) -> (t,n, expr v))
235       ↪ func.locals;
236     sbody = match check_stmt (Block func.body) with
237     SBlock(sl) -> sl
238     | _ -> raise (Failure ("internal error: block didn't
239       ↪ become a block?"))
240   }
241   in (globals, List.map check_function functions)

```

8.1.7 codegen.ml

```

1  (* Code generation: translate takes a semantically checked
2    ↪ AST and
3  produces LLVM IR
4
5  LLVM tutorial: Make sure to read the OCaml version of the
6    ↪ tutorial
7
8  Detailed documentation on the OCaml LLVM library:
9
10 http://llvm.moe/
11 http://llvm.moe/ocaml/
12
13 *)
14
15 module L = Llvm
16 module A = Ast
17 open Sast
18

```

```

19 module StringMap = Map.Make(String)
20
21 (* translate : Sast.program -> LLVM.module *)
22 let translate (globals, functions) =
23   let context = LLVM.global_context () in
24   let llmem = LLVM.MemoryBuffer.of_file "matrix.bc" in
25   let llm = LLVM_bitreader.parse_bitcode context llmem in
26
27 (* Create the LLVM compilation module into which
28    we will generate code *)
29 let the_module = LLVM.create_module context "Matrx" in
30
31 (* Get types from the context *)
32 let i32_t = LLVM.i32_type context
33 and i8_t = LLVM.i8_type context
34 and i1_t = LLVM.i1_type context
35 and float_t = LLVM.double_type context
36 and void_t = LLVM.void_type context
37 and string_t = LLVM.pointer_type (LLVM.i8_type context)
38 and matrx_t = LLVM.pointer_type (match LLVM.type_by_name llm
39   ↪ "struct.matrix" with
40     None -> raise (Failure "Missing implementation for
41       ↪ struct Matrix")
42     | Some t -> t)
43 in
44
45 (* Return the LLVM type for a Matrx type *)
46 let ltype_of_typ = function
47   | A.Int -> i32_t
48   | A.Bool -> i1_t
49   | A.Float -> float_t
50   | A.Void -> void_t
51   | A.Char -> i8_t
52   | A.String -> string_t
53   | A.Matrix -> matrx_t
54 in
55
56 (* Create a map of global variables after creating each *)
57 let global_vars : LLVM.value StringMap.t =
58   let global_var m (t, n, _) =
59     let init = match t with
60       | A.Float -> LLVM.const_float (ltype_of_typ t) 0.0
61       | _ -> LLVM.const_int (ltype_of_typ t) 0
62     in StringMap.add n (LLVM.define_global n init the_module)
63     ↪ m in
64   List.fold_left global_var StringMap.empty globals in
65
66 let printf_t : LLVM.lltype =
67   LLVM.var_arg_function_type i32_t [| LLVM.pointer_type i8_t
68   ↪ [] |] in
69 let printf_func : LLVM.value =
70   LLVM.declare_function "printf" printf_t the_module in
71
72 let printMatrix_t = LLVM.function_type i32_t [| matrx_t |] in

```

```

69 let printMatrix_f = L.declare_function "display"
    ↪ printMatrix_t the_module in
70 let matrix_init_t = L.function_type matrx_t [| i32_t ;
    ↪ i32_t |] in
71 let matrix_init_f = L.declare_function "initMatrix_CG"
    ↪ matrix_init_t the_module in
72 let store_matrix_t = L.function_type matrx_t [| matrx_t ;
    ↪ i32_t |] in
73 let store_matrix_f = L.declare_function "storeVal"
    ↪ store_matrix_t the_module in
74 let transpose_matrix_t = L.function_type matrx_t [| matrx_t ;
    ↪ |] in
75 let transpose_matrix_f = L.declare_function "transpose"
    ↪ transpose_matrix_t the_module in
76 let mult_matrix_t = L.function_type matrx_t [| matrx_t ;
    ↪ matrx_t |] in
77 let mult_matrix_f = L.declare_function "matrixMult"
    ↪ mult_matrix_t the_module in
78 let add_matrix_t = L.function_type matrx_t [| matrx_t ;
    ↪ matrx_t |] in
79 let add_matrix_f = L.declare_function "mAdd" add_matrix_t
    ↪ the_module in
80 let dot_matrix_t = L.function_type matrx_t [| matrx_t ;
    ↪ matrx_t |] in
81 let dot_matrix_f = L.declare_function "dotProduct"
    ↪ dot_matrix_t the_module in
82 let scale_matrix_t = L.function_type matrx_t [| matrx_t ;
    ↪ i32_t |] in
83 let scale_matrix_f = L.declare_function "timesScalar"
    ↪ scale_matrix_t the_module in
84 let det_matrix_t = L.function_type matrx_t [| matrx_t ;
    ↪ i32_t |] in
85 let det_matrix_f = L.declare_function "determinant"
    ↪ det_matrix_t the_module in
86
87
88 let string_get_t = L.function_type string_t [| string_t;
    ↪ i32_t |] in
89 let string_get_f = L.declare_function "string_get"
    ↪ string_get_t the_module in
90 let string_length_t = L.function_type i32_t [| string_t |]
    ↪ in
91 let string_length_f = L.declare_function "string_length"
    ↪ string_length_t the_module in
92 let string_concat_t = L.function_type string_t [| string_t
    ↪ ; string_t |] in
93 let string_concat_f = L.declare_function "string_concat"
    ↪ string_concat_t the_module in
94 let string_equals_t = L.function_type i32_t [| string_t;
    ↪ string_t |] in
95 let string_equals_f = L.declare_function "string_equals"
    ↪ string_equals_t the_module in
96 let string_substr_t = L.function_type string_t [| string_t
    ↪ ; i32_t; i32_t |] in

```

```

97 let string_substr_f = L.declare_function "string_substr"
98   ↪ string_substr_t the_module in
99
100 (* Define each function (arguments and return type) so
101    ↪ we can
102    call it even before we've created its body *)
103 let function_decls : (L.llvalue * sfunc_decl) StringMap.t
104   ↪ =
105 let function_decl m fdecl =
106   let name = fdecl.sfname
107   and formal_types =
108 Array.of_list (List.map (fun (t,_,_) -> ltype_of_typ t)
109   ↪ fdecl.sformals)
110   in let ftype = L.function_type (ltype_of_typ fdecl.
111   ↪ styp) formal_types in
112   StringMap.add name (L.define_function name ftype
113   ↪ the_module, fdecl) m in
114   List.fold_left function_decl StringMap.empty functions
115   ↪ in
116
117 (* Fill in the body of the given function *)
118 let build_function_body fdecl =
119   let (the_function, _) = StringMap.find fdecl.sfname
120   ↪ function_decls in
121   let builder = L.builder_at_end context (L.entry_block
122   ↪ the_function) in
123   let int_format_str = L.build_global_stringptr "%d\n" "
124   ↪ fmt" builder in
125   let float_format_str = L.build_global_stringptr "%g\n" "
126   ↪ fmt" builder in
127   let string_format_str = L.build_global_stringptr "%s\n" "
128   ↪ "fmt" builder in
129   (* Construct the function's "locals": formal arguments
130    ↪ and locally
131    declared variables. Allocate each on the stack,
132    ↪ initialize their
133    value, if appropriate, and remember their values in
134    ↪ the "locals" map *)
135 let local_vars =
136   let add_formal m (t, n) p =
137     L.set_value_name n p;
138   let local = L.build_alloca (ltype_of_typ t) n builder in
139     ignore (L.build_store p local builder);
140   StringMap.add n local m
141
142   (* Allocate space for any locally declared variables
143    ↪ and add the
144    * resulting registers to our map *)
145   and add_local m (t, n) =
146   let local_var = L.build_alloca (ltype_of_typ t) n builder
147   in StringMap.add n local_var m
148     in
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
857
858
859
859
860
861
862
863
864
865
866
867
867
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
937
938
939
939
940
941
942
943
944
945
946
946
947
948
948
949
950
951
952
953
954
955
956
957
957
958
959
959
960
961
962
963
964
965
966
967
967
968
969
969
970
971
972
973
974
975
975
976
977
977
978
979
979
980
981
982
983
984
985
986
986
987
988
988
989
989
990
991
992
993
994
994
995
996
996
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1730
1731
1731
1732
1732
1733
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1740
1741
1741
1742
1742
1743
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1750
1751
1751
1752
1752
1753
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1760
1761
1761
1762
1762
1763
1763
1764
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1770
1771
1771
1772
1772
1773
1773
1774
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1780
1781
1781
1782
1782
1783
1783
1784
1784
1785
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1790
1791
1791
1792
1792
1793
1793
1794
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
179
```

```

134     let sformals = List.map (fun (tp, vName, _) -> (tp,
135                                 ↪ vName)) fdecl.sformals in
136     let slocals= List.map (fun (tp, vName, _) -> (tp,
137                                 ↪ vName)) fdecl.slocals in
138     let formals = List.fold_left2 add_formal StringMap.
139                                 ↪ empty sformals
140             (Array.to_list (L.params the_function)) in
141             List.fold_left add_local formals slocals
142         in
143
144         (* Return the value for a variable or formal argument.
145            Check local names first, then global names *)
146         let lookup n = try StringMap.find n local_vars
147                         with Not_found -> StringMap.find n
148                             ↪ global_vars
149         in
150
151         (* Construct code for an expression; return its value *)
152         let rec expr_builder ((_, e) : sexpr) = match e with
153             SLiteral i -> L.const_int i32_t i
154             | SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
155             | SFliteral i -> L.const_float float_t (
156                 ↪ float_of_string i)
157             | SCharLit l -> L.const_int i8_t (int_of_char l)
158             | SStringLit l -> L.build_global_stringptr l "tmp"
159                 ↪ builder
160             | SNoexpr      -> L.const_int i32_t 0
161             | SID s        -> L.build_load (lookup s) s builder
162             | SMatrixLit (contents, rows, cols) ->
163                 let rec expr_list = function
164                     [] -> []
165                     | hd::tl -> expr_builder hd::expr_list tl
166                 in
167                 let contents' = expr_list contents
168                 in
169                 let m = L.build_call matrix_init_f [| L.
170                     ↪ const_int i32_t cols; L.const_int i32_t
171                     ↪ rows |] "matrix_init" builder
172                 in
173                 ignore(List.map (fun v -> L.build_call
174                     ↪ store_matrix_f [| m ; v |] "store_val"
175                     ↪ builder) contents'); m
176             | SAsn (s, e) -> let e' = expr_builder e in
177                 ignore(L.build_store e' (lookup s)
178                     ↪ builder); e'
179             | SBinop ((A.Float,_) as e1, op, e2) ->
180                 let e1' = expr_builder e1
181                 and e2' = expr_builder e2 in
182                 (match op with
183                  A.Add      -> L.build_fadd
184                  | A.Sub      -> L.build_fsub
185                  | A.Mult     -> L.build_fmul
186                  | A.Div      -> L.build_fdiv
187                  | A.Equal    -> L.build_fcmp L.Fcmp.Oeq

```

```

177 | A.Neq      -> L.build_fcmp L.Fcmp.One
178 | A.Less     -> L.build_fcmp L.Fcmp.Olt
179 | A.Leq      -> L.build_fcmp L.Fcmp.Ole
180 | A.Greater  -> L.build_fcmp L.Fcmp.Ogt
181 | A.Geq      -> L.build_fcmp L.Fcmp.Oge
182 | A.And | A.Or ->
183     raise (Failure "internal error: semant should have
184     ↪ rejected and/or on float")
184 ) e1' e2' "tmp" builder
185     | SBinop (e1, op, e2) ->
186 let e1' = expr builder e1
187 and e2' = expr builder e2 in
188 (match op with
189   A.Add      -> L.build_add
190 | A.Sub      -> L.build_sub
191 | A.Mult     -> L.build_mul
192   | A.Div      -> L.build_sdiv
193 | A.And      -> L.build_and
194 | A.Or       -> L.build_or
195 | A.Equal    -> L.build_icmp L.Icmp.Eq
196 | A.Neq      -> L.build_icmp L.Icmp.Ne
197 | A.Less     -> L.build_icmp L.Icmp.Slt
198 | A.Leq      -> L.build_icmp L.Icmp.Sle
199 | A.Greater  -> L.build_icmp L.Icmp.Sgt
200 | A.Geq      -> L.build_icmp L.Icmp.Sge
201 ) e1' e2' "tmp" builder
202     | SUUnop(op, ((t, _) as e)) ->
203         let e' = expr builder e in
204 (match op with
205   A.Neg when t = A.Float -> L.build_fneg
206 | A.Neg           -> L.build_neg
207   | A.Not          -> L.build_not) e' "tmp"
208     ↪ builder
209     | SCall ("print", [e]) | SCall ("printb", [e]) ->
210         L.build_call printf_func [| int_format_str ; (expr
211             ↪ builder e) |] "printf" builder
212     | SCall ("printstr", [e]) ->
213         L.build_call printf_func [| string_format_str ; (
214             ↪ expr builder e) |] "printf" builder
215     | SCall ("printf", [e]) ->
216         L.build_call printf_func [| float_format_str ; (expr
217             ↪ builder e) |] "printf" builder
218     | SCall ("printm", [e]) ->
219         L.build_call printMatrix_f [| (expr builder e) |] "
220             ↪ printm" builder
221     | SCall ("transpose", [e]) ->
222         L.build_call transpose_matrix_f [| (expr builder e)
223             ↪ |] "transpose" builder
224     | SCall ("matmult", [e1; e2]) ->
225         L.build_call mult_matrix_f [| (expr builder e1); (
226             ↪ expr builder e2) |] "matmult" builder
227     | SCall ("matadd", [e1; e2]) ->
228         L.build_call add_matrix_f [| (expr builder e1); (
229             ↪ expr builder e2) |] "matadd" builder

```

```

222 | SCall ("dot", [e1; e2]) ->
223   L.build_call dot_matrix_f [| (expr builder e1); (
224     ↪ expr builder e2) |] "dot" builder
225 | SCall ("matscale", [e1; e2]) ->
226   L.build_call scale_matrix_f [| (expr builder e1); (
227     ↪ expr builder e2) |] "matscale" builder
228 | SCall ("det", [e1; e2]) ->
229   L.build_call det_matrix_f [| (expr builder e1); (
230     ↪ expr builder e2) |] "det" builder
231 | SCall ("length", [e]) ->
232   L.build_call string_length_f [| (expr builder e) |]
233   ↪ "string_length" builder
234 | SCall ("get_char", [e; index]) ->
235   let index = expr builder index in
236   let e = expr builder e in
237     L.build_call string_get_f [| e; index |] "
238       ↪ string_get" builder;
239 | SCall("sconcat", [e1; e2]) ->
240   L.build_call string_concat_f [| (expr builder e1);
241     ↪ (expr builder e2) |] "string_concat" builder
242 | SCall("sequals", [e1; e2]) ->
243   L.build_call string_equals_f [| (expr builder e1);
244     ↪ (expr builder e2) |] "string_equals" builder
245 | SCall("substring", [s; e1; e2]) ->
246   L.build_call string_substr_f [| (expr builder s); (
247     ↪ expr builder e1); (expr builder e2) |]
248   "string_substr" builder
249 | SCall (f, args) ->
250   let (fdef, fdecl) = StringMap.find f function_decls
251   ↪ in
252 let llargs = (List.rev (List.map (expr builder) (List.rev
253   ↪ args))) in
254 let result = (match fdecl.styp with
255   A.Void -> ""
256   | _ -> f ^ "_result") in
257   L.build_call fdef (Array.of_list llargs) result
258   ↪ builder
259   in
260   ignore(List.map (fun (_, _, v) -> expr builder v) fdecl.
261     ↪ sformals);
262   ignore(List.map (fun (_, _, v) -> expr builder v) fdecl.
263     ↪ slocals);
264
265 (* LLVM insists each basic block end with exactly one "
266   ↪ terminator"
267   instruction that transfers control. This function
268   ↪ runs "instr builder"
269   if the current block does not already have a
270   ↪ terminator. Used,
271   e.g., to handle the "fall off the end of the function
272   ↪ " case. *)
273 let add_terminal builder instr =
274   match L.block_terminator (L.insertion_block builder)
275   ↪ with

```

```

258 Some _ -> ()
259 | None -> ignore (instr builder) in
260
261 (* Build the code for the given statement; return the
262   ↪ builder for
263   the statement's successor (i.e., the next instruction
264   ↪ will be built
265   after the one generated by this call) *)
266
267 let rec stmt builder m = function
268 SBlock sl ->
269     let helper (bldr, map) = stmt bldr map in
270     let (b, _) = List.fold_left helper (builder, m
271       ↪ ) sl in
272     (b, m)
273   | SExpr e -> ignore(expr builder e); (builder, m)
274   | SReturn e -> ignore(match fdecl.styp with
275     (* Special "return nothing"
276       ↪ instr *)
277     A.Void -> L.build_ret_void
278       ↪ builder
279     (* Build return statement *)
280     | _ -> L.build_ret (expr builder
281       ↪ e) builder );
282     (builder, m)
283   | SIf (predicate, then_stmt, else_stmt) ->
284     let bool_val = expr builder predicate in
285     let merge_bb = L.append_block context "merge" the_function
286       ↪ in
287     let build_br_merge = L.build_br merge_bb in (*
288       ↪ partial function *)
289
290     let then_bb = L.append_block context "then" the_function
291       ↪ in
292     let (b', _) = stmt (L.builder_at_end context then_bb) m
293       ↪ then_stmt in
294     add_terminal b'
295     build_br_merge;
296
297     let else_bb = L.append_block context "else" the_function
298       ↪ in
299     let (b', _) = stmt (L.builder_at_end context else_bb) m
300       ↪ else_stmt in
301     add_terminal b'
302     build_br_merge;
303
304     ignore(L.build_cond_br bool_val then_bb else_bb builder);
305     (L.builder_at_end context merge_bb, m)
306
307   | SWhile (predicate, body) ->
308     let pred_bb = L.append_block context "while" the_function
309       ↪ in
310     ignore(L.build_br pred_bb builder);

```

```

299 | let body_bb = L.append_block context "while_body"
300 |   ↪ the_function in
301 | let (b', _) = stmt (L.builder_at_end context body_bb) m
302 |   ↪ body in
303 | add_terminal b'
304 |   (L.build_br pred_bb);
305 |
306 |
307 | let merge_bb = L.append_block context "merge"
308 |   ↪ the_function in
309 | ignore(L.build_cond_br bool_val body_bb merge_bb
310 |   ↪ pred_builder);
311 | (L.builder_at_end context merge_bb, m)
312 |
313 | (* Implement for loops as while loops *)
314 |   | SFor (e1, e2, e3, body) -> stmt builder m
315 |   ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr
316 |     ↪ e3]) ] )
317 |   in
318 |
319 | (* Build the code for each statement in the function *)
320 | let (builder, _) = stmt builder local_vars (SBlock fdecl
321 |   ↪ .sbody) in
322 |
323 | (* Add a return if the last block falls off the end *)
324 | add_terminal builder (match fdecl.styp with
325 |   A.Void -> L.build_ret_void
326 |   | A.Float -> L.build_ret (L.const_float float_t 0.0)
327 |   | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
328 | in
329 |
330 | List.iter build_function_body functions;
331 | the_module

```

8.1.8 matrx.ml

```

1 (* Top-level of the Matrx compiler: scan & parse the input,
2  check the resulting AST and generate an SAST from it,
3  ↪ generate LLVM IR,
4  and dump the module *)
5
6 type action = Ast | Sast | LLVM_IR | Compile
7
8 let () =
9   let action = ref Compile in
10  let set_action a () = action := a in
11  let speclist = [
12    ("-a", Arg.Unit (set_action Ast), "Print the AST");
13    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
14    ("-l", Arg.Unit (set_action LLVM_IR), "Print the
15      ↪ generated LLVM IR");

```

```

14     ("-c", Arg.Unit (set_action Compile),
15      "Check and print the generated LLVM IR (default)");
16  ] in
17  let usage_msg = "usage: ./matrix.native [-a|-s|-l|-c] [file
18    ↪ .mc]" in (* changed name to matrix *)
19  let channel = ref stdin in
20  Arg.parse speclist (fun filename -> channel := open_in
21    ↪ filename) usage_msg;
22
23  let lexbuf = Lexing.from_channel !channel in
24  let ast = Parser.program Scanner.token lexbuf in (*
25    ↪ changed name to Parser *)
26  match !action with
27    Ast -> print_string (Ast.string_of_program ast)
28  | _ -> let sast = Semant.check ast in
29    match !action with
30      Ast -> ()
31    | Sast -> print_string (Sast.string_of_sprogram sast)
32    | LLVM_IR -> print_string (Llvm.string_of_llmodule (
33      ↪ Codegen.translate sast))
34    | Compile -> let m = Codegen.translate sast in
35    Llvm_analysis.assert_valid_module m;
36  print_string (Llvm.string_of_llmodule m)

```

8.1.9 matrix.c

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4
5 static void die(const char *message)
6 {
7     perror(message);
8     exit(1);
9 }
10
11 struct matrix {
12     int num_rows;
13     int num_cols;
14     int** matrixAddr; // accessed [row][col]
15     int buildPosition;
16 };
17 typedef struct matrix matrix;
18
19 int debug = 0;
20
21
22 matrix* storeVal(matrix* target, int value) {
23     int position = target->buildPosition;
24     int curr_row = position / target->num_cols;
25     int curr_col = position % target->num_cols;
26
27     if(debug == 1) {
28         printf("Storing: %d\n", value);
29         printf("in row: %d\n", curr_row);
30         printf("in col: %d\n", curr_col);
31     }

```

```

32     }
33
34     target->matrixAddr [ curr_row ][ curr_col ] = value;
35     target->buildPosition = target->buildPosition + 1;
36     return target;
37 }
38
39 matrix* initMatrix( int* listOfValues , int num_cols , int num_rows ) {
40     int** matrixValues = malloc( num_rows * sizeof( int* ) );
41
42     if( debug == 1 ) {
43         printf("Building matrix:\n");
44         printf("num_rows: %d\n" , num_rows );
45         printf("num_cols: %d\n" , num_cols );
46     }
47
48     //set all values in matrix to NULL if list of values is NULL
49     if (listOfValues == NULL) {
50         for( int i = 0; i < num_rows; i++ ) {
51             int* matrix_row = malloc( num_cols * sizeof( int ) );
52             *(matrixValues + i) = matrix_row;
53             for( int j = 0; j < num_cols; j++ ) {
54                 matrix_row[ j ] = 0;
55             }
56         }
57     }
58
59     //load values from a list of values
60     else {
61         for( int i = 0; i < num_cols; i++ ) {
62             int* matrix_col = malloc( num_rows * sizeof( int ) );
63             *(matrixValues + i) = matrix_col;
64             for( int j = 0; j < num_rows; j++ ) {
65                 matrix_col[ j ] = listOfValues[ i*num_rows + j ];
66             }
67         }
68     }
69
70     //return a pointer to matrix struct
71     matrix* result = malloc( sizeof( struct matrix ) );
72     result->num_cols = num_cols;
73     result->num_rows = num_rows;
74     result->matrixAddr = matrixValues;
75     result->buildPosition = 0;
76     return result;
77 }
78
79 matrix* initMatrix_CG( int num_cols , int num_rows ) {
80     return initMatrix( NULL , num_cols , num_rows );
81 }
82
83 matrix* mAdd( matrix* lhs , matrix* rhs ) {
84     //check dimensions
85     if (lhs->num_rows != rhs->num_rows || lhs->num_cols != rhs->
86         ↪ num_rows) {
87         die("matrix add size mismatch");
88     }
89     int rows = lhs->num_rows;
90     int cols= lhs->num_cols;
91     matrix *result = initMatrix( NULL , rows , cols );
92     for( int i=0; i<rows; i++ ) {
93         for( int j=0; j<cols; j++ ) {

```

```

93     int sum = lhs->matrixAddr[i][j] + rhs->matrixAddr[i][j];
94     result->matrixAddr[i][j] = sum;
95   }
96 }
97
98 return result;
99 }
100
101
102 void getMinor(matrix* m, matrix* temp, int p, int q, int n) {
103   int i=0, j=0;
104   for(int row=0; row<n; row++) {
105     for(int col=0; col<n; col++) {
106       if(row != p && col != q) {
107         temp->matrixAddr[i][j++] = m->matrixAddr[row][col];
108         if(j == n-1) {
109           j=0;
110           i++;
111         }
112       }
113     }
114   }
115 }
116
117
118 int determinant(matrix* input, int n) {
119   int row = input->num_rows;
120   int col = input->num_cols;
121   int d = 0;
122
123   if(row==col) {
124     //base case: matrix contains single element
125     if(n == 1)
126       return input->matrixAddr[0][0];
127
128     matrix* temp = initMatrix(NULL, row, col);
129
130     int sign=1;
131     for(int f=0; f<n; f++) {
132       getMinor(input, temp, 0, f, n);
133       d += sign * input->matrixAddr[0][f] * determinant(temp,
134         ↪ n-1);
135       sign = -sign;
136     }
137
138     return d;
139   }
140   else {
141     printf("Your matrix must be square to compute the
142           ↪ determinant.\n");
143     return 0;
144   }
145 }
146
147
148 matrix* dotProduct(matrix* lhs, matrix* rhs) {
149   //check to make sure matrices are the same size
150   if (lhs->num_cols != rhs->num_rows) {
151     die("Matrices are not the same size!");
152   }

```

```

153 //once we know that matrices are same size , we can compute
154 // result
155 matrix *result = initMatrix(NULL, rhs->num_cols, lhs->num_rows)
156 // ;
157 for (int i=0; i < lhs->num_rows; i++)
158 {
159     for (int j=0; j < rhs->num_cols; j++)
160     {
161         for (int k=0; k < rhs->num_rows; k++)
162         {
163             result->matrixAddr[i][j] += lhs->matrixAddr[i][k] *
164             // rhs->matrixAddr[k][j];
165         }
166     }
167     return result;
168 }
169
170 matrix* transpose(matrix* input) {
171     int rows = input->num_cols;
172     int cols = input->num_rows;
173
174     int** matrixValues = malloc(cols * sizeof(int*));
175
176     for (int i = 0; i < rows; i++) {
177         int* matrix_col = malloc(rows * sizeof(int));
178         *(matrixValues + i) = matrix_col;
179         for (int j = 0; j < cols; j++) {
180             matrix_col[j] = *((*(input->matrixAddr) + j)+i);
181         }
182     }
183
184     input->num_rows = rows;
185     input->num_cols = cols;
186     input->matrixAddr = matrixValues;
187
188     return input;
189 }
190
191 matrix* matrixMult(matrix* lhs , matrix* rhs) {
192     //check dimensions
193     if (lhs->num_rows != rhs->num_rows || lhs->num_cols != rhs->
194     // num_rows) {
195         die("matrix add size mismatch");
196     }
197     int rows = lhs->num_rows;
198     int cols= lhs->num_cols;
199     matrix *result = initMatrix(NULL, rows, cols);
200     for(int i=0; i<rows; i++) {
201         for(int j=0; j<cols; j++) {
202             int product = lhs->matrixAddr[i][j] * rhs->matrixAddr[i][j
203             // ];
204             result->matrixAddr[i][j] = product;
205         }
206     }
207
208     return result;
209 }
210
211 matrix* timesScalar(matrix* input, int scalar) {

```

```

210 |     int rows = input->num_rows;
211 |     int cols= input->num_cols;
212 |     matrix *result = initMatrix(NULL, rows, cols);
213 |     for(int i=0; i<rows; i++) {
214 |         for(int j=0; j<cols; j++) {
215 |             int product = input->matrixAddr[i][j] * scalar;
216 |             result->matrixAddr[i][j] = product;
217 |         }
218 |     }
219 |
220 |     return result;
221 | }
222 |
223 |
224 | void display(matrix* input) {
225 |     int row = input->num_rows;
226 |     int col = input->num_cols;
227 |     for(int i = 0; i<row; i++) {
228 |         for(int j=0; j<col; j++) {
229 |             printf("%d", input->matrixAddr[i][j]);
230 |         }
231 |         printf("\n");
232 |     }
233 | }
234 |
235 | int string_length(char *s) {
236 |     return strlen(s);
237 | }
238 |
239 | char *string_get(char *s, int i) {
240 |     char *c = malloc(2);
241 |     c[0] = s[i];
242 |     c[1] = '\0';
243 |     return c;
244 | }
245 |
246 | char *string_concat(char *s1, char *s2) {
247 |     char *new = (char *) malloc(strlen(s1) + strlen(s2) + 1);
248 |     strcpy(new, s1);
249 |     strcat(new, s2);
250 |     return new;
251 | }
252 |
253 | int string_equals(char *s1, char *s2) {
254 |     return (strcmp(s1, s2) == 0);
255 | }
256 |
257 | char *string_substr(char *s, int start, int end) {
258 |     char *substr = malloc(end - start+1);
259 |     int i;
260 |     for(i = 0; i < (end - start); i++) {
261 |         substr[i] = s[start + i];
262 |     }
263 |     substr[end-start]=0;
264 |     return substr;
265 | }
266 |
267 | #ifdef BUILD_TEST
268 | int main(int argc,char** argv) {
269 |     //run tests of each function
270 |     //initMatrix and display of empty matrix
271 |     matrix *null_matrix=initMatrix(NULL, 2, 2);

```

```

272     printf("NULL MATRIX: \n");
273     display(null_matrix);
274
275
276
277
278 //initMatrix and display of 2x2 matrix
279 int vals1 [] = {3, 8, 4, 6};
280 int *list1 = vals1;
281 matrix *m = initMatrix(list1, 2, 2);
282 printf("2x2 MATRIX: \n");
283 display(m);
284
285 //TODO test codegen builder
286 for( int i = 0; i < 4; i++) {
287     m = storeVal(m, 5);
288     printf("Stroing 5: \n");
289     display(m);
290 }
291
292
293 //add 2 of the same matrix
294 matrix *result_sum = mAdd(m, m);
295 printf("ADD TWO OF THE ORIGINAL 2x2 MATRIX: \n");
296 display(result_sum);
297
298 //multiply two matrices
299 matrix *result_product = matrixMult(m, m);
300 printf("MULTIPLY TWO OF THE ORIGINAL 2X2 MATRIX: \n");
301 display(result_product);
302
303 //scalar multiplication
304 matrix *result_scalar = timesScalar(m, 3);
305 printf("SCALAR MULTIPLICATION OF ORIGINAL MATRIX BY 3: \n");
306 display(result_scalar);
307
308 //determinant of 2x2 matrix
309 printf("The determinant is %d\n", determinant(m, 2));
310
311 //determinant of 3x3 matrix
312 int vals2 [] = {6, 1, 1, 4, -2, 5, 2, 8, 7};
313 int *list2 = vals2;
314 matrix *n = initMatrix(list2, 3, 3);
315 printf("3x3 MATRIX: \n");
316 display(n);
317 printf("The determinant is %d\n", determinant(n, 3));
318
319 //dot product tests
320 int values_1[4] = {1,2,3,4};
321 matrix* m1 = initMatrix(&values_1[0], 2, 2);
322 int values_2[4] = {5,6,7,8};
323 matrix* m2 = initMatrix(&values_2[0], 2, 2);
324
325 printf("The dot product of matrices 1 and 2 is %d\n", dotProduct(
    ↪ m1,m2));
326
327 transpose(m1);
328 display(m1);
329 }
#endif

```

8.1.10 Makefile

```

1 # "make test" Compiles everything and runs the regression tests
2
3 .PHONY : test
4 test : all testall.sh
5 ./testall.sh
6
7 # "make all" builds the executable
8 # to test linking external code
9
10 .PHONY : all
11 all : matrix.native matrix.o
12
13 # "make matrix.native" compiles the compiler
14 #
15 # The _tags file controls the operation of ocamlbuild , e.g., by
16 # including
17 # packages , enabling warnings
18 # See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual
19 #           ↪ .adoc
20
21 matrix.native : matrix.bc
22 opam config exec -- \
23   ocamlbuild -use-ocamlfind matrix.native -pkgs llvm,llvm.analysis ,
24   ↪ llvm.bitreader
25
26 # "make clean" removes all generated files
27 .PHONY : clean
28 clean :
29   ocamlbuild -clean
30   rm -rf testall.log ocaml LLVM *.diff *.ll *.o *.bc matrix
31
32 #build the matrix file
33
34 matrix : matrix.c
35   cc -o matrix -DBUILD_TEST matrix.c
36
37 matrix.bc : matrix.c
38   clang -emit-llvm -o matrix.bc -c matrix.c -Wno-varargs
39
40 # Building the tarball
41 TESTS = \
42   add1 arith1 arith2 arith3 fib float1 float2 float3 for1 for2
43   ↪ func1 \
44   func2 func3 func4 func5 func6 func7 func8 func9 gcd2 gcd global1
45   ↪ \
46   global2 global3 hello if1 if2 if3 if4 if5 if6 local1 local2 ops1
47   ↪ \
48   ops2 var1 var2 while1 while2
49
50 FAILS = \
51   assign1 assign2 assign3 dead1 dead2 expr1 expr2 expr3 float1
52   ↪ float2 \
53   for1 for2 for3 for4 for5 func1 func2 func3 func4 func5 func6
54   ↪ func7 \
55   func8 func9 global1 global2 if1 if2 if3 nomain printb print \
56   return1 return2 while1 while2

```

```

53 TESTFILES = $(TESTS:%=test-%.mc) $(TESTS:%=test-%.out) \
54   $(FAILS:%=fail-%.mc) $(FAILS:%=fail-%.err)
55
56 TARFILES = ast.ml sast.ml codegen.ml Makefile _tags matrx.ml parser
57   ↪ .mly \
58 README scanner.mll semant.ml testall.sh \
59 arcade-font.pbm font2c \
60 Dockerfile \
61 $(TESTFILES:%=tests/%)
62
63 matrx.tar.gz : $(TARFILES)
64   cd .. && tar czf matrx/matrx.tar.gz \

```

8.1.11 README

```

1 The Matrix compiler
2
3 Coded in OCaml, this takes a highly stripped-down subset of C (ints
4   ↪ ,
5 bools, and void types, arithmetic, if-else, for, and while
6   ↪ statements,
7 and user-defined functions) and compiles it into LLVM IR.
8
9 It needs the OCaml llvm library, which is most easily installed
10  ↪ through opam.
11
12 The version of the OCaml llvm library must match the version of the
13   ↪ LLVM
14 system installed on your system.
15
16 In addition to print, which calls the C library function printf(),
17 Matrix gratuitiously includes a primitive function "printm," which
18 prints large ASCII-encoded characters.
19
20 The stock C compiler compiles matrix.o. testall.sh runs the Matrix
21 executable on each testcase (.mx file) to produce a .ll file,
22   ↪ invokes
23 "llc" (the LLVM compiler) to produce a .s (assembly) file, then
24 invokes "cc" (the stock C compiler) to assemble the .s file, link
25   ↪ in
26 matrix.o, and generate an executable. See testall.sh for details.
27
28 If you get errors about llvm.analysis not being found, it's
29   ↪ probably
30 because opam environment information is missing. Either run
31
32 eval $(opam config env)
33
34 or run ocamldoc like this:
35
36 opam config exec -- ocamldoc <args>
37
38 Using Docker
39
40 * Install Docker on whatever operating system you're on
41
42 Under Ubuntu,

```

```

39      apt install docker.io
40
41 * Test your installation
42
43   docker run hello-world
44
45 If this fails , you will need to correct your installation .
46
47 Under Ubuntu, add yourself to the "docker" group:
48
49 sudo usermod -aG docker <username>
50
51 * Move to where you unpacked the Matrx source:
52
53   cd Matrx
54
55 * Invoke docker
56
57   docker run --rm -it -v `pwd`:/home/Matrx -w=/home/Matrx
      ↪ columbiasedwards/plt
58
59 * Inside docker , compile Matrx and run the regression tests:
60
61 # make
62 ...
63 test-add1...OK
64 test-arith1...OK
65 test-arith2...OK
66 test-arith3...OK
67 ... etc.
68
69 # make clean
70
71
72 Installation under Ubuntu 16.04
73
74 LLVM 3.8 is the default under 16.04. Install the matching version
      ↪ of
75 the OCaml LLVM bindings:
76
77 sudo apt install ocamlllvm llvml runtime m4 opam
78 opam init
79 opam install llvm.3.8
80 eval `opam config env`
81
82 make
83 ./testall.sh
84
85
86 Installation under Ubuntu 15.10
87
88 LLVM 3.6 is the default under 15.10, so we ask for a matching
      ↪ version of the
89 OCaml library.
90
91 sudo apt-get install -y ocamll m4 llvml opam
92 opam init
93 opam install llvm.3.6 ocamlfind
94 eval `opam config env`
95
96 make
97 ./testall.sh

```

```

98
99
100 Installation under Ubuntu 14.04
101
102 The default LLVM package is 3.4, so we install the matching OCaml
103 library using opam. The default version of opam under 14.04 is too
104 old; we need to use a newer package.
105
106 sudo apt-get install m4 llvm software-properties-common
107
108 sudo add-apt-repository --yes ppa:avsm/ppa
109 sudo apt-get update -qq
110 sudo apt-get install -y opam
111 opam init
112
113 eval `opam config env`
114
115 opam install llvm.3.4 ocamlfind
116
117 Installation under OS X
118
119 1. Install Homebrew:
120
121     ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew
122             ↪ /install/master/install)"
123
124 2. Verify Homebrew is installed correctly:
125
126     brew doctor
127
128 3. Install opam:
129
130     brew install opam
131
132 4. Set up opam:
133
134     opam init
135
136 5. Install llvm:
137
138     brew install llvm
139
140     Take note of where brew places the llvm executables. It will
141             ↪ show
142     you the path to them under the CAVEATS section of the post-
143             ↪ install
144     terminal output. For me, they were in /usr/local/opt/llvm/bin.
145             ↪ Also
146     take note of the llvm version installed. For me, it was 3.6.2.
147
148 6. Have opam set up your enviroment:
149
150     eval `opam config env`
151
152 7. Install the OCaml llvm library:
153
154     opam install llvm.3.6
155
156     Ensure that the version of llvm you install here matches the
157     version you installed via brew. Brew installed llvm version
158             ↪ 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.

```

206 IF YOU GET OPERATION NOT PERMITTED ERROR, then this is probably
207 ↪ a
208 result of OSX's System Integrity Protection.
209 One way to get around this is to reboot your machine into
210 ↪ recovery
211 mode (by holding cmd-r when restarting). Open a terminal from
212 recovery mode by going to Utilities → Terminal, and enter the
213 following commands:
214 csrutil disable
215 reboot
216 After your machine has restarted, try the ‘ln . . .’ command again
217 ↪ ,
218 and it should succeed.
219
220 IMPORTANT: the previous step disables System Integrity Protection
221 ↪ ,
222 which can leave your machine vulnerable. It’s highly advisable
223 ↪ to
224 reenable System Integrity Protection when you are done by
225 rebooting your machine into recovery mode and entering the
226 ↪ following
227 command in the terminal:
228
229 csrutil enable
230 reboot
231
232 9. To run and test, navigate to the Matrix folder. Once there, run
233 make ; ./testall.sh
234
235 Matrix should build without any complaints and all tests should
236 pass.
237
238 IF RUNNING ./testall.sh FAILS ON SOME TESTS, check to make sure
239 ↪ you
240 have symlinked the correct executable from your llvm
241 ↪ installation.
242 For example, if the executable is named lli-[version], then the
243 previous step should have looked something like:
244
245 sudo ln -s /usr/local/opt/llvm/bin/lli-3.7 /usr/bin/lli
246
247 As before, you may also modify the path to lli in testall.sh
248
249 To run and test:
250
251 $ make
252 ocamldoc -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4
253 ↪ Matrix.native
254 Finished, 22 targets (0 cached) in 00:00:01.
255 cc -c -o matrix.o matrix.c
256 $ ./testall.sh
257 test-arith1...OK
258 test-arith2...OK
259 test-arith3...OK
260 test-fib...OK
261 ...
262 fail-while1...OK

```

259 | fail-while2 ...OK

8.1.12 demo.mx

```
1 int main() {
2     matrix a = [[1,2]];
3     matrix b;
4     matrix c = [[1,2],[3,4]];
5     matrix d = [[4,4],[4,4]];
6     matrix e;
7     matrix f = [[1][2][3][4][5][6]];
8
9     printstr("  matrix a = [[1,2]];
10    matrix b;
11    matrix c = [[1,2],[3,4]];
12    matrix d = [[4,4],[4,4]];
13    matrix e;
14    matrix f = [[1][2][3][4][5][6]];
15  ");
16
17    b = [[3,4]];
18    printstr(" Print Matrix A:");
19    printm(a);
20    printstr("");
21    printstr(" Print Matrix B:");
22    printm(b);
23
24    printstr("");
25
26    printstr(" Transpose Matrix B:");
27    printm(transpose(b));
28
29    printstr("");
30
31    printstr("Dot A and B");
32    printm(dot(a,b));
33
34    printstr("");
35    printstr("Dot B and A");
36    printm(dot(b,a));
37
38    printstr("");
39    printstr("[[1,2,3][3,3,3]] DOT [[2][3][1]])");
40    printm(dot([[1,2,3][3,3,3]], [[2][3][1]]));
41
42    printstr("");
43    printstr(" Print Matrix C:");
44    printm(c);
45    printstr("");
46
47    printstr(" Print Matrix D:");
48    printm(d);
49    printstr("");
50
51    printstr("E = C + D:");
52    printm(e = matadd(c, d));
53    printstr("");
54
55    printstr("C * D:");
56    printm(matmult(c, d));
57    printstr("");
```

```

58 printstr("Det of [[1,0][0,1]]:");
59 print(det([[1,0][0,1]],2));
60 printstr("");
61
62 printstr("Det of [[1,2][3,4]]:");
63 print(det([[1,2][3,4]],2));
64 printstr("");
65
66 printstr("Scale [[1,1][2,2]] by 3");
67 printm(matscale([[1,1][2,2]], 3));
68 printstr("");
69
70 return 0;
71 }

```

8.2 Tests

8.2.1 fail-assign1.err

```

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

```

8.2.2 fail-assign1.mx

```

1 int main()
2 {
3     int i;
4     bool b;
5
6     i = 42;
7     i = 10;
8     b = true;
9     b = false;
10    i = false; /* Fail: assigning a bool to an integer */
11 }

```

8.2.3 fail-assign2.err

```

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

```

8.2.4 fail-assign2.mx

```

1 int main()
2 {
3     int i;
4     bool b;
5
6     b = 48; /* Fail: assigning an integer to a bool */
7 }

```

8.2.5 fail-assign3.err

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

8.2.6 fail-assign3.mx

```
1 void myvoid()  
2 {  
3     return;  
4 }  
5  
6 int main()  
7 {  
8     int i;  
9  
10    i = myvoid(); /* Fail: assigning a void to an integer */  
11 }
```

8.2.7 fail-dead1.err

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

8.2.8 fail-dead1.mx

```
1 int main()  
2 {  
3     int i;  
4  
5     i = 15;  
6     return i;  
7     i = 32; /* Error: code after a return */  
8 }
```

8.2.9 fail-dead2.err

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

8.2.10 fail-dead2.mx

```
1 int main()  
2 {  
3     int i;  
4  
5     {  
6         i = 15;  
7         return i;  
8     }  
9     i = 32; /* Error: code after a return */  
10 }
```

8.2.11 fail-expr1.err

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

8.2.12 fail-expr1.mx

```
1 int a;
2 bool b;
3
4 void foo( int c, bool d)
5 {
6     int dd;
7     bool e;
8     a + c;
9     c - a;
10    a * 3;
11    c / 2;
12    d + a; /* Error: bool + int */
13 }
14
15 int main()
16 {
17     return 0;
18 }
```

8.2.13 fail-expr2.err

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

8.2.14 fail-expr2.mx

```
1 int a;
2 bool b;
3
4 void foo( int c, bool d)
5 {
6     int d;
7     bool e;
8     b + a; /* Error: bool + int */
9 }
10
11 int main()
12 {
13     return 0;
14 }
```

8.2.15 fail-expr3.err

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

8.2.16 fail-expr3.mx

```
1 int a;
2 float b;
3
4 void foo(int c, float d)
5 {
6     int d;
7     float e;
8     b + a; /* Error: float + int */
9 }
10
11 int main()
12 {
13     return 0;
14 }
```

8.2.17 fail-float1.err

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

8.2.18 fail-float1.mx

```
1 int main()
2 {
3     -3.5 && 1; /* Float with AND? */
4     return 0;
5 }
```

8.2.19 fail-float2.err

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

8.2.20 fail-float2.mx

```
1 int main()
2 {
3     -3.5 && 2.5; /* Float with AND? */
4     return 0;
5 }
```

8.2.21 fail-for1.err

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

8.2.22 fail-for1.mx

```
1 int main()
2 {
3     int i;
4     for ( ; true ; ) {} /* OK: Forever */
5
6     for (i = 0 ; i < 10 ; i = i + 1) {
7         if (i == 3) return 42;
8     }
9
10    for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */
11
12    return 0;
13 }
```

8.2.23 fail-for2.err

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

8.2.24 fail-for2.mx

```
1 int main()
2 {
3     int i;
4
5     for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */
6
7     return 0;
8 }
```

8.2.25 fail-for3.err

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

8.2.26 fail-for3.mx

```
1 int main()
2 {
3     int i;
4
5     for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */
6
7     return 0;
8 }
```

8.2.27 fail-for4.err

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

8.2.28 fail-for4.mx

```
1 int main()
2 {
3     int i;
4
5     for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */
6
7     return 0;
8 }
```

8.2.29 fail-for5.err

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

8.2.30 fail-for5.mx

```
1 int main()
2 {
3     int i;
4
5     for (i = 0; i < 10 ; i = i + 1) {
6         foo(); /* Error: no function foo */
7     }
8
9     return 0;
10 }
```

8.2.31 fail-func1.err

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

8.2.32 fail-func1.mx

```
1 int foo() {}
2 int bar() {}
3 int baz() {}
4
5 void bar() {} /* Error: duplicate function bar */
6
7 int main()
8 {
9     return 0;
10 }
```

8.2.33 fail-func2.err

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

8.2.34 fail-func2.mx

```
1 int foo(int a, bool b, int c) { }
2
3 void bar(int a, bool b, int a) {} /* Error: duplicate formal a in
   ↪ bar */
4
5 int main()
6 {
7     return 0;
8 }
```

8.2.35 fail-func3.err

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

8.2.36 fail-func3.mx

```
1 int foo(int a, bool b, int c) { }
2
3 void bar(int a, void b, int c) {} /* Error: illegal void formal b
   ↪ */
4
5 int main()
6 {
7     return 0;
8 }
```

8.2.37 fail-func4.err

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

8.2.38 fail-func4.mx

```
1 int foo() {}
2
3 void bar() {}
4
5 int print() {} /* Should not be able to define print */
6
7 void baz() {}
8
9 int main()
10 {
11     return 0;
12 }
```

8.2.39 fail-func5.err

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

8.2.40 fail-func5.mx

```
1 int foo() {}
2
3 int bar() {
4     int a;
5     void b; /* Error: illegal void local b */
6     bool c;
7
8     return 0;
9 }
10
11 int main()
12 {
13     return 0;
14 }
```

8.2.41 fail-func6.err

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

8.2.42 fail-func6.mx

```
1 void foo(int a, bool b)
2 {
3 }
4
5 int main()
6 {
7     foo(42, true);
8     foo(42); /* Wrong number of arguments */
9 }
```

8.2.43 fail-func7.err

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

8.2.44 fail-func7.mx

```
1 void foo(int a, bool b)
2 {
3 }
4
5 int main()
6 {
7     foo(42, true);
8     foo(42, true, false); /* Wrong number of arguments */
9 }
```

8.2.45 fail-func8.err

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

8.2.46 fail-func8.mx

```
1 void foo(int a, bool b)
2 {
3 }
4
5 void bar()
6 {
7 }
8
9 int main()
10 {
11     foo(42, true);
12     foo(42, bar()); /* int and void, not int and bool */
13 }
```

8.2.47 fail-func9.err

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

8.2.48 fail-func9.mx

```
1 void foo(int a, bool b)
2 {
3 }
4
5 int main()
6 {
7     foo(42, true);
8     foo(42, 42); /* Fail: int, not bool */
9 }
```

8.2.49 fail-global1.err

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

8.2.50 fail-global1.mx

```
1 int c;
2 bool b;
3 void a; /* global variables should not be void */
4
5
6 int main()
7 {
8     return 0;
9 }
```

8.2.51 fail-global2.err

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

8.2.52 fail-global2.mx

```
1 int b;
2 bool c;
3 int a;
4 int b; /* Duplicate global variable */
5
6 int main()
7 {
8     return 0;
9 }
```

8.2.53 fail-if1.err

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

8.2.54 fail-if1.mx

```
1 int main()
2 {
3     if (true) {}
4     if (false) {} else {}
5     if (42) {} /* Error: non-bool predicate */
6 }
```

8.2.55 fail-if2.err

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

8.2.56 fail-if2.mx

```
1 int main()
2 {
3     if (true) {
4         foo; /* Error: undeclared variable */
5     }
6 }
```

8.2.57 fail-if3.err

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

8.2.58 fail-if3.mx

```
1 int main()
2 {
3     if (true) {
4         42;
5     } else {
6         bar; /* Error: undeclared variable */
7     }
8 }
```

8.2.59 fail-matrix-dims-3D.err

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

8.2.60 fail-matrix-dims-3D.mx

```
1 int main(){
2     matrix m;
3     m = [
4         [
5             [ [1, 2, 3], [3, 4, 5] ],
6             [ [1, 2, 3], [3, 4, 5] ]
7         ];
}
```

8.2.61 fail-matrix-dims.err

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

8.2.62 fail-matrix-dims.mx

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

8.2.63 fail-nomain.err

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

8.2.64 fail-printb.err

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

8.2.65 fail-printb.mx

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

8.2.66 fail-print.err

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

8.2.67 fail-print.mx

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

8.2.68 fail-return1.err

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

8.2.69 fail-return1.mx

```
1 int main()
2 {
3   return true; /* Should return int */
4 }
```

8.2.70 fail-return2.err

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

8.2.71 fail-return2.mx

```
1 void foo()
2 {
3   if (true) return 42; /* Should return void */
4   else return;
5 }
6
7 int main()
8 {
9   return 42;
10 }
```

8.2.72 fail-while1.err

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

8.2.73 fail-while1.mx

```
1 int main()
2 {
3     int i;
4
5     while (true) {
6         i = i + 1;
7     }
8
9     while (42) { /* Should be boolean */
10        i = i + 1;
11    }
12 }
```

8.2.74 fail-while2.err

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

8.2.75 fail-while2.mx

```
1 int main()
2 {
3     int i;
4
5     while (true) {
6         i = i + 1;
7     }
8
9     while (true) {
10        foo(); /* foo undefined */
11    }
12 }
```

8.2.76 test-add1.mx

```
1 int add(int x, int y)
2 {
3     return x + y;
4 }
5
6 int main()
7 {
8     print( add(17, 25) );
9     return 0;
10 }
```

8.2.77 test-add1.out

```
1 42
```

8.2.78 test-arith1.mx

```
1 int main()
2 {
3     print(39 + 3);
4     return 0;
5 }
```

8.2.79 test-arith1.out

```
1 42
```

8.2.80 test-arith2.mx

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

8.2.81 test-arith2.out

```
1 11
```

8.2.82 test-arith3.mx

```
1 int foo(int a)
2 {
3     return a;
4 }
5
6 int main()
7 {
8     int a;
9     a = 42;
10    a = a + 5;
11    print(a);
12    return 0;
13 }
```

8.2.83 test-arith3.out

```
1 47
```

8.2.84 test-fib.mx

```
1 int fib (int x)
2 {
3     if (x < 2) return 1;
4     return fib(x-1) + fib(x-2);
5 }
6
7 int main()
8 {
9     print(fib(0));
10    print(fib(1));
11    print(fib(2));
12    print(fib(3));
13    print(fib(4));
14    print(fib(5));
15    return 0;
16 }
```

8.2.85 test-fib.out

```
1 1
2 1
3 2
4 3
5 5
6 8
```

8.2.86 test-float1.mx

```
1 int main()
2 {
3     float a;
4     a = 3.14159267;
5     printf(a);
6     return 0;
7 }
```

8.2.87 test-float1.out

```
1 3.14159
```

8.2.88 test-float2.mx

```
1 int main()
2 {
3     float a;
4     float b;
5     float c;
6     a = 3.14159267;
7     b = -2.71828;
8     c = a + b;
9     printf(c);
10    return 0;
11 }
```

8.2.89 test-float2.out

```
1 0.423313
```

8.2.90 test-float3.mx

```
1 void testfloat( float a, float b)
2 {
3     printf(a + b);
4     printf(a - b);
5     printf(a * b);
6     printf(a / b);
7     printb(a == b);
8     printb(a == a);
9     printb(a != b);
10    printb(a != a);
11    printb(a > b);
12    printb(a >= b);
13    printb(a < b);
14    printb(a <= b);
15 }
16
17 int main()
18 {
19     float c;
20     float d;
21
22     c = 42.0;
23     d = 3.14159;
24
25     testfloat(c, d);
26
27     testfloat(d, d);
28
29     return 0;
30 }
```

8.2.91 test-float3.out

```
1 45.1416
2 38.8584
3 131.947
4 13.369
5 0
6 1
7 1
8 0
9 1
10 1
11 0
12 0
13 6.28318
14 0
15 9.86959
16 1
17 1
18 1
19 0
```

```
20 0
21 0
22 1
23 0
24 1
```

8.2.92 test-for1.mx

```
1 int main()
2 {
3     int i;
4     for (i = 0 ; i < 5 ; i = i + 1) {
5         print(i);
6     }
7     print(42);
8     return 0;
9 }
```

8.2.93 test-for1.out

```
1 0
2 1
3 2
4 3
5 4
6 42
```

8.2.94 test-for2.mx

```
1 int main()
2 {
3     int i;
4     i = 0;
5     for ( ; i < 5; ) {
6         print(i);
7         i = i + 1;
8     }
9     print(42);
10    return 0;
11 }
```

8.2.95 test-for2.out

```
1 0
2 1
3 2
4 3
5 4
6 42
```

8.2.96 test-func1.mx

```
1 int add(int a, int b)
2 {
3     return a + b;
4 }
5
6 int main()
7 {
8     int a;
9     a = add(39, 3);
10    print(a);
11    return 0;
12 }
```

8.2.97 test-func1.out

```
1 42
```

8.2.98 test-func2.mx

```
1 /* Bug noticed by Pin-Chin Huang */
2
3 int fun(int x, int y)
4 {
5     return 0;
6 }
7
8 int main()
9 {
10    int i;
11    i = 1;
12
13    fun(i = 2, i = i+1);
14
15    print(i);
16    return 0;
17 }
```

8.2.99 test-func2.out

```
1 2
```

8.2.100 test-func3.mx

```
1 void printem(int a, int b, int c, int d)
2 {
3     print(a);
4     print(b);
5     print(c);
6     print(d);
7 }
8
9 int main()
```

```
10 {
11     printem(42,17,192,8);
12     return 0;
13 }
```

8.2.101 test-func3.out

```
1 42
2 17
3 192
4 8
```

8.2.102 test-func4.mx

```
1 int add(int a, int b)
2 {
3     int c;
4     c = a + b;
5     return c;
6 }
7
8 int main()
9 {
10    int d;
11    d = add(52, 10);
12    print(d);
13    return 0;
14 }
```

8.2.103 test-func4.out

```
1 62
```

8.2.104 test-func5.mx

```
1 int foo(int a)
2 {
3     return a;
4 }
5
6 int main()
7 {
8     return 0;
9 }
```

8.2.105 test-func6.mx

```
1 void foo() {}
2
3 int bar(int a, bool b, int c) { return a + c; }
4
5 int main()
```

```
6 {  
7     print(bar(17,  false , 25));  
8     return 0;  
9 }
```

8.2.106 test-func6.out

```
1 42
```

8.2.107 test-func7.mx

```
1 int a;  
2  
3 void foo(int c)  
4 {  
5     a = c + 42;  
6 }  
7  
8 int main()  
9 {  
10    foo(73);  
11    print(a);  
12    return 0;  
13 }
```

8.2.108 test-func7.out

```
1 115
```

8.2.109 test-func8.mx

```
1 void foo(int a)  
2 {  
3     print(a + 3);  
4 }  
5  
6 int main()  
7 {  
8     foo(40);  
9     return 0;  
10 }
```

8.2.110 test-func8.out

```
1 43
```

8.2.111 test-func9.mx

```
1 void foo(int a)
2 {
3     print(a + 3);
4     return;
5 }
6
7 int main()
8 {
9     foo(40);
10    return 0;
11 }
```

8.2.112 test-func9.out

```
1 43
```

8.2.113 test-gcd2.mx

```
1 int gcd(int a, int b) {
2     while (a != b)
3         if (a > b) a = a - b;
4         else b = b - a;
5     return a;
6 }
7
8 int main()
9 {
10     print(gcd(14,21));
11     print(gcd(8,36));
12     print(gcd(99,121));
13     return 0;
14 }
```

8.2.114 test-gcd2.out

```
1 7
2 4
3 11
```

8.2.115 test-gcd.mx

```
1 int gcd(int a, int b) {
2     while (a != b) {
3         if (a > b) a = a - b;
4         else b = b - a;
5     }
6     return a;
7 }
8
9 int main()
10 {
11     print(gcd(2,14));
```

```
12 |     print(gcd(3,15));
13 |     print(gcd(99,121));
14 |     return 0;
15 | }
```

8.2.116 test-gcd.out

```
1 2
2 3
3 11
```

8.2.117 test-global1.mx

```
1 int a;
2 int b;
3
4 void printa()
5 {
6     print(a);
7 }
8
9 void printbb()
10 {
11     print(b);
12 }
13
14 void incab()
15 {
16     a = a + 1;
17     b = b + 1;
18 }
19
20 int main()
21 {
22     a = 42;
23     b = 21;
24     printa();
25     printbb();
26     incab();
27     printa();
28     printbb();
29     return 0;
30 }
```

8.2.118 test-global1.out

```
1 42
2 21
3 43
4 22
```

8.2.119 test-global2.mx

```
1 bool i;
2
3 int main()
4 {
5     int i; /* Should hide the global i */
6
7     i = 42;
8     print(i + i);
9
10 }
```

8.2.120 test-global2.out

```
1 84
```

8.2.121 test-global3.mx

```
1 int i;
2 bool b;
3 int j;
4
5 int main()
6 {
7     i = 42;
8     j = 10;
9     print(i + j);
10 }
11 }
```

8.2.122 test-global3.out

```
1 52
```

8.2.123 test-hello.mx

```
1 int main()
2 {
3     print(42);
4     print(71);
5     print(1);
6     return 0;
7 }
```

8.2.124 test-hello.out

```
1 42
2 71
3 1
```

8.2.125 test-if1.mx

```
1 int main()
2 {
3     if (true) print(42);
4     print(17);
5     return 0;
6 }
```

8.2.126 test-if1.out

```
1 42
2 17
```

8.2.127 test-if2.mx

```
1 int main()
2 {
3     if (true) print(42); else print(8);
4     print(17);
5     return 0;
6 }
```

8.2.128 test-if2.out

```
1 42
2 17
```

8.2.129 test-if3.mx

```
1 int main()
2 {
3     if (false) print(42);
4     print(17);
5     return 0;
6 }
```

8.2.130 test-if3.out

```
1 17
```

8.2.131 test-if4.mx

```
1 int main()
2 {
3     if (false) print(42); else print(8);
4     print(17);
5     return 0;
6 }
```

8.2.132 test-if4.out

```
1 8  
2 17
```

8.2.133 test-if5.mx

```
1 int cond(bool b)  
2 {  
3     int x;  
4     if (b)  
5         x = 42;  
6     else  
7         x = 17;  
8     return x;  
9 }  
10  
11 int main()  
12 {  
13     print(cond(true));  
14     print(cond(false));  
15     return 0;  
16 }
```

8.2.134 test-if5.out

```
1 42  
2 17
```

8.2.135 test-if6.mx

```
1 int cond(bool b)  
2 {  
3     int x;  
4     x = 10;  
5     if (b)  
6         if (x == 10)  
7             x = 42;  
8     else  
9         x = 17;  
10    return x;  
11 }  
12  
13 int main()  
14 {  
15     print(cond(true));  
16     print(cond(false));  
17     return 0;  
18 }
```

8.2.136 test-if6.out

```
1 42  
2 10
```

8.2.137 test-local1.mx

```
1 void foo(bool i)
2 {
3     int i; /* Should hide the formal i */
4
5     i = 42;
6     print(i + i);
7 }
8
9 int main()
10 {
11     foo(true);
12     return 0;
13 }
```

8.2.138 test-local1.out

```
1 84
```

8.2.139 test-local2.mx

```
1 int foo(int a, bool b)
2 {
3     int c;
4     bool d;
5
6     c = a;
7
8     return c + 10;
9 }
10
11 int main() {
12     print(foo(37, false));
13     return 0;
14 }
```

8.2.140 test-local2.out

```
1 47
```

8.2.141 test-matadd.mx

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

8.2.142 test-matadd.out

```
1 2 4
2 6 8
```

8.2.143 test-matdet.mx

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

8.2.144 test-matdet.out

```
1 [-2
```

8.2.145 test-mat-dot2.mx

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

8.2.146 test-mat-dot2.out

```
1 1 2
2 3 6
```

8.2.147 test-mat-dot.mx

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

8.2.148 test-mat-dot.out

```
1 7
```

8.2.149 test-matmult.mx

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

8.2.150 test-matmult.out

```
1  1 4  
2  9 16
```

8.2.151 test-mat-print2.mx

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

8.2.152 test-mat-print2.out

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

8.2.153 test-mat-print.mx

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

8.2.154 test-mat-print.out

```
1  1 2  
2  3 4
```

8.2.155 test-matrix1.mx

```
1 int main(){  
2     matrix a = [ [1 , 2] [3 , 4] ];  
3     printm(a);  
4     return 0;  
5 }
```

8.2.156 test-matrix1.out

```
1  1 2  
2  3 4
```

8.2.157 test-matscale.mx

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

8.2.158 test-matscale.out

```
1 4 8
2 12 16
```

8.2.159 test-mat-trans.mx

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

8.2.160 test-mat-trans.out

```
1 1 3
2 2 4
```

8.2.161 test-ops1.mx

```
1 int main()
2 {
3     print(1 + 2);
4     print(1 - 2);
5     print(1 * 2);
6     print(100 / 2);
7     print(99);
8     printb(1 == 2);
9     printb(1 == 1);
10    print(99);
11    printb(1 != 2);
12    printb(1 != 1);
13    print(99);
14    printb(1 < 2);
15    printb(2 < 1);
16    print(99);
17    printb(1 <= 2);
18    printb(1 <= 1);
19    printb(2 <= 1);
20    print(99);
21    printb(1 > 2);
22    printb(2 > 1);
23    print(99);
24    printb(1 >= 2);
25    printb(1 >= 1);
26    printb(2 >= 1);
```

```
27 }     return 0;  
28 }
```

8.2.162 test-ops1.out

```
1 3  
2 -1  
3 2  
4 50  
5 99  
6 0  
7 1  
8 99  
9 1  
10 0  
11 99  
12 1  
13 0  
14 99  
15 1  
16 1  
17 0  
18 99  
19 0  
20 1  
21 99  
22 0  
23 1  
24 1
```

8.2.163 test-ops2.mx

```
1 int main()  
2 {  
3     printb(true);  
4     printb(false);  
5     printb(true && true);  
6     printb(true && false);  
7     printb(false && true);  
8     printb(false && false);  
9     printb(true || true);  
10    printb(true || false);  
11    printb(false || true);  
12    printb(false || false);  
13    printb(!false);  
14    printb(!true);  
15    print(-10);  
16    print(--42);  
17 }
```

8.2.164 test-ops2.out

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

```
5 0
6 0
7 1
8 1
9 1
10 0
11 1
12 0
13 -10
14 42
```

8.2.165 test-string1.mx

```
1 int main()
2 {
3     string s1 = "This is an example";
4     printstr(get_char(s1, 0));
5     return 0;
6 }
```

8.2.166 test-string1.out

```
1 T
```

8.2.167 test-string2.mx

```
1 int main()
2 {
3     printb(sequals("MATRX", "MATRX"));
4     printstr(sconcat("This is an example ", "of a string in MATRIX!")
5                 ↪ );
6     printstr(substring("This is an example of a string in MATRIX!", 3,
7                         ↪ 7));
6     return 0;
7 }
```

8.2.168 test-string2.out

```
1
2 This is an example of a string in MATRIX!
3 s is
```

8.2.169 test-string-decl1.mx

```
1 int main()
2 {
3     string s1 = "This is an example";
4     printstr(s1);
5
6     return 0;
7 }
```

8.2.170 test-string-decl1.out

```
1 This is an example
```

8.2.171 test-string-decl2.mx

```
1 int main()
2 {
3     string s2;
4     s2 = "This is another example";
5     printstr(s2);
6
7     return 0;
8 }
```

8.2.172 test-string-decl2.out

```
1 This is another example
```

8.2.173 test-var1.mx

```
1 int main()
2 {
3     int a;
4     a = 42;
5     print(a);
6     return 0;
7 }
```

8.2.174 test-var1.out

```
1 42
```

8.2.175 test-var2.mx

```
1 int a;
2
3 void foo( int c )
4 {
5     a = c + 42;
6 }
7
8 int main()
9 {
10    foo(73);
11    print(a);
12    return 0;
13 }
```

8.2.176 test-var2.out

```
1 115
```

8.2.177 test-while1.mx

```
1 int main()
2 {
3     int i;
4     i = 5;
5     while (i > 0) {
6         print(i);
7         i = i - 1;
8     }
9     print(42);
10    return 0;
11 }
```

8.2.178 test-while1.out

```
1 5
2 4
3 3
4 2
5 1
6 42
```

8.2.179 test-while2.mx

```
1 int foo(int a)
2 {
3     int j;
4     j = 0;
5     while (a > 0) {
6         j = j + 2;
7         a = a - 1;
8     }
9     return j;
10 }
11
12 int main()
13 {
14     print(foo(7));
15     return 0;
16 }
```

8.2.180 test-while2.out

```
1 14
```