COMS W4115
Programming Languages and Translators

M^2 Final Report

Shelley Zhong - sz2699

Tengyu Zhou - tz2338

Christine Pape - cmp2223

Jeffrey Monahan - jm4155

Montana St. Pierre - mrs2296

December 20, 2017

Contents

1 Introduction

1.1 Motivation

Our language is certainly designed with certain predecessors in mind (C, Java, and even Python were all kept in mind when designing M^2). Truthfully, we wanted to find a balance between something like C, which, for a seasoned programmer, is incredibly fun and empowering (but can be devastating to anyone who doesn't have intimate understanding of scope and memory), and Python, which is, perhaps, too simplified in that it disposes of any and all type system and renders anything beyond a moderately complex block of code nigh impossible to read (we don't like Python).

Matrices are an integral part of many mathematical operations and have uses ranging anywhere from graphics to cryptography. Our goal is to create a language which streamlines working with matrices, as well as combining matrix operations with broader mathematical operations. Our intention was to keep the language as open as possible, in that it would not serve to box the user in by hard coding too many operations in for them, and thus not allowing them to use creative solutions. Thus, our motivation behind the design of this language was to ensure two primary pillars of design: 1. streamlining mathematical operations, by providing intuitive functionality (to the best of our ability) and 2. allow the programmer to approach a problem from various angles. For instance, there is a function provided for doing various matrix operations, but this does not keep the programmer from designing their own functions to utilize instead of ours, or combining our functions to create something new. By creating this type of platform wherein intuitive building blocks are provided for combining matrices and other mathematical components, we hoped to streamline the process of performing matrix operations.

1.2 Language Description

M^2 is a programming language with the primary purpose of simplifying and streamlining matrix operations and the auxiliary goal of doing the same for some broader mathematical operations. To this end, we aim to implement the basic building blocks of all complex matrix functions (e.g. transpose, trace, inverse, etc.), basic vector operations (e.g. dot product) and matrix-vector operations, and basic mathematical functions. As matrices and vectors are, naturally, mathematically interchangeable in many ways, we decided to unify the two into one data structure; therefore, to define a vector, one need only create a 1-dimensional matrix, and all matrix operators, then, apply.

When implementing our matrix data structure, we decided to model it so that the user would define a matrix row by row. The following is the syntax for instantiating a 3×3 matrix variable:

```
matrix int [3][3] A;
A = [[1, 2, 3]; [4, 5, 6]; [7, 8, 9]];
```

Furthermore, we will implement the fundamental operators needed to perform most mathematical operations. We feel that it is important the program has the ability to interact with the user, so we keep the idea of strings. We also keep basic fundamentals, such as conditional statements and loops. Our syntactical style remains similar to the widely used languages. i.e. We will be keeping our language easy enough to learn by someone who already knows modern programming languages while adding enough matrix-specific features so that it is specialized.

2 Language Tutorial

In this section, we present a quick tutorial of our language.

2.1 Basic Syntax

M^2 is very similar to the C programming language in terms of syntax. The entry point to a program is the main function, which is built as follows:

```
int main(){

}
```

Like C, main returns an int, generally 0 for success. This is achieved by the return keyword.

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

Unlike C, however, variables must be declared before they can be initialized.

```
int i;
i = 0;
```

Comments can be single line starting with // or multi-line, enclosed by /* */

Other elements, such as conditional statements, loops, and user-defined functions, also follow the syntax of C. A full specification can be found in section 3.

M^2 also provides a set of built-in functions. These will also be described in detail in section 3, but they include print functions for each data type (e.g. printStr(String s)) and matrix specific functions, like M:rows, which returns the number of rows in matrix M.

2.2 Data Types

M^2 contains six data types:

- void
  - ▷ The none type
  - ▷ Usage: variables cannot be created in the void type. It is used only to signify lack of a value, such as for a return type of a function
- int
  - ▷ An integer value
  - ▷ Usage:

```
int i;
i = 0;
```

- float
  - ▷ A floating point decimal
  - ▷ Usage:

```
float j;
j = 2.5;
```

- boolean
  - ▷ One of two values: true or false
  - ▷ Usage:

```
bool t;
t = true;
```

- string
  - ▷ An array of characters
  - ▷ Usage:

```
String a;
a = "hello";
```

- Matrix
  - ▷ Can be of either int or float type, stores elements similar to a 2D array. Size is determined at declaration.
  - ▷ Usage:

```
matrix int [i][j] m;  //i,j are integers
m[0][j-1] = 3;    //matrices are 0 indexed
```

2.3 Build Matrix

Matrices are the backbone of M^2. For this reason, we take a little longer to show the basics of building and accessing a matrix, as well as using some of the built-in functions.

The following is a sample program for building a matrix:

```
int main(){
        matrix int [3][3] m;  // 1
int count;
count = 0;
int i;
for(i = 0; i<m:rows; ++i){  // 2
        int j;
        for(j = 0; j<m:cols; ++j){  // 3
```

```
                m[i][j] = count; // 4
                ++count;
            }
        }

        return 0;
    }
```

Some interesting points of the above program:

1. Here, a 3×3 matrix named m is declared but not initialized.
2. Here, we use m:rows to find the number of rows in m. This allows us to loop through and access all the rows without going out of bounds
3. For the inner loop, we use another built-in function, m:cols, to find the number of columns in m.
4. Inside the loops, we use the loop counting variables to access each element of the matrix, and fill it with the current value of count. This produces a matrix like:

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

2.4 Sample Programs

This section includes a few simple programs to provide a feel for the language. The first two show how similar our syntax is to C, while the last is more specific to M^2. The specifics of all functionalities presented can be viewed in section 3

2.4.1 Hello World

```
int main() {
String hello;
    hello = "Hello, world!"
    printStr(hello);
        return 0;
}
```

2.4.2 GCD

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

2.4.3 Determinant of a 3×3 matrix

This program is slightly more complicated, using the submatrix function to simplify finding the determinant of a 3×3 matrix.

```
int det(matrix int [2][2] A) {
        int determinant;
        determinant = A[0][0] * A[1][1] - A[1][0] * A[0][1];
    return determinant;
}

int main(){
        matrix int[3][3] M;
    M = [[5,8,9];[3,-7,-8];[-7,-2,1]];

    int term1;
    int term2;
    int term3;
    term1 = M[0][0] * det(M:submatrix(1,2,1,2));
    term2 = M[0][1] * det([[M[1][0],M[1][2]];[[M[2][0],
       M[2][2]]]);
    term3 = M[0][2] * det(M:submatrix(1,2,0,1));

    int determinant;
    determinant = term1 - term2 + term3;

    printInt(determinant);

    return 0;
}
```

3 Language Manual

3.1 General Syntax

Comments

Comments have two different forms: single line comments and block comments.

• A single line comment begins with characters //
• A block comment begins with characters /* and end with characters */

Identifiers Σ: { a-z, A-Z, 0-9, _}

An identifier of a variable is a sequence of characters consists of lowercase letters, uppercase letters, numbers, and underscores. The first character must be a lowercase/uppercase letter.

Reserved words

The following identifiers are reserved for keywords, and should not be declared or used in other circumstances.

- `if`
- `else`
- `return`
- `while`
- `for`
- `void`
- `int`
- `float`
- `main`
- `matrix`
- `string`

Grouping

{ }

Braces are used for grouping a bunch of statements into a single unit. Nested groupings are allowed, but the characters '{' and '}' should always be balanced.

“ ”

Strings will be enclosed by “ ”

Terminator

;

Every statement ends with the character ;. Users should always include ; at the end of statement even if the statement is the last statement in a { } group.

Character Constants

The following sets of characters in single quotes ‘ ’ will be used to define special characters and whitespace as follows:

• \n	newline
• \r	carriage return
• \t	tab

An example of their use is

```
string x = "She said hello.\n";
```

The value of x would then be She said hello. followed by a new line.

Variable Definition

Variables will be defined as **typename varname**

For example, to define an integer called count, one would type:

```
int count;
```

The initialization of variables at declaration is optional. Alphanumeric characters and underscores are valid in variable names.

White Space

Whitespace is ignored unless it is included as a string literal using " "

3.2 Data Types

3.2.1 Primitive Data Types

int

Standard representation of signed or unsigned integers. Example declaration:

```
int x;
x = 1;
```

Literal:

```
['0'-'9']+
```

float

Standard representation of unsigned single-precision floating point numbers. Example declaration:

```
float x;
x = 1.0;
```

Literal:

```
['0'-'9']+ '.' ['0'-'9']+
```

boolean

Standard representation. Holds value 1 for true and 0 for false. Example declaration:

```
bool x;
x = false;
```

Literal:

```
(false|true)
```

void

The none type. Used to signify that a function will not return a value. Variables cannot be declared as void.

matrix

Storing a matrix is not all that different to storing a multidimensional array. However, by implementing it as its own data type, it provides an easy interface for users to work with and manipulate that would be much more difficult if they had to access each element individually. For example, calculating the cross product of two 3×1 matrices involves six separate calculations. That doesn't sound like a lot, but with such a common operation, it would be much easier to simply write $A \times B$ rather than accessing the elements of A and B individually and storing the results in a new matrix. Many operations can be simplified for the user in this way.

The matrix type has a simple definition similar to that of an array in Java. A matrix is stored as an array of elements, where the top left is the 0^{th} element and the bottom right is the last. Example declaration for a 2×3 matrix:

```
matrix int [2][3] M;
```

Matrix literals take the form of

```
[[x_1,..., x_n]; ... [y_1,..., y_n] ]
```

where $x_1, x_2, ..., x_n$ and $y_1, y_2, ... y_n$ all share a common type. For example,

```
matrix type [2][3] M;
M = [[x,y,z];[a,b,c]];
```

The line above will create a 2×3 Matrix of the form

$$\begin{bmatrix} x & y & z \\ a & b & c \end{bmatrix}$$

where x, y, z, a, b, c are instances of <type>. However, ill-formatted definitions such as

```
M = [(a,b,c),(x,z)];
```

would throw an exception because the number of columns is mismatched.

Matrices must also be declared before they can be filled. So the following would also throw an exception:

```
matrix type[2][3] M = [[x,y,z];[a,b,c]];
```

The definition must happen on a separate line.

string

Strings are stored as arrays of characters. Example declaration:

```
string A;
A = "hello world";
```

Literal:

```
'"' (([^ '"'] | "\\\"")* as str) '"'
```

array

Arrays are not an explicit type in M^2. Instead, it is expected that the programmer will make use of a one-dimensional matrix where necessary.

3.3 Declarative Statements

Declaring a variable for primitive types involves a general syntactical structure of:

```
variable_type variable_name;
variable_name = ...;
```

3.4 Operators

3.4.1 Basic Operators

Arithmetic Operators

$$+, -, *, /$$

The + operator can be used on combinations of int/int, int/float, float/float, string/string, and matrix/matrix.

$-$ can be used on int/int, int/float, float/float, and matrix/matrix.

$*$ can be used for int/int, int/float, float/float, matrix/matrix, int/matrix, and float/matrix.

/ can be used on int/int, int/float, and float/float.

For all arithmetic operations of type matrix/matrix, the dimensions of the two operands must be correct. For addition and subtraction, this means both matrices' dimensions must match. For matrix multiplication, this means that the number of columns in the first matrix must match the number of rows in the second matrix.

Relational Operators

$$==, !=, >, <, >=, <=$$

$==$ and $!=$ operators are valid on matching types of int, float, or string.

$>, <, >=, <=$ operators are valid between matching types of int and float.

Matching types mean that both sides of the operator have the same type.The following example would be not be semantically correct.

```
int x = 3;
float y = 4;
if(y >= x){
   ...
}
```

Logical Operators

$$!, \&\&, ||$$

Logical operators are only valid between two boolean values.

Assignment Operator

$$=$$

3.4.2 Matrix-Specific Operators

We use the same arithmetic operators $+, -, *$ for numbers and matrices. Example:

```
A + B, A - B, A * B
```

For addition and subtraction, the matrices must have the same dimention. For multiplication, the number of rows of the first matrix must match the number of columns of the second matrix.

3.5 Conditionals, Loops

3.5.1 Conditionals

If statement

The four forms of the conditional statement are

```
if (expression) {statement1}
```

▷ if expression is non-zero, statement1 will be executed

```
if (expression) {statement1}
else {statement2}
```

▷ if expression is non-zero, statement1 will be executed

▷ If expression is zero, statement 2 will be executed

```
if (expression1) {statement1}
else if (expression2) {statement2}
```

▷ if expression1 is non-zero, statement1 will be executed

▷ If expression1 is zero and expression2 is non-zero, statement 2 will be executed.

▷ Multiple **else if** clauses are allowed following an **if** clause. Expressions will be evaluated from left to right and until one is non-zero and its corresponding statement is executed. Once a statement is executed, no more expressions in this set will be tested.

```
if (expression1) {statement1}
else if (expression2) {statement2}
...
else {statement3}
```

▷ if expression1 is non-zero, statement1 will be executed

▷ If expression1 is zero and expression2 is non-zero, statement 2 will be executed.

▷ Multiple **else if** clauses are allowed following an **if** clause. Expressions will be evaluated from left to right and until one is non-zero and its corresponding statement is executed. Once a statement is executed, no more expressions in this set will be tested.

▷ If all expressions have evaluated to zero (meaning no statement was executed, statement3 will be executed.

3.5.2 Loops

While Statement

The while statement has the form of

```
while (expression) {statement}
```

The expression is checked first, and whenever its non-zero, the statement will be executed. After that it will return back to check the expression and repeat until the expression is zero.

For Statement

The for statement follows the syntactical form of

```
for(index_variable declaration/instantiation;
   conditional_statement; action) {statement}
```

Before the first iteration, the index variable will be created/initialized. The for statement will then check the conditional statement. If the conditional statement is evaluated to be non-zero, the statement will be executed. Upon completing the statement, the action will be executed. The conditional statement will once again be evaluated. This process will repeat until the conditional statement evaluates to zero.

3.6 Functions

3.6.1 Built-in Functions

Print Functions

There are several print functions, depending on what data type is to be printed.

- `printStr(s);`

▷ prints string *s* to the console.

- `printInt(a);`

▷ prints integer a to the console.

- `printFloat(b);`

▷ prints float b to the console.

- `printBool(c);`

▷ prints boolean c to the console. If c == true, the output is 1. If c == false, the output is 0.

Matrix specific functions

The following functions are applied to matrices. Let M be a delcared variable for a matrix datatype.

- `M:rows`

▷ Calculates the number of rows in matrix M.
▷ Returns an integer

- `M:cols`

▷ Calculates the number of columns in matrix M.
▷ Returns an integer

- `M[i][j]`

▷ Access the element of M at position i, j (row, column).
▷ Returns the element of M at position i, j

- `M:transpose`

▷ Calculates transposed matrix of M.
▷ Returns the transpose of M in a copy, that is, M is not modified.

- `M:trace`

▷ Calculates the trace of matrix M
▷ Returns an integer or float value depending on the entries of the matrix.

- `M:submatrix(index1, index2, index3, index4)`

▷ Calculates a submatrix of matrix M. The upper left corner of the submatrix is indexed by (index1, index3). The lower right corner of the submatrix in indexed by (index2, index4).
▷ Returns a matrix

3.6.2 User-defined Functions

A user can write their own functions to be called in their programs. The format for creating these functions is:

```
return_type function_name(param_type1 param_name1,
param_typeN param_nameN){statement}
```

• return_type: type of variable the function will return. Void if none.

• function_name: the name of the function. It must start with a letter, but the alphabet is defined as $\Sigma : \{A-Z, a-z, 0-9, \_\}$

▷ Functions cannot be overloaded. That is, two functions cannot have the same name even if their parameter list differs.

• param_type1 param_name1, param_typeN param_nameN: List of parameters to be sent into the function. The type for each parameter must be identified as well as the name they will be referenced by within the function.

• statement: the code to be executed when the function is called

All functions must be defined before they can be called.

All code to be executed will be defined in the main() function.

Sample code for declaring a GCD function:

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

3.7 Scope

3.7.1 Scope of Variable

Global Variables

Global variables are defined outside of main() or any function. They can be accessed by the entire program.

Local Variable

Local Variables are defined within main or a function. Copies of local variables can be passed to functions.

Scope of local variables is within the { } brackets in which they are created. For instance, if variable is created in a for loop, it would be deallocated after the for loop has executed.

3.7.2 Scope of functions

New user-defined functions can be defined anywhere, as long as they are in the same file where they are called.

4 Project Plan

4.1 Process

From the start, we decided to meet at least once a week in addition to our weekly TA meeting. We hoped this would help us to try and stay on track through the semester. Often, we worked together remotely, especially for non-coding requirements such as the LRM. Dips in productivity happened, but we always met to at least discuss. Although there were some points that resulted in no tangible progress, our group always met for discussion to try to overcome whatever obstacle was preventing us from moving forward.

4.1.1 Specification

Our group became attached to the idea of writing a matrix based language almost instantly. A lot of our time and energy in the beginning went to exploring the merits of having a programming language for matrices and their operations. This led to a reasonable idea of how we wanted to use the matrix as a data type, what operations a programmer would have access to, and how intuitive some of those operations should be.

After receiving feedback from our proposal, we realized that we had not included the basics of programming itself in our language. Continuing to the LRM, we shifted our focus to make sure that our language was not missing any more basic specifications that would be necessary for someone to write something more than a basic program. Our language has continued to evolve since the LRM milestone, but the ideas at its core remain the same.

4.1.2 Development

Our development process started out with MicroC. We used MicroC as a jumping point to understand how writing a compiler truly functioned. We spent some time going through the files to combine our understanding of what we learned in class to actually applying it in OCaml. By implementing some basic aspects of a programming language, MicroC allowed us to see how the parts of the compiler function and interact in practice. This gave us room to experiment, make changes, and figure out where we went wrong when trying to apply the same basic principles to the more complex components of our language.

We particularly had trouble at points with trying to parse matrix literals and went back and forth along the way on how best to define them to be easily parsable. After many failed attempts, we ended up rearranging some of our parser code to make them work.

About halfway through, we also decided to include a second abstract syntax tree. While it wasn't in MicroC, we did see that it was in the class slides. This was built after semant.ml, so its purpose was to allow additional semantic information to be included and sent into codegen.ml. This made writing the codegen file much easier, as some of this information helped to determine what functionality applied to different components with similar syntax.

4.1.3 Testing

We fell into the trap of not writing our tests immediately. That's not to say we didn't test at all. Instead we did several test to pass cases, and did them manually. The automated test suite consisting of full coverage for unit and integration tests began development closer to the end of our project. We set the basic tests up with adaptations from the test suite of MicroC. We then added the manual tests and devised many test to fail cases to ensure incorrect syntax and semantic elements would not pass through our compiler.

4.2 Programming Style Guide

Our group tried our best to adhere to the following style guidelines:

• Line lengths of no longer than 140 characters.
• 4-space indentation
• Newline between blocks of code to preserve readability
 ▷ A block is a loose definition, essentially a piece of functionality that makes sense to keep together, such as a function or a grammar. Generally they do not exceed 20 lines

```
int CountVariable;
```

• Variable names: capitalize the first letter of each word

```
(* Arithmetic Operators *)
        | '+'  { PLUS }
        | '-'  { MINUS }
        | '*'  { TIMES }
        | '/'  { DIVIDE }
        | '='  { ASSIGN }
        | "++" { INC }
        | "--" { DEC }
```

• Matching alignment of list elements, aligning elements like |, { } , and ->where applicable.
• In cases where tokens are different lengths, they will be lined up at the start of each column, filling with whitespace where necessary.
 ▷ An example of this is illustrated in the Arithmetic Operators code snippet.

4.3 Project Timeline

Table 1

Date	**Milestone and Work Accomplished**
Sep 8 - Sep 27	Language Proposal - ideas for language, heavily focused on matrix specifics
Sep 28 - Oct 16	LRM - nailed down details, fleshed out the rest of the elements to form a complete language
Oct 17 - Nov 8	Hello World - get the bare minimum working
Nov 8 - Dec 1	Team Milestone 1 - get matrix fully working as a datatype
Dec 1 - Dec 8	Team Milestone 2 - work out semantic issues and make final edits to our language specification
Dec 8 - End	Final Presentation/Delivery - implement last functions and ensure project is thoroughly tested

Table 1 outlines the overall path our language development took. Our project was very deadline-focused until after Hello World was due. It was at this point that our team's productivity started to decline, so we decided on creating a couple of our own milestones in order to figure out the path forward. These dates weren't specific deadlines we set, but the dates that things actually happened. We created them to define our next step, so as to not become overwhelmed by this huge project sitting in front of us.

4.4 Roles and Responsibilities

Member	**Original Assigned Role**
Shelley	Tester
Tengyu	System Architect
Jeff	Language Guru
Christine	Manager
Montana	(joined later)

Contribution:

Shelley: Implemented various kinds of checks in static semantic check. Adapted the testall.sh from MicroC as the automation of our test suite. Designed pass tests and fail tests in the test suite.

Tengyu: Implemented the code generator including the matrix literal and matrix operators. Also implemented the built-in functions of matrix using OCaml as well as library functions of matrix using our own language. Devised the coolest program of our language.

Jeff: Implemented sections of parser, abstract syntax tree and semantics. Helped to design the matrix literal, as well as the matrix functions. One of the leaders in designing the syntax/language, as well as provided the basic idea on top of which our project would be based upon (a.k.a. the implementation of matrices and matrix operations/operators, and the possibilities that could give to the user).

Christine: Helped to implement scanner, parser, ast and some work on semantics. Helped to create the matrix literal as well as its grammar. Did a lot of the group's organizational things such as finding meeting space and creating documents

Montana: Participated in a little of everything, including working with code on the front end of the compiler, preforming research tasks concurrently with development, and simulating grammars and shift-reduce automatons by hand.

4.5 Development Environment

We used the following environment to develop M^2:

Programming: OCaml v4.0.5
Code Generation/Optimization: LLVM v3.7
Version control: Git

4.6 Project Log

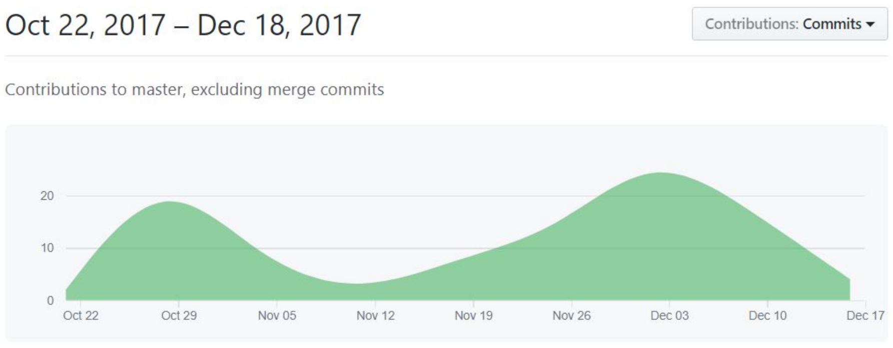


Figure 1: Log of Git commit history from Github

Figure 1 details the overall commit history from the start of our coding. We got started shortly after the LRM milestone on October 16, and worked diligently at the start. The log provides the evidence for our earlier statements that we fell off track for a bit shortly after the hello world program. The generation of our own milestones, as detailed in section 4.3, helped us to overcome this block, which resulted in a large push towards the end. Had we determined some of those earlier, our commit distribution may have been slightly more even.

5 Language Evolution

5.1 Early Stages

When we started, we quickly honed in on the notion of creating a programming language aimed at the purpose of performing matrix operations. However, we had some additional ideas which we never ended up pursuing. In fact, the primary reason we started down the path of matrices was to pursue the idea of outputting graphs. We were drawn by the notion that, if one combines the ability to output graphical components to the screen and do matrix arithmetic, they are not far off from creating a setting to create simple animations.

Realizing it would be a fairly ambitious goal, we put this idea on the back burner fairly quickly. We instead decided to first focus on the complete implementation of matrices. As the semester progressed, it became clear that this was the right choice.

From this point, we quickly started discussing the question of syntax. We all agreed that, while we wanted the language to be as easy as possible to pick up for people with different skill sets, we didn't want to omit too much complexity (e.g. variable types, definitive line breaks, return types, etc). Without some complexity in syntax, the code could become difficult to visually interpret, especially for a seasoned programmer embarking on a larger than average project. This is in direct comparison to languages like Python and Javascript who have no distinct variable types. To this end, we decided to follow a style which borrows heavily from C/Java, as we all felt that these are examples of our favorite forms of syntax.

5.2 Formulating Our Semantics

Deciding on a C-like syntactical style solved a lot of the semantic problems for us. For pretty much everything implemented, one may infer the semantics of our language knowledge of C. The biggest differences are our print functions (we have four, printInt(), printFloat(), printBool(), and printStr()), our decrementation/incrementation operators ($--i$ and $++i$, instead of $i--$ and $i++$), and variable assignment (one must first declare a variables type and id before assignment can occur). However, there was still the decision of how to implement our matrix data structure. We were toying with several ideas, including defining a matrix by taking into account whitespace (namely, tabs and newlines) in order to achieve an implementation which would have the user define a matrix in the following fashion:

```
matrix M =      [1, 2, 3,
                4, 5, 6,
                7, 8, 9];
```

We ended up abandoning these ideas because such implementations seemed less intuitive to implement and didn't seem likely to increase readability or intuitiveness for the user. Thus, we settled upon designing matrix in the image of a 2-dimensional array. Following is an example of a definition of the same matrix from before:

```
matrix int [3][3] m;
```

```
m = [[1, 2, 3]; [4, 5, 6]; [7, 8, 9]];
```

There are several important points here. First, in addition to the type (matrix) and the id (m), one must also declare the type of matrix that they wish to instantiate (M^2 supports matrix types of int or float). Matrix dimensions must be also be specified. These are declared as the number of rows followed by the number of columns. When initializing or assigning to a matrix variable, the dimensions must match those declared. Accessing the elements of a matrix must also be within bounds, following a [row][column] order. For example:

```
int m11;
m11 = m[1][1];
```

will access the first row and the first element of that row, and assign it to the variable m11; The pattern continues from there.

Semantically, we felt this would be very intuitive for the user to understand. Furthermore, there is only one other order in which matrix elements could be accessed - switching the precedence of rows and columns from our implementation. Applied to matrices, the choice between the two seemed somewhat arbitrary, but the choice of giving precedence to rows in both instantiation and assignment also ties into the users (assumed) knowledge of arrays. We felt that keeping a horizontal orientation for the user, as most tend to imagine data structures akin to lists or arrays, would allow for past knowledge of data structures to cross over more seamlessly.

Once we firmly established this as the structure and semantics of our matrix, there was then the question of vectors. Matrix operations are extremely limited without the presence of vectors, so we knew that we would have to include support for them as well. We toyed with several ideas, mostly revolving around creating an entirely separate vector data structure and defining operations between vector and matrix. In the end, this seemed needless. We eventually arrived at the realization that we really didnt need to implement anything new in order to support vectors. This is because vectors can be easily visualized as single column matrices. In fact, they are ofter represented this way in mathematics. Thus,

```
matrix int [3][1] m;
m = [[1]; [0]; [0]];
```

in our language serves to represent a vector as a 3 row x 1 column matrix.

5.3 Matrix Functions and Operators

After this, we set out to establish which operators for matrices and what matrix functions we would build into our language. We talked about a few different sets, but eventually we came up with a working list of operators (+ matrix addition, - matrix subtraction, * matrix multiplication, and ˆmatrix exponent) and functions (determinant, trace, submatrix, inverse, transpose, diagonalization, identity, number of columns, number of rows). We thought that matrix exponent was a little redundant since we knew we would add matrix multiplication, so we decided to drop " ˆ" as a matrix operator. For functions, we decided to remove identity, as the user could easily their own. We also removed diagonalization, determinant and inverse as these processes have somewhat complex algorithms that could often vary depending on the size of the matrix.

This left us with +, -, and * as our matrix operators, and trace, submatrix, transpose, number of columns, and number of rows as our matrix functions. Our initial plan was to make these similar to methods, like in Java, which would look something like

```
m.transpose();
```

in the case of the transpose function. However, we thought this style was too closely related to the notion of object orientation, which our language is not. This draws from the idea that our language is syntactically similar to provide familiarity. Structuring matrix functions like this could give the wrong impression in this case. Therefore, we opted for a simple syntax which wouldnt mix in such notions. Every function follows a similar format,

```
m:trace;                          returns int or float
m:transpose;                      returns matrix
m:subMatrix[r1,r2,c1,c2];         returns matrix
m:cols;                           returns int
m:rows;                           returns int
```

We felt that this would give the user enough functionality while also keeping things relatively intuitive. Matrix operators also follow a simple syntax,

```
m + m;
m - m;  all return type matrix
m * m;
```

In the beginning, we had a goal of not allowing our language to become overly complex. We all agreed that our favorite languages are those which are able to provide functionality, while not steering away from the heritage of the styles of the most widespread and foundational languages in use today (namely, C, C++ and Java). Though we explored various ideas, we feel that we held true to our original goal throughout, and all that fluctuated within our languages design was how best to fit that goal.

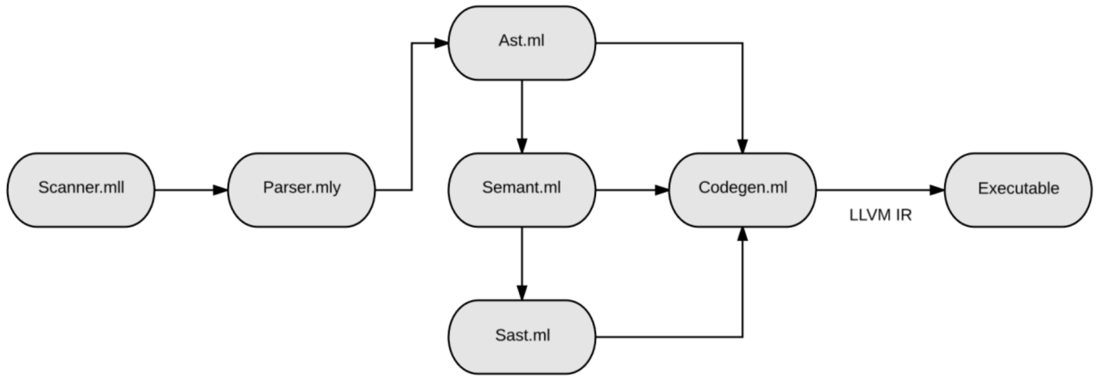
5.4 Problems Encountered

Our largest problems occurred during our implementation of matrix and the matrix functions/operators. To begin, we struggled until a fair amount after the Hello World assignment to actually come up with a definitive literal and grammar for matrix. We decided on utilizing a secondary abstract syntax tree which would be produced by our semant file and also feed into codegen, in order to simplify our implementation. Going from there, we struggled tremendously to weave the necessary components for matrix into every file. Every file has to work at the same time, so it was very difficult to even get past the phase of getting OCaml to compile.

Even when it did compile, we faced two major issues. The first was incorrect outputs within our tests (both for fail tests and pass tests), and the second were OCaml errors during runtime (namely, parsing errors). However, at this stage, getting the code to operate properly was similar to a most programming projects. When OCaml doesn't compile, there are so many interdependencies that its difficult to actually get a handle on where the problem is occurring, but once that is handled it becomes a problem of gaining an understanding of the error messages and debugging bit by bit.

6 Architectural Design

6.1 Components of the compiler



There are six major components in our compiler: semant.mll, parser.mly, ast.ml, semant.ml, sast.ml, and codegen.ml. The entire team worked together on scanner.mll, parser.mly, ast.ml, and sast.ml. Shelley and Tengyu took the lead on implementing semant.ml and codegen.ml. Other team members also contributed to semant.ml and codegen.ml.

Scanner

We use scanner.mll as our scanner to read source code. The source code is treated as a sequence of characters, and the scanner outputs a stream of tokens using lexical analysis. The tokens are defined using explicit characters or a regular expression. The scanner discards whitespace and comments. If any illegal character is in the source code, the scanner will raise an exception.

Parser and AST

We use parser.mly as the parser to build an abstract syntax tree from the tokens. We defined precedence and associativities in the parser to avoid any ambiguity in the grammar. We also define a program to be a list of function declarations and variable declarations in order to perform the static semantic check later.

Static Semantic Check and SAST

Once an abstract syntax tree is built, we perform the static semantic check on the tree to solve any type issues. In semant.ml, we check every declared variable and function in the program. Common checks include duplicate variable, void bindings, mismatched data type, etc. During the static semantic check, we collect type information on different kinds of expressions, and add the information to ast so as to build a sast (semantically checked abstract syntax tree). Using sast provides us with great benefit in the code generation process, since we do not have check the actual types of expressions again.

Code Generator and Executable

codegen.ml is our code generator that generates LLVM IR code. The IR code is then turned into assembly code, which leads to an executable program in the end.

7 Test Plan

7.1 Test Suite and Automation

Our test suite consists of two main sections, test1 and test2. Test1 consists of test cases that were adapted from MicroC to match our language while Test2 was developed for the unique data types and functionalities of M^2. Test2 was primarily developed by our tester, Shelley, but other group members did have input on developing various test cases. The complete test suite is included in the appendix section with a total of 41 fail tests and 57 pass tests.

Our test suite was automated through a shell script, named testall.sh. To execute testing, testall.sh should be placed in the directory that contains the folders src and tests. The the user should type ./testall.sh in command line and it will run everything within the tests folder. In addition to the tests written in M^2, the tests folder includes files with the expected output for each test. This allows the script to not only run the tests, but act as the test oracle as well. The script will check whether the actual test output matches what is expected, and if it does, the test passes. This the largest benefit of automated testing. Figure 2 shows an example output of a successful test suite run.

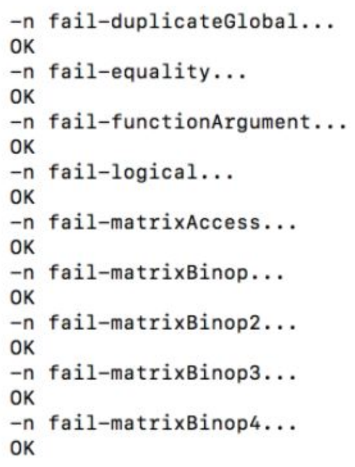

```
-n fail-duplicateGlobal...
OK
-n fail-equality...
OK
-n fail-functionArgument...
OK
-n fail-logical...
OK
-n fail-matrixAccess...
OK
-n fail-matrixBinop...
OK
-n fail-matrixBinop2...
OK
-n fail-matrixBinop3...
OK
-n fail-matrixBinop4...
OK
```

Figure 2: Successful run of several test cases

7.2 Test Cases

Our test cases fall into two main categories: Pass/Fail and Unit/Integration.

7.2.1 Pass/Fail

The pass tests and the fail tests are separated into their own subdirectories within each test folder.

Pass tests are designed to pass, as one might expect. The purpose of these tests is to ensure that our language functions the way its described in section 3. In essence, the tests are designed to make sure everything works the way we want it to. The expected output of the pass tests are contained in their corresponding .out files, which contains the expected output should the program run successfully. It does not compare the target source code that is generated.

Fail tests, on the other hand, are designed to ensure that incorrect semantics do not pass through the compiler. Many of our fail tests focus on testing our static semantic checking. For instance, if logical operators are not allowed to work on matrices, the test containing such a case should print the correct error message. This will then be checked against the tests corresponding .err file, and if the output error matches, the fail test passes.

7.2.2 Unit/Integration

The other category of testing we focus on is unit vs. integration testing. We have several unit tests that test only the functionality of single or multiple small components. This allows us to pinpoint the problem faster that by testing a large program.

Ideally, users of M^2 will use it for larger programs, which is why integration testing is important as well. This tests that combining the smaller, already tested units doesn't introduce new errors not previously caught. Generally, integration tests are pass tests, because testing to fail really only make sense when there is one known error in the code. Unlike pass/fail, unit and integration tests are not explicitly defined in our test suite.

7.3 Example of tests

7.3.1 Test 1

The following test is a unit test used to test semantic checking on types for the equality operator

```
int main(){
        matrix int [1][1] a;
        matrix int [1][1] b;
        a = [[1]];
        b = [[2]];
        a == b;
        return 0;
}
```

The output of the test is

```
Fatal error: exception Failure("Illegal type for equality
   operators")
```

Since this is a fail test, no LLVM IR is generated for the test.

7.3.2 Test 2

The following test is a larger integration test created to solve a simple system of linear equations.

```
int main() {
  matrix float [3][3] m;
  matrix float [3][1] m2;
  matrix float [3][3] tr;
  matrix float [3][3] adj;
  matrix float [3][3] inverse;
  matrix float [3][1] result;
  float det;
  int i;
  int j;

  m = [[1.0, 1.0, 1.0];
       [2.0, 3.0, 1.0];
       [1.0, 1.0, 5.0]];

  m2 = [[3.0];
        [6.0];
        [7.0]];

  tr = m:transpose;

  det = m[0][0] * (m[1][1] * m[2][2] - m[1][2] * m[2][1])
    - m[0][1] * (m[1][0] * m[2][2] - m[1][2] * m[2][0])
    + m[0][2] * (m[1][0] * m[2][1] - m[1][1] * m[2][0]);

  if (det != 0.0){
    adj[0][0] = tr[1][1]*tr[2][2] - tr[1][2]*tr[2][1];
    adj[0][1] = tr[1][0]*tr[2][2] - tr[1][2]*tr[2][0];
    adj[0][2] = tr[1][0]*tr[2][1] - tr[1][1]*tr[2][0];
    adj[1][0] = tr[0][1]*tr[2][2] - tr[0][2]*tr[2][1];
    adj[1][1] = tr[0][0]*tr[2][2] - tr[0][2]*tr[2][0];
    adj[1][2] = tr[0][0]*tr[2][1] - tr[0][1]*tr[2][0];
    adj[2][0] = tr[0][1]*tr[1][2] - tr[0][2]*tr[1][1];
    adj[2][1] = tr[0][0]*tr[1][2] - tr[0][2]*tr[1][0];
    adj[2][2] = tr[0][0]*tr[1][1] - tr[0][1]*tr[1][0];


    adj[0][1] = adj[0][1] * (0.0-1.0);
    adj[1][0] = adj[1][0] * (0.0-1.0);
    adj[1][2] = adj[1][2] * (0.0-1.0);
    adj[2][1] = adj[2][1] * (0.0-1.0);

    for(i = 0; i < inverse:rows; ++i){
      for (j = 0; j < inverse:cols; ++j){
        inverse[i][j] = adj[i][j] / det;
      }
    }

    result = inverse * m2;

    printStr("Input equations are:");

    for (i = 0; i < m:rows; ++i){
      printFloat(m[i][0]);
      printStr("x + ");
      printFloat(m[i][1]);
      printStr("y ");
      printFloat(m[i][2]);
      printStr("z = ");
      printFloat(m2[i][0]);
      printStr("");
      printStr("");
    }

    printStr("Solutions are:");
    printStr("x = ");
    printFloat(result[0][0]);
    printStr("");
    printStr("y = ");
    printFloat(result[1][0]);
    printStr("");
    printStr("z = ");
    printFloat(result[2][0]);

  }
  else{
    printStr("Cannot find a solution");
  }

  return 0;
}
```

```
void myprint(matrix float [3][1] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
```

The output of the test is

```
Input equations are:
1.000000        x +
1.000000        y
1.000000        z =
3.000000

2.000000        x +
3.000000        y
1.000000        z =
6.000000

1.000000        x +
1.000000        y
5.000000        z =
7.000000

Solutions are:
x =
1.000000
y =
1.000000
z =
1.000000
```

The LLVM IR generated for the test is

```
; ModuleID = 'M2'

@fmt = private unnamed_addr constant [4 x i8] c"%d\09\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%f\09\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@tmp = private unnamed_addr constant [1 x i8] zeroinitializer
@fmt.3 = private unnamed_addr constant [4 x i8] c"%d\09\00"
```

```
@fmt.4 = private unnamed_addr constant [4 x i8] c"%f\09\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@tmp.6 = private unnamed_addr constant [21 x i8] c"Input
   equations are:\00"
@tmp.7 = private unnamed_addr constant [5 x i8] c"x + \00"
@tmp.8 = private unnamed_addr constant [3 x i8] c"y \00"
@tmp.9 = private unnamed_addr constant [5 x i8] c"z = \00"
@tmp.10 = private unnamed_addr constant [1 x i8]
   zeroinitializer
@tmp.11 = private unnamed_addr constant [1 x i8]
   zeroinitializer
@tmp.12 = private unnamed_addr constant [15 x i8] c"Solutions
   are:\00"
@tmp.13 = private unnamed_addr constant [5 x i8] c"x = \00"
@tmp.14 = private unnamed_addr constant [1 x i8]
   zeroinitializer
@tmp.15 = private unnamed_addr constant [5 x i8] c"y = \00"
@tmp.16 = private unnamed_addr constant [1 x i8]
   zeroinitializer
@tmp.17 = private unnamed_addr constant [5 x i8] c"z = \00"
@tmp.18 = private unnamed_addr constant [23 x i8] c"Cannot
   find a solution\00"

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

define void @myprint([3 x [1 x double]] %M) {
entry:
  %M1 = alloca [3 x [1 x double]]
  store [3 x [1 x double]] %M, [3 x [1 x double]]* %M1
  %i = alloca i32
  %j = alloca i32
  store i32 0, i32* %i
  br label %while

while:                                          ; preds =
   %merge, %entry
  %i14 = load i32, i32* %i
  %tmp15 = icmp slt i32 %i14, 3
  br i1 %tmp15, label %while_body, label %merge16

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

while2:                                         ; preds =
   %while_body3, %while_body
  %j9 = load i32, i32* %j
  %tmp10 = icmp slt i32 %j9, 1
  br i1 %tmp10, label %while_body3, label %merge

while_body3:                                      ; preds =
   %while2
  %i4 = load i32, i32* %i
  %j5 = load i32, i32* %j
  %M6 = getelementptr [3 x [1 x double]], [3 x [1 x double]]*
     %M1, i32 0, i32 %i4, i32 %j5
  %M7 = load double, double* %M6
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0),
     double %M7)
  %j8 = load i32, i32* %j
  %tmp = add i32 %j8, 1
  store i32 %tmp, i32* %j
  br label %while2

merge:                                            ; preds =
   %while2
  %printf11 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8*
     getelementptr inbounds ([1 x i8], [1 x i8]* @tmp, i32 0,
     i32 0))
  %i12 = load i32, i32* %i
  %tmp13 = add i32 %i12, 1
  store i32 %tmp13, i32* %i
  br label %while

merge16:                                          ; preds =
   %while
  ret void
}

define i32 @main() {
entry:
  %m = alloca [3 x [3 x double]]
  %m2 = alloca [3 x [1 x double]]
  %tr = alloca [3 x [3 x double]]
  %adj = alloca [3 x [3 x double]]
  %inverse = alloca [3 x [3 x double]]
  %result = alloca [3 x [1 x double]]
  %det = alloca double
  %i = alloca i32
  %j = alloca i32
  store [3 x [3 x double]] [[3 x double] [double
     1.000000e+00, double 1.000000e+00, double 1.000000e+00],
     [3 x double] [double 2.000000e+00, double 3.000000e+00,
     double 1.000000e+00], [3 x double] [double 1.000000e+00,
     double 1.000000e+00, double 5.000000e+00]], [3 x [3 x
     double]]* %m
  store [3 x [1 x double]] [[1 x double] [double
     3.000000e+00], [1 x double] [double 6.000000e+00], [1 x
     double] [double 7.000000e+00]], [3 x [1 x double]]* %m2
  %tmpmat = alloca [3 x [3 x double]]
  %m1 = getelementptr [3 x [3 x double]], [3 x [3 x double]]*
     %m, i32 0, i32 0, i32 0
  %m3 = load double, double* %m1
  %tmpmat4 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tmpmat, i32 0, i32 0, i32 0
  store double %m3, double* %tmpmat4
  %m5 = getelementptr [3 x [3 x double]], [3 x [3 x double]]*
     %m, i32 0, i32 0, i32 1
  %m6 = load double, double* %m5
  %tmpmat7 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tmpmat, i32 0, i32 1, i32 0
  store double %m6, double* %tmpmat7
  %m8 = getelementptr [3 x [3 x double]], [3 x [3 x double]]*
     %m, i32 0, i32 0, i32 2
  %m9 = load double, double* %m8
  %tmpmat10 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tmpmat, i32 0, i32 2, i32 0
  store double %m9, double* %tmpmat10
  %m11 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 1, i32 0
  %m12 = load double, double* %m11
  %tmpmat13 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tmpmat, i32 0, i32 0, i32 1
  store double %m12, double* %tmpmat13
  %m14 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 1, i32 1
  %m15 = load double, double* %m14
  %tmpmat16 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tmpmat, i32 0, i32 1, i32 1
  store double %m15, double* %tmpmat16
  %m17 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 1, i32 2
  %m18 = load double, double* %m17
  %tmpmat19 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tmpmat, i32 0, i32 2, i32 1
  store double %m18, double* %tmpmat19
  %m20 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 2, i32 0
  %m21 = load double, double* %m20
  %tmpmat22 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tmpmat, i32 0, i32 0, i32 2
```

```
store double %m21, double* %tmpmat22
%m23 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 2, i32 1
%m24 = load double, double* %m23
%tmpmat25 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tmpmat, i32 0, i32 1, i32 2
store double %m24, double* %tmpmat25
%m26 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 2, i32 2
%m27 = load double, double* %m26
%tmpmat28 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tmpmat, i32 0, i32 2, i32 2
store double %m27, double* %tmpmat28
%tmpmat29 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tmpmat, i32 0
%tmpmat30 = load [3 x [3 x double]], [3 x [3 x double]]*
   %tmpmat29
store [3 x [3 x double]] %tmpmat30, [3 x [3 x double]]* %tr
%m31 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 0, i32 0
%m32 = load double, double* %m31
%m33 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 1, i32 1
%m34 = load double, double* %m33
%m35 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 2, i32 2
%m36 = load double, double* %m35
%tmp = fmul double %m34, %m36
%m37 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 1, i32 2
%m38 = load double, double* %m37
%m39 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 2, i32 1
%m40 = load double, double* %m39
%tmp41 = fmul double %m38, %m40
%tmp42 = fsub double %tmp, %tmp41
%tmp43 = fmul double %m32, %tmp42
%m44 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 0, i32 1
%m45 = load double, double* %m44
%m46 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 1, i32 0
%m47 = load double, double* %m46
%m48 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %m, i32 0, i32 2, i32 2
%m49 = load double, double* %m48
%tmp50 = fmul double %m47, %m49
%m51 = getelementptr [3 x [3 x double]], [3 x [3 x
```

```
     double]]* %m, i32 0, i32 1, i32 2
  %m52 = load double, double* %m51
  %m53 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 2, i32 0
  %m54 = load double, double* %m53
  %tmp55 = fmul double %m52, %m54
  %tmp56 = fsub double %tmp50, %tmp55
  %tmp57 = fmul double %m45, %tmp56
  %tmp58 = fsub double %tmp43, %tmp57
  %m59 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 0, i32 2
  %m60 = load double, double* %m59
  %m61 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 1, i32 0
  %m62 = load double, double* %m61
  %m63 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 2, i32 1
  %m64 = load double, double* %m63
  %tmp65 = fmul double %m62, %m64
  %m66 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 1, i32 1
  %m67 = load double, double* %m66
  %m68 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 2, i32 0
  %m69 = load double, double* %m68
  %tmp70 = fmul double %m67, %m69
  %tmp71 = fsub double %tmp65, %tmp70
  %tmp72 = fmul double %m60, %tmp71
  %tmp73 = fadd double %tmp58, %tmp72
  store double %tmp73, double* %det
  %det74 = load double, double* %det
  %tmp75 = fcmp one double %det74, 0.000000e+00
  br i1 %tmp75, label %then, label %else

merge:                                           ; preds =
   %else, %merge320
  ret i32 0

then:                                            ; preds =
   %entry
  %adj76 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 0, i32 0
  %tr77 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tr, i32 0, i32 1, i32 1
  %tr78 = load double, double* %tr77
  %tr79 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tr, i32 0, i32 2, i32 2
  %tr80 = load double, double* %tr79
```

```
%tmp81 = fmul double %tr78, %tr80
%tr82 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 2
%tr83 = load double, double* %tr82
%tr84 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 1
%tr85 = load double, double* %tr84
%tmp86 = fmul double %tr83, %tr85
%tmp87 = fsub double %tmp81, %tmp86
store double %tmp87, double* %adj76
%adj88 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %adj, i32 0, i32 0, i32 1
%tr89 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 0
%tr90 = load double, double* %tr89
%tr91 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 2
%tr92 = load double, double* %tr91
%tmp93 = fmul double %tr90, %tr92
%tr94 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 2
%tr95 = load double, double* %tr94
%tr96 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 0
%tr97 = load double, double* %tr96
%tmp98 = fmul double %tr95, %tr97
%tmp99 = fsub double %tmp93, %tmp98
store double %tmp99, double* %adj88
%adj100 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %adj, i32 0, i32 0, i32 2
%tr101 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 0
%tr102 = load double, double* %tr101
%tr103 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 1
%tr104 = load double, double* %tr103
%tmp105 = fmul double %tr102, %tr104
%tr106 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 1
%tr107 = load double, double* %tr106
%tr108 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 0
%tr109 = load double, double* %tr108
%tmp110 = fmul double %tr107, %tr109
%tmp111 = fsub double %tmp105, %tmp110
store double %tmp111, double* %adj100
%adj112 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %adj, i32 0, i32 1, i32 0
```

```
%tr113 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 1
%tr114 = load double, double* %tr113
%tr115 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 2
%tr116 = load double, double* %tr115
%tmp117 = fmul double %tr114, %tr116
%tr118 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 2
%tr119 = load double, double* %tr118
%tr120 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 1
%tr121 = load double, double* %tr120
%tmp122 = fmul double %tr119, %tr121
%tmp123 = fsub double %tmp117, %tmp122
store double %tmp123, double* %adj112
%adj124 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %adj, i32 0, i32 1, i32 1
%tr125 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 0
%tr126 = load double, double* %tr125
%tr127 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 2
%tr128 = load double, double* %tr127
%tmp129 = fmul double %tr126, %tr128
%tr130 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 2
%tr131 = load double, double* %tr130
%tr132 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 0
%tr133 = load double, double* %tr132
%tmp134 = fmul double %tr131, %tr133
%tmp135 = fsub double %tmp129, %tmp134
store double %tmp135, double* %adj124
%adj136 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %adj, i32 0, i32 1, i32 2
%tr137 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 0
%tr138 = load double, double* %tr137
%tr139 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 1
%tr140 = load double, double* %tr139
%tmp141 = fmul double %tr138, %tr140
%tr142 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 1
%tr143 = load double, double* %tr142
%tr144 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 2, i32 0
```

```
%tr145 = load double, double* %tr144
%tmp146 = fmul double %tr143, %tr145
%tmp147 = fsub double %tmp141, %tmp146
store double %tmp147, double* %adj136
%adj148 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %adj, i32 0, i32 2, i32 0
%tr149 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 1
%tr150 = load double, double* %tr149
%tr151 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 2
%tr152 = load double, double* %tr151
%tmp153 = fmul double %tr150, %tr152
%tr154 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 2
%tr155 = load double, double* %tr154
%tr156 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 1
%tr157 = load double, double* %tr156
%tmp158 = fmul double %tr155, %tr157
%tmp159 = fsub double %tmp153, %tmp158
store double %tmp159, double* %adj148
%adj160 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %adj, i32 0, i32 2, i32 1
%tr161 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 0
%tr162 = load double, double* %tr161
%tr163 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 2
%tr164 = load double, double* %tr163
%tmp165 = fmul double %tr162, %tr164
%tr166 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 2
%tr167 = load double, double* %tr166
%tr168 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 0
%tr169 = load double, double* %tr168
%tmp170 = fmul double %tr167, %tr169
%tmp171 = fsub double %tmp165, %tmp170
store double %tmp171, double* %adj160
%adj172 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %adj, i32 0, i32 2, i32 2
%tr173 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 0, i32 0
%tr174 = load double, double* %tr173
%tr175 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %tr, i32 0, i32 1, i32 1
%tr176 = load double, double* %tr175
```

```
  %tmp177 = fmul double %tr174, %tr176
  %tr178 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tr, i32 0, i32 0, i32 1
  %tr179 = load double, double* %tr178
  %tr180 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %tr, i32 0, i32 1, i32 0
  %tr181 = load double, double* %tr180
  %tmp182 = fmul double %tr179, %tr181
  %tmp183 = fsub double %tmp177, %tmp182
  store double %tmp183, double* %adj172
  %adj184 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 0, i32 1
  %adj185 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 0, i32 1
  %adj186 = load double, double* %adj185
  %tmp187 = fmul double %adj186, -1.000000e+00
  store double %tmp187, double* %adj184
  %adj188 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 1, i32 0
  %adj189 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 1, i32 0
  %adj190 = load double, double* %adj189
  %tmp191 = fmul double %adj190, -1.000000e+00
  store double %tmp191, double* %adj188
  %adj192 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 1, i32 2
  %adj193 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 1, i32 2
  %adj194 = load double, double* %adj193
  %tmp195 = fmul double %adj194, -1.000000e+00
  store double %tmp195, double* %adj192
  %adj196 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 2, i32 1
  %adj197 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 2, i32 1
  %adj198 = load double, double* %adj197
  %tmp199 = fmul double %adj198, -1.000000e+00
  store double %tmp199, double* %adj196
  store i32 0, i32* %i
  br label %while

while:                                            ; preds =
   %merge215, %then
  %i218 = load i32, i32* %i
  %tmp219 = icmp slt i32 %i218, 3
  br i1 %tmp219, label %while_body, label %merge220

while_body:                                       ; preds =
   %while
  store i32 0, i32* %j
  br label %while200

while200:                                        ; preds =
   %while_body201, %while_body
  %j213 = load i32, i32* %j
  %tmp214 = icmp slt i32 %j213, 3
  br i1 %tmp214, label %while_body201, label %merge215

while_body201:                                   ; preds =
   %while200
  %i202 = load i32, i32* %i
  %j203 = load i32, i32* %j
  %inverse204 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %inverse, i32 0, i32 %i202, i32 %j203
  %i205 = load i32, i32* %i
  %j206 = load i32, i32* %j
  %adj207 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %adj, i32 0, i32 %i205, i32 %j206
  %adj208 = load double, double* %adj207
  %det209 = load double, double* %det
  %tmp210 = fdiv double %adj208, %det209
  store double %tmp210, double* %inverse204
  %j211 = load i32, i32* %j
  %tmp212 = add i32 %j211, 1
  store i32 %tmp212, i32* %j
  br label %while200

merge215:                                        ; preds =
   %while200
  %i216 = load i32, i32* %i
  %tmp217 = add i32 %i216, 1
  store i32 %tmp217, i32* %i
  br label %while

merge220:                                        ; preds =
   %while
  %inverse221 = load [3 x [3 x double]], [3 x [3 x double]]*
     %inverse
  %m2222 = load [3 x [1 x double]], [3 x [1 x double]]* %m2
  %tmpmat223 = alloca [3 x [1 x double]]
  %tmpproduct = alloca double
  store double 0.000000e+00, double* %tmpproduct
  store double 0.000000e+00, double* %tmpproduct
  %inverse224 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %inverse, i32 0, i32 0, i32 0
  %inverse225 = load double, double* %inverse224
```

```
%m2226 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 0, i32 0
%m2227 = load double, double* %m2226
%tmp228 = fmul double %inverse225, %m2227
%addtmp = load double, double* %tmpproduct
%tmp229 = fadd double %tmp228, %addtmp
store double %tmp229, double* %tmpproduct
%inverse230 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %inverse, i32 0, i32 0, i32 1
%inverse231 = load double, double* %inverse230
%m2232 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 1, i32 0
%m2233 = load double, double* %m2232
%tmp234 = fmul double %inverse231, %m2233
%addtmp235 = load double, double* %tmpproduct
%tmp236 = fadd double %tmp234, %addtmp235
store double %tmp236, double* %tmpproduct
%inverse237 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %inverse, i32 0, i32 0, i32 2
%inverse238 = load double, double* %inverse237
%m2239 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 2, i32 0
%m2240 = load double, double* %m2239
%tmp241 = fmul double %inverse238, %m2240
%addtmp242 = load double, double* %tmpproduct
%tmp243 = fadd double %tmp241, %addtmp242
store double %tmp243, double* %tmpproduct
%tmpmat244 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %tmpmat223, i32 0, i32 0, i32 0
%restmp = load double, double* %tmpproduct
store double %restmp, double* %tmpmat244
store double 0.000000e+00, double* %tmpproduct
%inverse245 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %inverse, i32 0, i32 1, i32 0
%inverse246 = load double, double* %inverse245
%m2247 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 0, i32 0
%m2248 = load double, double* %m2247
%tmp249 = fmul double %inverse246, %m2248
%addtmp250 = load double, double* %tmpproduct
%tmp251 = fadd double %tmp249, %addtmp250
store double %tmp251, double* %tmpproduct
%inverse252 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %inverse, i32 0, i32 1, i32 1
%inverse253 = load double, double* %inverse252
%m2254 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 1, i32 0
%m2255 = load double, double* %m2254
```

```
%tmp256 = fmul double %inverse253, %m2255
%addtmp257 = load double, double* %tmpproduct
%tmp258 = fadd double %tmp256, %addtmp257
store double %tmp258, double* %tmpproduct
%inverse259 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %inverse, i32 0, i32 1, i32 2
%inverse260 = load double, double* %inverse259
%m2261 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 2, i32 0
%m2262 = load double, double* %m2261
%tmp263 = fmul double %inverse260, %m2262
%addtmp264 = load double, double* %tmpproduct
%tmp265 = fadd double %tmp263, %addtmp264
store double %tmp265, double* %tmpproduct
%tmpmat266 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %tmpmat223, i32 0, i32 1, i32 0
%restmp267 = load double, double* %tmpproduct
store double %restmp267, double* %tmpmat266
store double 0.000000e+00, double* %tmpproduct
%inverse268 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %inverse, i32 0, i32 2, i32 0
%inverse269 = load double, double* %inverse268
%m2270 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 0, i32 0
%m2271 = load double, double* %m2270
%tmp272 = fmul double %inverse269, %m2271
%addtmp273 = load double, double* %tmpproduct
%tmp274 = fadd double %tmp272, %addtmp273
store double %tmp274, double* %tmpproduct
%inverse275 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %inverse, i32 0, i32 2, i32 1
%inverse276 = load double, double* %inverse275
%m2277 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 1, i32 0
%m2278 = load double, double* %m2277
%tmp279 = fmul double %inverse276, %m2278
%addtmp280 = load double, double* %tmpproduct
%tmp281 = fadd double %tmp279, %addtmp280
store double %tmp281, double* %tmpproduct
%inverse282 = getelementptr [3 x [3 x double]], [3 x [3 x
   double]]* %inverse, i32 0, i32 2, i32 2
%inverse283 = load double, double* %inverse282
%m2284 = getelementptr [3 x [1 x double]], [3 x [1 x
   double]]* %m2, i32 0, i32 2, i32 0
%m2285 = load double, double* %m2284
%tmp286 = fmul double %inverse283, %m2285
%addtmp287 = load double, double* %tmpproduct
%tmp288 = fadd double %tmp286, %addtmp287
```

```
  store double %tmp288, double* %tmpproduct
  %tmpmat289 = getelementptr [3 x [1 x double]], [3 x [1 x
     double]]* %tmpmat223, i32 0, i32 2, i32 0
  %restmp290 = load double, double* %tmpproduct
  store double %restmp290, double* %tmpmat289
  %tmpmat291 = getelementptr [3 x [1 x double]], [3 x [1 x
     double]]* %tmpmat223, i32 0
  %tmpmat292 = load [3 x [1 x double]], [3 x [1 x double]]*
     %tmpmat291
  store [3 x [1 x double]] %tmpmat292, [3 x [1 x double]]*
     %result
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([21 x i8], [21 x i8]* @tmp.6,
     i32 0, i32 0))
  store i32 0, i32* %i
  br label %while293

while293:                                        ; preds =
   %while_body294, %merge220
  %i318 = load i32, i32* %i
  %tmp319 = icmp slt i32 %i318, 3
  br i1 %tmp319, label %while_body294, label %merge320

while_body294:                                   ; preds =
   %while293
  %i295 = load i32, i32* %i
  %m296 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 %i295, i32 0
  %m297 = load double, double* %m296
  %printf298 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0),
     double %m297)
  %printf299 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([5 x i8], [5 x i8]* @tmp.7, i32
     0, i32 0))
  %i300 = load i32, i32* %i
  %m301 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 %i300, i32 1
  %m302 = load double, double* %m301
  %printf303 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0),
     double %m302)
  %printf304 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([3 x i8], [3 x i8]* @tmp.8, i32
     0, i32 0))
  %i305 = load i32, i32* %i
  %m306 = getelementptr [3 x [3 x double]], [3 x [3 x
     double]]* %m, i32 0, i32 %i305, i32 2
  %m307 = load double, double* %m306
  %printf308 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0),
     double %m307)
  %printf309 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([5 x i8], [5 x i8]* @tmp.9, i32
     0, i32 0))
  %i310 = load i32, i32* %i
  %m2311 = getelementptr [3 x [1 x double]], [3 x [1 x
     double]]* %m2, i32 0, i32 %i310, i32 0
  %m2312 = load double, double* %m2311
  %printf313 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0),
     double %m2312)
  %printf314 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([1 x i8], [1 x i8]* @tmp.10, i32
     0, i32 0))
  %printf315 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([1 x i8], [1 x i8]* @tmp.11, i32
     0, i32 0))
  %i316 = load i32, i32* %i
  %tmp317 = add i32 %i316, 1
  store i32 %tmp317, i32* %i
  br label %while293

merge320:                                         ; preds =
   %while293
  %printf321 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([15 x i8], [15 x i8]* @tmp.12,
     i32 0, i32 0))
  %printf322 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([5 x i8], [5 x i8]* @tmp.13, i32
     0, i32 0))
  %result323 = getelementptr [3 x [1 x double]], [3 x [1 x
     double]]* %result, i32 0, i32 0, i32 0
  %result324 = load double, double* %result323
  %printf325 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0),
     double %result324)
  %printf326 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([1 x i8], [1 x i8]* @tmp.14, i32
     0, i32 0))
  %printf327 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([5 x i8], [5 x i8]* @tmp.15, i32
     0, i32 0))
  %result328 = getelementptr [3 x [1 x double]], [3 x [1 x
     double]]* %result, i32 0, i32 1, i32 0
  %result329 = load double, double* %result328
  %printf330 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0),
     double %result329)
  %printf331 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([1 x i8], [1 x i8]* @tmp.16, i32
     0, i32 0))
  %printf332 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([5 x i8], [5 x i8]* @tmp.17, i32
     0, i32 0))
  %result333 = getelementptr [3 x [1 x double]], [3 x [1 x
     double]]* %result, i32 0, i32 2, i32 0
  %result334 = load double, double* %result333
  %printf335 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0),
     double %result334)
  br label %merge

else:                                              ; preds =
   %entry
  %printf336 = call i32 (i8*, ...) @printf(i8* getelementptr
     inbounds ([4 x i8], [4 x i8]* @fmt.5, i32 0, i32 0), i8*
     getelementptr inbounds ([23 x i8], [23 x i8]* @tmp.18,
     i32 0, i32 0))
  br label %merge
}
```

8 Lessons Learned

8.1 Most Important Learning

Jeffrey Monahan

The most important lesson Jeffrey learned is to be truthful with his teammates and with himself about where the group is, and how it is progressing. A lot of the time, it is easy to lull oneself into a false state of comfort, due to the project lasting so long and it seeming like there is a plethora of time. There is a good amount of time for the project, but the semester always passes by quicker than one thinks. It is important that the group continuously progresses, and if the group reaches a point where that is not happening, one needs to be able to take notice and address it. As long as the group makes incremental progress, the project is not as daunting as it seems.

Christine Pape

Christine learned that sitting down, working problems out, and coding together can be a really effective way of problem solving. One person can figure out one problem which could provide a direct path to the next person figuring out another. There are parts of our project that may have not worked or at least taken a lot longer without this approach.

Montana St. Pierre

An invaluable skill Montana learned while working with the M^2 development team was not concerned with the importance of group dynamics, but rather how the individuals must yield themselves to the team through conscientious introspection. Frequent communication was not enough to establish an effective and productive environment. Each participant needed to overcome preexisting fears or self-fulfilling prophecies and allow themselves to become vulnerable. Criticism should never antagonize a peer and should not be difficult and uncomfortable to produce. Breaking down other similar inhibitions was essential to promoting a safe, interactive, and pragmatic discussion. This was not to say to be unnaturally friendly with a teammate if the situations were unwarranted, but to instead be always comfortable in their assemblage and assured of their capabilities. Montana uncovered these simple tenets after the team was gracious enough to absorb him from his previously dissolved project, and it was essential for him to overcome apprehensions of overstepping when proffering his ideas.

Shelley Zhong

Shelley learned the importance of teamwork. This is not an individual project, so it can be very challenging. Be sure to talk to one's teammates and work collaboratively. Teammates can be a great source of inspiration! Learning OCaml can be very painful, but it is definitely very useful in building a compiler.

Tengyu Zhou

Tengyu learned in order to handle the hard projects and all the finals, the group has to prepare themselves everyday. They need to learn as mush as possible, and build solid fundamentals. Furthermore, teamwork is the most important component for a successful project.

8.2 Advice for Future Teams

Beside the obvious answer of starting early, a good practice includes stepping back and evaluating where the team is week to week. The sooner the project falls off track, the harder it is to get back on, exemplifying the importance of self-awareness.

9 Appendix

9.1 scanner.mll

```
(*
        Jeffrey Monahan         - jm4155
        Christine Pape          - cmp2223
        Montana St. Pierre      - mrs2296
        Shelley Zhong           - sz2699
        Tengyu Zhou             - tz2338
*)
{ open Parser }

rule token = parse
        (* Whitespace *)
        [' ' '\t' '\r' '\n'] { token lexbuf }

        (* Comments *)
        | "/*"     { blockComment lexbuf }
        | "//"     { singleComment lexbuf }

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

        (* Arithmetic Operators *)
        | '+'  { PLUS }
        | '-'  { MINUS }
        | '*'  { TIMES }
        | '/'  { DIVIDE }
        | '='  { ASSIGN }
        | "++" { INC }
        | "--" { DEC }

        (* Relational Operators *)
        | "==" { EQ }
        | "!=" { NEQ }
        | '<'  { LT }
        | ">"  { GT }
        | "<=" { LEQ }
        | ">=" { GEQ }

        (* Logical Operators *)
        | "&&" { AND }
        | "||" { OR }
        | '!'  { NOT }

        (* Control Flow *)
        | "if"     { IF }
        | "else"   { ELSE }
        | "while"  { WHILE }
        | "for"    { FOR }
        | "return" { RETURN }

        (* Boolean Values *)
        | "true"  { TRUE }
        | "false" { FALSE }

        (* Data Types *)
        | "int"    { INT }
        | "float"  { FLOAT }
        | "bool"   { BOOL }
        | "void"   { VOID }
        | "String" { STRING }
        | "matrix" { MATRIX }

        (* Matrix-related *)
        | "rows" { ROWS }
        | "cols" { COLS }
        | "transpose"   { TRANSPOSE }
        | "trace" { TRACE }
        | "subMatrix" { SUBMATRIX }

        (* Literals, Identifiers, EOF *)
        | ['0'-'9']+ as lxm {
           NUM_LIT(Ast.IntLit(int_of_string lxm)) }
        | ['0'-'9']+ '.' ['0'-'9']+ as lxm {
           NUM_LIT(Ast.FloatLit(float_of_string lxm)) }
        | '"' (([^ '"'] | "\\\"")* as str) '"' {
           STRING_LIT(str) }
        | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as
           lxm { ID(lxm) }
        | eof { EOF }
        | _ as char { raise (Failure("illegal character " ^
           Char.escaped char)) }

and blockComment = parse
  "*/" { token lexbuf }
```

```
        | _    { blockComment lexbuf }

and singleComment = parse
  ['\n' '\r'] { token lexbuf}
        | _    { singleComment lexbuf }
```

9.2 parser.mly

```
/*
        Jeffrey Monahan         - jm4155
        Christine Pape          - cmp2223
        Montana St. Pierre      - mrs2296
        Shelley Zhong           - sz2699
        Tengyu Zhou             - tz2338
*/
%{ open Ast %}

/* Delimiters */
%token LPAREN RPAREN LBRACKET RBRACKET LBRACE RBRACE
%token SEMI COMMA COLON

/* Arithmetic Operators */
%token PLUS MINUS TIMES DIVIDE ASSIGN INC DEC

/* Relational Operators */
%token EQ NEQ LT GT LEQ GEQ

/* Logical Operators */
%token AND OR NOT

/* Control Flow */
%token IF ELSE WHILE FOR RETURN

/* Boolean Values */
%token TRUE FALSE

/* Data Types */
%token INT FLOAT BOOL VOID STRING MATRIX

/* Matrix-related */
%token  ROWS COLS TRANSPOSE TRACE SUBMATRIX

/* Literals, Identifiers, EOF */
%token <Ast.num> NUM_LIT
%token <string> STRING_LIT
%token <string> ID
%token EOF
```

```
/* Precedence and associativity of each operator */
%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS INC DEC
%left TIMES DIVIDE
%right NOT NEG

%start program
%type <Ast.program> program



program:
    decls EOF { $1 }

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

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

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

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

typ:
    INT       { Int }
  | FLOAT     { Float }
  | BOOL      { Bool }
  | VOID      { Void }
  | STRING    { String }
  | MATRIX typ LBRACKET NUM_LIT RBRACKET LBRACKET NUM_LIT
     RBRACKET   { Matrix($2, $4, $7) }

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

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

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

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

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

expr:
    NUM_LIT { NumLit($1) }
  | STRING_LIT { StringLit($1) }
  | TRUE { BoolLit(true) }
  | FALSE { BoolLit(false) }
  | ID { Id($1) }
  | expr PLUS expr { Binop($1, Add, $3) }
  | expr MINUS expr { Binop($1, Sub, $3) }
  | expr TIMES expr { Binop($1, Mult, $3) }
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr EQ expr { Binop($1, Equal, $3) }
  | expr NEQ expr { Binop($1, Neq, $3) }
  | expr LT expr { Binop($1, Less, $3) }
  | expr LEQ expr { Binop($1, Leq, $3) }
  | expr GT expr { Binop($1, Greater, $3) }
  | expr GEQ expr { Binop($1, Geq, $3) }
  | expr AND expr { Binop($1, And, $3) }
  | expr OR expr { Binop($1, Or, $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr { Unop(Not, $2) }
  | INC expr { Unop(Inc, $2) }
  | DEC expr { Unop(Dec, $2) }
  | expr ASSIGN expr { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }
  | LBRACKET mat_lit RBRACKET { MatrixLit($2) }
  | ID LBRACKET expr RBRACKET LBRACKET expr RBRACKET {
     MatrixAccess($1, $3, $6) }
  | ID COLON ROWS { Rows($1) }
  | ID COLON COLS { Cols($1) }
  | ID COLON TRANSPOSE { Transpose($1) }
  | ID COLON TRACE { Trace($1) }
  | ID COLON SUBMATRIX LBRACKET expr COMMA expr COMMA expr
     COMMA expr RBRACKET { SubMatrix($1, $5, $7, $9, $11) }

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

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

mat_lit:
    LBRACKET lit_list RBRACKET                          { [$2] }
    | mat_lit SEMI LBRACKET lit_list RBRACKET           { $4 ::
      $1 }

lit_list:
    lit                            { [$1] }
    | lit_list COMMA lit           { $3 :: $1 }

lit:
    NUM_LIT                        { $1 }
```

9.3 ast.ml

```
(*
        Jeffrey Monahan         - jm4155
        Christine Pape          - cmp2223
        Montana St. Pierre      - mrs2296
        Shelley Zhong           - sz2699
        Tengyu Zhou             - tz2338
*)
```

```
(* Binary Operators *)
type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq |
   Greater | Geq | And | Or

(* Unary Operators *)
type uop = Neg | Not | Inc | Dec

(* Num *)
type num = IntLit of int | FloatLit of float

(* Data Types *)
type typ = Int | Float | Bool | Void    | String | Matrix of
   typ * num * num

(* Bind *)
type bind = typ * string

(* Expressions *)
type expr =
        | NumLit of num
        | BoolLit of bool
        | StringLit of string
        | MatrixLit of num list list
        | Id of string
        | Binop of expr * op * expr
        | Unop of uop * expr
        | Assign of expr * expr
        | Call of string * expr list
        | Noexpr
        | MatrixAccess of string * expr * expr
        | Rows of string
        | Cols of string
        | Transpose of string
        | SubMatrix of string * expr * expr * expr * expr
        | Trace of string

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

(* Function Declarations *)
type func_decl = {
        typ             : typ;
        fname           : string;
        formals         : bind list;
        locals          : bind list;
        body            : stmt list;
}

(* Program *)
type program = bind list * func_decl list


(* Pretty-printing functions *)
let string_of_num = function
        IntLit(x)         -> string_of_int x
        | FloatLit(x) -> string_of_float x

let string_of_typ = function
        Int                             -> "int"
        | Float                 -> "float"
        | Void                  -> "void"
        | Bool                  -> "bool"
        | String                -> "String"
        | Matrix(t,r,c) -> "matrix(" ^ (string_of_num r) ^
           "," ^ (string_of_num c) ^ ")"

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

let string_of_uop = function
        Not                             -> "not"
        | Inc                   -> "++"
        | Dec                   -> "--"
        | Neg                   -> "-"

let rec string_of_expr = function
        NumLit(i)                                       ->
           string_of_num i
        | BoolLit(b)                            -> if b then
```

```
          "true" else "false"
        | StringLit(s)                          -> "\"" ^
          (String.escaped s) ^ "\""
        | Id(s)                                         -> s
        | Binop(e1, o, e2)                      ->
          (string_of_expr e1) ^ " " ^ (string_of_op o) ^ " "
          ^ (string_of_expr e2)
        | Unop(uop, e)                          ->
          (string_of_uop uop) ^ "(" ^ string_of_expr e ^ ")"
        | Assign(e1, e2)                        ->
          (string_of_expr e1) ^ " = " ^ (string_of_expr e2)
        | Call(f, el)                           -> f ^ "(" ^
          String.concat ", " (List.map string_of_expr el) ^
          ")"
        | Noexpr                                        -> ""
        | MatrixLit(el)                         -> "MatrixLit"
        | MatrixAccess (s, i, j)        -> (s) ^ "[" ^
          (string_of_expr i) ^ "," ^ (string_of_expr j) ^ "]"
        | Rows(s)                                       ->
          (s) ^ ":rows"
        | Cols(s)                                       ->
          (s) ^ ":cols"
        | Transpose(s)                          -> (s) ^
          ":transpose"
        | Trace(s)                     -> (s) ^ ":trace"
        | SubMatrix(s,e1,e2,e3,e4)  -> (s) ^ ":submatrix"

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

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

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

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

9.4 semant.ml

```
(*
        Jeffrey Monahan          - jm4155
        Christine Pape           - cmp2223
        Montana St. Pierre       - mrs2296
        Shelley Zhong            - sz2699
        Tengyu Zhou              - tz2338
*)
open Ast
open Sast

module StringMap = Map.Make(String)


let check (globals, functions) =

        (* Checking duplicates *)
        let report_duplicate exceptf list =
                let rec helper = function
                        n1 :: n2 :: _ when n1 = n2 -> raise
                           (Failure (exceptf n1))
                        | _ :: t -> helper t
                        | [] -> ()
                in helper (List.sort compare list)
        in

        (* Checking void type in binding *)
        let check_not_void exceptf = function
              (Void, n) -> raise (Failure (exceptf n))
            | _ -> ()
```

```
in

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


(**** Checking User-defined Functions. Cannot define
   built-in functions.  ****)
if List.mem "printInt" (List.map (fun fd -> fd.fname)
   functions)
then raise (Failure ("function printInt may not be
   defined")) else ();
if List.mem "printFloat" (List.map (fun fd ->
   fd.fname) functions)
then raise (Failure ("function printFloat may not be
   defined")) else ();
if List.mem "printStr" (List.map (fun fd -> fd.fname)
   functions)
then raise (Failure ("function printStr may not be
   defined")) else ();
if List.mem "printBool" (List.map (fun fd ->
   fd.fname) functions)
then raise (Failure ("function printBool may not be
   defined")) else ();

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

(* Add built-in print functions *)
let built_in_decls = [{
                typ                     = Void;
                fname           = "printInt";
                formals         = [(Int, "i")];
                locals          = [];
                body            = [];
        };
        {
                typ                     = Void;
                fname           = "printFloat";
                formals         = [(Float, "f")];
                locals          = [];
                body            = [];
        };
        {
                typ                         = Void;
                fname               = "printStr";
                formals             = [(String, "s")];
                locals              = [];
                body                = [];
        };
        {
                typ                         = Void;
                fname               = "printBool";
                formals             = [(Bool, "b")];
                locals              = [];
                body                = [];
        }]
in

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

(* Checking if s a declared function *)
let function_decl s =
        try StringMap.find s function_decls
        with Not_found -> raise (Failure
           "Unrecognized function")
in

let _ = function_decl "main" in

(* Several checks on a single function *)
let check_function func =
        (* Check void/duplicate for formals and
           locals *)
        List.iter (check_not_void (fun n -> "illegal
           void formal " ^ n ^ " in " ^ func.fname))
           func.formals;
List.iter (check_not_void (fun n -> "illegal void
   local " ^ n ^ " in " ^ func.fname)) func.locals;
        report_duplicate (fun n -> "duplicate formal
           " ^ n ^ " in " ^ func.fname) (List.map snd
           func.formals);
report_duplicate (fun n -> "duplicate local " ^ n ^ "
   in " ^ func.fname) (List.map snd func.locals);

        (* Check for duplicate variables *)
        ignore(report_duplicate (fun n -> "duplicate
           global variable " ^ n) (List.map snd
```

```
            globals));
        ignore(report_duplicate (fun n -> "duplicate
            formal variable " ^ n) (List.map snd
            func.formals));
        ignore(report_duplicate (fun n -> "duplicate
            local variable " ^ n) (List.map snd
            func.locals));
in


(* Build function symbol table *)
let func_to_symbols func =
        List.fold_left (fun m (t, n) -> StringMap.add
            n t m) StringMap.empty (globals @
            func.formals @ func.locals)
in

let rec type_of_identifier s symbols =
        try StringMap.find s symbols
        with | Not_found -> raise (Failure("Undefined
            ID " ^ s))

and check_eq_type se1 se2 op = function
        (Int, Int) -> SBinop(se1, op, se2, Int)
        | (Float, Float) -> SBinop(se1, op, se2,
            Float)
        | (String, String) -> SBinop(se1, op, se2,
            String)
        | _ -> raise (Failure "Illegal type for
            equality operators")

and check_log_type se1 se2 op = function
                (Bool, Bool) -> SBinop(se1, op, se2,
                    Bool)
                | _ -> raise (Failure "Illegal type
                    for logical operators")

and check_arith_type se1 se2 op = function
                  (Int, Float)
                | (Float, Int)
                | (Float, Float)        ->
                    SBinop(se1, op, se2, Float)
                | (Int, Int)            ->
                    SBinop(se1, op, se2, Int)
                | (String, String)      ->
                       (match op with
                           Add -> SBinop(se1,
                              op, se2, String)
                | _ -> raise(Failure
                  "Invalid operation
                  on String"))
| (Matrix(t1, r1, c1), Matrix(t2, r2,
  c2)) ->
        (match op with
                Add | Sub       ->
                        if t1=t2 &&
                          r1=r2 &&
                          c1=c2 then
                               SBinop(se1,
                                 op,
                                 se2,
                                 Matrix(t1,
                                 r1,
                                 c2))
                        else
                          raise(Failure
                          "Incorrect
                          dimention/type
                          for matrix
                          addition/subtraction")
                | Mult          ->
                        if t1=t2 &&
                          c1 = r2
                          then
                               SBinop(se1,
                                 op,
                                 se2,
                                 Matrix(t1,
                                 r1,
                                 c2))
                        else
                          raise(Failure
                          "Incorrect
                          dimention/type
                          for matrix
                          multiplication")
                | _ ->
                  raise(Failure("Invalid
                  operation on
                  matrix")))
| (Int, Matrix(Int,r,c)) ->
        (match op with
                Mult -> SBinop(se1,
                  op, se2,
                  Matrix(Int, r, c))
                | _ -> raise(Failure
                                  "Invalid operation
                                  between integer
                                  and matrix"))
                | (Float, Matrix(Float,r,c)) ->
                        (match op with
                               Mult -> SBinop(se1,
                                  op, se2,
                                  Matrix(Float, r,
                                  c))
                               | _ ->
                                  raise(Failure("Invalid
                                  operation between
                                  float and
                                  matrix")))
                | _ -> raise (Failure("Invalid type
                   for arithmetic operators"))

and check_is_int symbols e = match e with
        NumLit(IntLit(n))       -> Int
        | Id(s)                        ->
           type_of_identifier s symbols
        | _                            ->
           raise(Failure"Integer required for matrix
           dimension/index")

and lit_to_slit n = match n with
        IntLit(n)     -> SNumLit(SIntLit(n))
        | FloatLit(n) -> SNumLit(SFloatLit(n))

and typ_of_lit n = match n with
        IntLit(n)     -> Int
        | FloatLit(n) -> Float

and sexpr symbols = function
NumLit(IntLit(n)) -> SNumLit(SIntLit(n))
| NumLit(FloatLit(n)) -> SNumLit(SFloatLit(n))
| BoolLit(b) -> SBoolLit(b)
| StringLit(s) -> SStringLit(s)
| Id(s) -> SId(s, type_of_identifier s symbols)
| Binop(e1, op, e2) ->
let se1 = sexpr symbols e1 in
        let se2 = sexpr symbols e2 in
        let t1 = Sast.get_sexpr_type se1 in
        let t2 = Sast.get_sexpr_type se2 in
        (match op with
        Equal | Neq -> check_eq_type se1 se2 op (t1,
           t2)
        | And | Or -> check_log_type se1 se2 op (t1,
                t2)
            | Less | Leq | Greater | Geq when (t1 = Int
                || t1 = Float) && t1 = t2 -> SBinop(se1,
                op, se2, t1)
            | Add | Mult | Sub | Div -> check_arith_type
                se1 se2 op (t1, t2)
            | _ -> raise (Failure "Invalid binary
                operator"))
| Unop(op, e) -> let se = sexpr symbols e in
        let t = Sast.get_sexpr_type se in
        (match op with
                Neg when t = Int -> SUnop(op, se ,t)
            | Neg when t = Float -> SUnop(op, se ,t)
            | Inc when t = Int -> SUnop(op, se ,t)
            | Inc when t = Float -> SUnop(op, se ,t)
            | Dec when t = Int -> SUnop(op, se ,t)
            | Dec when t = Float -> SUnop(op, se ,t)
            | Not when t = Bool -> SUnop(op, se, t)
            | _ -> raise(Failure "Invalid datatype for
                unop")

| Assign(s,e) -> let se1 = sexpr symbols s in
        let se2 = sexpr symbols e in
        let t1 = Sast.get_sexpr_type se1 in
        let t2 = Sast.get_sexpr_type se2 in
        (if t1 = t2
        then SAssign(se1, se2, t1)
        else raise(Failure "Mismatched assignment
                type"))
| Call(fname, actuals) -> let fd = function_decl
    fname in
        if List.length actuals != List.length
                fd.formals
then raise (Failure "Incorrect number of arguments")
        else SCall(fname, List.map (sexpr symbols)
                actuals,fd.typ)
| Noexpr -> SNoexpr
| MatrixAccess(s, dim1, dim2) ->
ignore(check_is_int symbols dim1);
        ignore(check_is_int symbols dim2);
        let typ = type_of_identifier s symbols  in
        (match typ with
        Matrix(t,r,c) -> SMatrixAccess(s, sexpr
                symbols dim1, sexpr symbols dim2, t)
        | _ -> raise(Failure "Cannot operate on
                nonmatrix")     )
| MatrixLit(mlist) -> let smlist = (List.map (fun l
    -> (List.map lit_to_slit l)) mlist) in
```

```
          let entry = List.hd (List.hd mlist) in
          let row_size = List.length (List.hd mlist) in
          ignore(List.iter (fun nl -> if (List.length
             nl = row_size) then () else raise(Failure
             "Rows of matrix must have the same
             size")) mlist);
          let entry_typ = typ_of_lit entry in
          ignore(List.iter (fun nl -> List.iter (fun n
             ->
          (let typ = typ_of_lit n in
          if (typ = entry_typ)
          then ()
          else raise(Failure "More than one datatype
             in a matrix" ))) nl) mlist);
          SMatrixLit(smlist, entry_typ)
| Rows(s) -> (match type_of_identifier s symbols with
                        Matrix(_, r, _) ->
                        (match r with IntLit(n) ->
                           SRows(n)
                        | _ -> raise(Failure
                           "Integer required for
                           matrix dimension"))
                        | _ -> raise(Failure"Cannot
                           operate on nonmatrix"))
| Cols(s) -> (match type_of_identifier s symbols with
                        Matrix(_, _, c) ->
                        (match c with IntLit(n) ->
                           SCols(n)
                        | _ -> raise(Failure"Integer
                           required for matrix
                           dimension"))
                        | _ -> raise(Failure"Cannot
                           operate on nonmatrix"))
| Transpose(s) -> (match type_of_identifier s symbols
   with
                       Matrix(t, r, c) ->
                          STranspose(s, Matrix(t, c,
                          r))
                       | _ -> raise(Failure"Cannot
                          operate on nonmatrix"))
| Trace(s) -> (match type_of_identifier s symbols with
                       Matrix(t, r, c) -> (if r = c
                          then STrace(s,
                          Matrix(t,r,c))
                       else raise(Failure "Trace
                          only operates on square
                          matrices"))
                        | _ -> raise(Failure"Cannot
                            operator on nonmatrix"))
| SubMatrix(s,e1,e2,e3,e4) -> ignore(check_is_int
   symbols e1);
                          ignore(check_is_int symbols
                             e2);
                          ignore(check_is_int symbols
                             e3);
                          ignore(check_is_int symbols
                             e4);
                          let se1 = sexpr symbols e1 in
                          let se2 = sexpr symbols e2 in
                          let se3 = sexpr symbols e3 in
                          let se4 = sexpr symbols e4 in
                          let typ = type_of_identifier
                             s symbols in
                          (match typ with
                          Matrix(t,r,c) ->
                             SSubMatrix(s, se1, se2,
                             se3, se4, Matrix(t,r,c))
                          | _ -> raise(Failure"Cannot
                             operator on nonmatrix"))
in

let rec sstmt symbols = function
        Block(stmt_list)                ->
           SBlock(stmt_list_to_sstmt_list symbols
           stmt_list)
        | Expr(e)                               ->
           SExpr(sexpr symbols e)
        | If(e, s1, s2)                 -> SIf((sexpr
           symbols e), (sstmt symbols s1), (sstmt
           symbols s2))
        | For(e1, e2, e3, s)     -> SFor((sexpr
           symbols e1), (sexpr symbols e2), (sexpr
           symbols e3), (sstmt symbols s))
        | While(e, s)                   ->
           SWhile((sexpr symbols e), (sstmt symbols
           s))
        | Return(e)                             ->
           SReturn(sexpr symbols e)

and stmt_list_to_sstmt_list symbols stmt_list =
   List.map (sstmt symbols) stmt_list
in

let func_to_sfunc func =
        {
                sfname                  = func.fname;
```

```
                        styp                    = func.typ;
                        sformals                =
                           func.formals;
                        slocals                 = func.locals;
                        sbody                   =
                           (stmt_list_to_sstmt_list
                           (func_to_symbols func) func.body);
                }
        in

(* check functions *)
ignore(List.iter check_function functions);

(* convert func to sfunc *)
let sfuncs = List.map func_to_sfunc functions
in      (globals, sfuncs)
```

9.5 sast.ml

```
(*
        Jeffrey Monahan         - jm4155
        Christine Pape          - cmp2223
        Montana St. Pierre      - mrs2296
        Shelley Zhong           - sz2699
        Tengyu Zhou             - tz2338
*)
open Ast

type snum =
                SIntLit of int
        |       SFloatLit of float

type sexpr =
          SNumLit of snum
        | SBoolLit of bool
        | SStringLit of string
        | SMatrixLit of sexpr list list * typ
        | SId of string * typ
        | SBinop of sexpr * op * sexpr * typ
        | SUnop of uop * sexpr * typ
        | SAssign of sexpr * sexpr * typ
        | SCall of string * sexpr list * typ
        | SNoexpr
        | SMatrixAccess of string * sexpr * sexpr * typ
        | SRows of int
        | SCols of int
        | STranspose of string * typ
        | SSubMatrix of string * sexpr * sexpr * sexpr *
           sexpr * typ
        | STrace of string * typ

let get_sexpr_type sexpr = match sexpr with
        SNumLit(SIntLit(_)) -> Int
        | SNumLit(SFloatLit(_)) -> Float
        | SBoolLit(_) -> Bool
        | SStringLit(_) -> String
        | SNoexpr -> Void
        | SRows(r) -> Int
        | SCols(c) -> Int
        | STranspose(_,t) -> t
        | SId(_, t) -> t
        | SBinop(_, _, _, t) -> t
        | SAssign(_, _, t) -> t
        | SCall(_, _, t) -> t
        | SUnop(_, _, t) -> t
        | SMatrixAccess(_, _, _, t) -> t
        | SMatrixLit(smlist, t) ->
              let c = List.length (List.hd smlist) in
              let r = List.length smlist in
              (match t with
                      Int -> Matrix(Int, IntLit(r),
                         IntLit(c))
                      | Float -> Matrix(Float, IntLit(r),
                         IntLit(c))
                      | _ ->
                         raise(Failure"UnsupportedMatrixType"))
        | SSubMatrix (_,_,_,_,_,t) -> t
        | STrace(_,t) -> t

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

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

```
type sprogram = bind list * sfunc_decl list
```

9.6 codegen.ml

```
(*
        Jeffrey Monahan          - jm4155
        Christine Pape           - cmp2223
        Montana St. Pierre       - mrs2296
        Shelley Zhong            - sz2699
        Tengyu Zhou              - tz2338
*)
open Llvm
open Ast
open Sast
module L = Llvm
module A = Ast
module S = Sast

module StringMap = Map.Make(String)

let translate (globals, functions) =

    let context = L.global_context() in
    let the_module = L.create_module context "M2"
    and i32_t     = L.i32_type context
    and i8_t      = L.i8_type context
    and i1_t      = L.i1_type context
    and void_t    = L.void_type context
    and float_t   = L.double_type context
    and array_t   = L.array_type
    and pointer_t = L.pointer_type
    in

    let ltype_of_typ = function
          A.Int     -> i32_t
        | A.Float   -> float_t
        | A.Bool    -> i1_t
        | A.Void    -> void_t
        | A.String  -> pointer_t i8_t
        | A.Matrix(t, r, c) ->
            let rows = match r with IntLit(s) -> s
                                  | _ ->
                                     raise(Failure"Integer
                                     required for matrix
                                     dimension") in
            let cols = match c with IntLit(s) -> s
                                  | _ ->
```

```
                                raise(Failure"Integer
                                required for matrix
                                dimension") in
        (match t with
            A.Int      -> array_t (array_t i32_t cols)
               rows
            | A.Float  -> array_t (array_t float_t cols)
               rows
            | _ -> raise(Failure"Invalid datatype for
               matrix"))
in

let global_vars =
    let global_var m (t,n) =
    let init = L.const_int (ltype_of_typ t) 0
    in  StringMap.add n (L.define_global n init
       the_module) m in
    List.fold_left global_var StringMap.empty globals in

let printf_t = L.var_arg_function_type i32_t [|
   L.pointer_type i8_t |]
in

let printf_func = L.declare_function "printf" printf_t
   the_module
in

let function_decls =
    let function_decl m fdecl =
        let name = fdecl.S.sfname
        and formal_types = Array.of_list
            (List.map (function A.(t,s) -> ltype_of_typ
               t) fdecl.S.sformals) in
        let ftype =
            L.function_type (ltype_of_typ fdecl.S.styp)
               formal_types in
            StringMap.add name (L.define_function name
               ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions
in

let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.S.sfname
       function_decls in
    let builder = L.builder_at_end context (L.entry_block
       the_function) in

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

    let local_vars =
        let add_formal m (t, n) p = L.set_value_name n p;
        let local = L.build_alloca (ltype_of_typ t) n
           builder in
        ignore (L.build_store p local builder);
        StringMap.add n local m
    in

    let add_local m (t, n) =
        let local_var = L.build_alloca (ltype_of_typ t) n
           builder
        in StringMap.add n local_var m
    in

    let formals = List.fold_left2 add_formal
       StringMap.empty
        (List.map (function A.(t,n) -> (t,n))
           fdecl.S.sformals) (Array.to_list (L.params
           the_function)) in
        List.fold_left add_local formals (List.map
           (function A.(t,n) -> (t,n)) fdecl.S.slocals)
    in

    let lookup n = try StringMap.find n local_vars
        with Not_found -> StringMap.find n global_vars
    in

    let build_matrix_access s rowIndex colIndex builder
       assigned =
        let ptr = L.build_gep (lookup s) [|L.const_int
           i32_t 0; rowIndex; colIndex|] s builder in
        if assigned then ptr else L.build_load ptr s
           builder
    in

    let rec expr builder = function
        S.SNumLit(SIntLit(i))      -> L.const_int i32_t i
        | S.SNumLit(SFloatLit(f))  -> L.const_float
           float_t f
        | S.SBoolLit b             -> L.const_int i1_t
           (if b then 1 else 0)
```

```
| S.SStringLit s           ->
   L.build_global_stringptr s "tmp" builder
| S.SId (s, d)             -> L.build_load
   (lookup s) s builder
| S.SBinop (se1, op, se2, d)  ->
    let type1 = Sast.get_sexpr_type se1 in
    let type2 = Sast.get_sexpr_type se2 in
    let se1' = expr builder se1
    and se2' = expr builder se2 in

    let int_binop op se1' se2' =
        (match op with
            A.Add     -> L.build_add
            | A.Sub     -> L.build_sub
            | A.Mult    -> L.build_mul
            | A.Div     -> L.build_sdiv
            | A.Equal   -> L.build_icmp L.Icmp.Eq
            | A.Neq     -> L.build_icmp L.Icmp.Ne
            | A.Less    -> L.build_icmp L.Icmp.Slt
            | A.Leq     -> L.build_icmp L.Icmp.Sle
            | A.Greater -> L.build_icmp L.Icmp.Sgt
            | A.Geq     -> L.build_icmp L.Icmp.Sge
            | A.And     -> L.build_and
            | A.Or      -> L.build_or
            | _         -> raise(Failure "Invalid
               binary operator for int")) se1'
               se2' "tmp" builder
    in

    let float_binop op se1' se2' =
        (match op with
            A.Add       -> L.build_fadd
            | A.Sub       -> L.build_fsub
            | A.Mult      -> L.build_fmul
            | A.Div       -> L.build_fdiv
            | A.Equal     -> L.build_fcmp
               L.Fcmp.Oeq
            | A.Neq       -> L.build_fcmp
               L.Fcmp.One
            | A.Less      -> L.build_fcmp
               L.Fcmp.Olt
            | A.Leq       -> L.build_fcmp
               L.Fcmp.Ole
            | A.Greater   -> L.build_fcmp
               L.Fcmp.Ogt
            | A.Geq       -> L.build_fcmp
               L.Fcmp.Oge
            | _           ->
```

```
          raise(Failure"Invalid binary
          operator for float")) se1' se2'
          "tmp" builder
in

let bool_binop op se1' se2' =
    (match op with
        | A.And   -> L.build_and
        | A.Or    -> L.build_or
        | _       -> raise(Failure"Invalid
          boolean operator")) se1' se2'
          "tmp" builder
in

let matrix_binop mtype rDimension cDimension
   op se1 se2 =
    let lhs_str = (match se1 with SId(s,_) ->
       s | _ -> "") in
    let rhs_str = (match se2 with SId(s,_) ->
       s | _ -> "") in
    let operator_type = match mtype with
     "int" -> i32_t | "float" -> float_t | _
        -> i32_t
    in
    let operator_type2 = match mtype with
     "int" -> L.const_int | "float" ->
        L.const_int | _ -> L.const_int
    in
    let buildtype = match mtype with
     "int" -> (match op with A.Add ->
        L.build_add | A.Sub -> L.build_sub |
        A.Mult -> L.build_mul |  _ ->
        raise(Failure "Invalid Matrix Binop"))
    | "float" -> (match op with A.Add ->
       L.build_fadd | A.Sub -> L.build_fsub |
       A.Mult -> L.build_fmul | _ ->
       raise(Failure "Invalid Matrix Binop"))
    | _ -> L.build_add
    in
    let buildtype2 = match mtype with
     "int" -> L.build_add | "float" ->
        L.build_fadd | _ -> L.build_add
    in
    (match op with
    A.Add  | A.Sub ->
    let tmp_mat = L.build_alloca (array_t
       (array_t operator_type cDimension)
       rDimension) "tmpmat" builder in
```

```
for i=0 to (rDimension-1) do
for j=0 to (cDimension-1) do
let m1 = build_matrix_access lhs_str
   (L.const_int i32_t i) (L.const_int
   i32_t j) builder false in
let m2 = build_matrix_access rhs_str
   (L.const_int i32_t i) (L.const_int
   i32_t j) builder false in
let result = buildtype m1 m2 "tmp"
   builder in
let ld = L.build_gep tmp_mat [|
   L.const_int i32_t 0; L.const_int i32_t
   i; L.const_int i32_t j |] "tmpmat"
   builder in
ignore(build_store result ld builder);
done
done;
L.build_load (L.build_gep tmp_mat [|
   L.const_int i32_t 0 |] "tmpmat"
   builder) "tmpmat" builder
| A.Mult ->
let first_typ = Sast.get_sexpr_type se1 in
let tmp_mat = L.build_alloca (array_t
   (array_t operator_type cDimension)
   rDimension) "tmpmat" builder in
(match first_typ with
Int| Float ->
for i=0 to (rDimension-1) do
for j=0 to (cDimension-1) do
let m2 = build_matrix_access rhs_str
   (L.const_int i32_t i) (L.const_int
   i32_t j) builder false in
let result = buildtype (build_load
   (lookup lhs_str) "tmp" builder) m2
   "tmp" builder in
let ld = L.build_gep tmp_mat [|
   L.const_int i32_t 0; L.const_int i32_t
   i; L.const_int i32_t j |] "tmpmat"
   builder in
ignore(build_store result ld builder);
done
done;
L.build_load (L.build_gep tmp_mat [|
   L.const_int i32_t 0 |] "tmpmat"
   builder) "tmpmat" builder

| Matrix(Int,r,c) ->
let tmp_product = L.build_alloca
```

```
   operator_type "tmpproduct" builder in
let c_i = (match c with IntLit(n) -> n |
   _ -> -1) in
ignore(L.build_store (operator_type2
   operator_type 0) tmp_product builder);
for i=0 to (rDimension-1) do
for j=0 to (cDimension-1) do
ignore(L.build_store (operator_type2
   operator_type 0) tmp_product builder);
for k=0 to (c_i-1) do
let m1 = build_matrix_access lhs_str
   (L.const_int i32_t i) (L.const_int
   i32_t k) builder false in
let m2 = build_matrix_access rhs_str
   (L.const_int i32_t k) (L.const_int
   i32_t j) builder false in
let result = buildtype m1 m2 "tmp"
   builder in
ignore(L.build_store (buildtype2 result
   (L.build_load tmp_product "addtmp"
   builder) "tmp" builder) tmp_product
   builder);
done;
let ld = L.build_gep tmp_mat [|
   L.const_int i32_t 0; L.const_int i32_t
   i; L.const_int i32_t j |] "tmpmat"
   builder in
ignore(build_store (L.build_load
   tmp_product "restmp" builder) ld
   builder);
done
done;
L.build_load (L.build_gep tmp_mat [|
   L.const_int i32_t 0 |] "tmpmat"
   builder) "tmpmat" builder
| Matrix(Float,r,c) ->
let tmp_product = L.build_alloca float_t
   "tmpproduct" builder in
let c_i = (match c with IntLit(n) -> n |
   _ -> -1) in
ignore(L.build_store (L.const_float
   float_t 0.0) tmp_product builder);
for i=0 to (rDimension-1) do
for j=0 to (cDimension-1) do
ignore(L.build_store (L.const_float
   float_t 0.0) tmp_product builder);
for k=0 to (c_i-1) do
let m1 = build_matrix_access lhs_str
```

```
            (L.const_int i32_t i) (L.const_int
            i32_t k) builder false in
         let m2 = build_matrix_access rhs_str
            (L.const_int i32_t k) (L.const_int
            i32_t j) builder false in
         let result = L.build_fmul m1 m2 "tmp"
            builder in
         ignore(L.build_store (L.build_fadd result
            (L.build_load tmp_product "addtmp"
            builder) "tmp" builder) tmp_product
            builder);
         done;
         let ld = L.build_gep tmp_mat [|
            L.const_int i32_t 0; L.const_int i32_t
            i; L.const_int i32_t j |] "tmpmat"
            builder in
         ignore(build_store (L.build_load
            tmp_product "restmp" builder) ld
            builder);
         done
         done;
         L.build_load (L.build_gep tmp_mat [|
            L.const_int i32_t 0 |] "tmpmat"
            builder) "tmpmat" builder
         | _ -> L.const_int i32_t 0)

| _ -> raise(Failure "Invalid Matrix Binop"))
in

let build_binop operand1 operand2 type1 type2
   =
      match (type1, type2) with
          (Int, Int) ->  int_binop op operand1
             operand2
          | (Float, Float) ->  float_binop op
             operand1 operand2
          | (Bool, Bool) ->  bool_binop op
             operand1 operand2
          | (Int, Matrix(Int,r1,c2)) -> let
             rDimension = (match r1 with
             IntLit(n) -> n | _ -> -1)
           and cDimension = (match c2 with
              IntLit(n) -> n | _ -> -1) in
           matrix_binop "int" rDimension
              cDimension op se1 se2
          | (Float, Matrix(Float,r1,c2)) ->
              let rDimension = (match r1 with
                 IntLit(n) -> n | _ -> -1)
```

```
                and cDimension = (match c2 with
                   IntLit(n) -> n | _ -> -1) in
                matrix_binop "float" rDimension
                   cDimension op se1 se2
            | (Matrix(Int,r1,c1),
               Matrix(Int,r2,c2)) ->
                let rDimension = (match r1 with
                   IntLit(n) -> n | _ -> -1)
                and cDimension = (match c2 with
                   IntLit(n) -> n | _ -> -1) in
                matrix_binop "int" rDimension
                   cDimension op se1 se2
            | (Matrix(Float,r1,c1),
               Matrix(Float,r2,c2)) ->
                let rDimension = (match r1 with
                   IntLit(n) -> n | _ -> -1)
                and cDimension = (match c2 with
                   IntLit(n) -> n | _ -> -1) in
                matrix_binop "float" rDimension
                   cDimension op se1 se2
            | _ -> raise(Failure"Cannot build
               binop")
    in
    build_binop se1' se2' type1 type2

| S.SUnop(op, e, t) ->
    let e' = expr builder e in
    (match t with
    Int -> (match op with
            A.Neg -> L.build_neg e' "tmp" builder
            | A.Inc -> L.build_store (L.build_add
               e' (L.const_int i32_t 1) "tmp"
               builder) (lookup (match e with
               S.SId(s, d) -> s |
               _->raise(Failure"IncMustBeCalledOnID")))
               builder
            | A.Dec -> L.build_store (L.build_sub
               e' (L.const_int i32_t 1) "tmp"
               builder) (lookup (match e with
               S.SId(s, d) -> s |
               _->raise(Failure"DecMustBeCalledOnID")))
               builder
            | _ -> raise(Failure"IllegalIntUnop"))
    | Float -> (match op with
            A.Neg -> L.build_fneg e' "tmp" builder
            | _ ->
               raise(Failure"IllegalFloatUnop"))
    | Bool -> (match op with
          A.Not -> L.build_not e' "tmp" builder
          | _ ->
             raise(Failure"IllegalBoolUnop"))
    | _ -> (raise(Failure"InvalidUnopType")))
| S.SAssign (se1, se2, d) ->
    let se1' =
        (match se1 with
            S.SId(s,_) -> (lookup s)
            | S.SMatrixAccess(s, i1, j1, d) ->
                let i = expr builder i1 and j =
                   expr builder j1 in
                    build_matrix_access s i j
                       builder true
            | _ ->
               raise(Failure"AssignLHSMustBeAssignable"))
    and se2' = expr builder se2 in
    ignore (L.build_store se2' se1' builder); se2'
| S.SCall ("printStr", [e], _) ->
    L.build_call printf_func [| string_format_str
       ; (expr builder e) |] "printf" builder
| S.SCall ("printInt", [e], _) ->
    L.build_call printf_func [| int_format_str ;
       (expr builder e) |] "printf" builder
| S.SCall ("printBool", [e], _) ->
    L.build_call printf_func [| int_format_str ;
       (expr builder e) |] "printf" builder
| S.SCall ("printFloat", [e], _) ->
    L.build_call printf_func [| float_format_str
       ; (expr builder e) |] "printf" builder
| S.SCall (f, act, _) ->
    let (fdef, fdecl) = StringMap.find f
       function_decls in
    let actuals = List.rev (List.map (expr
       builder) (List.rev act)) in
    let result =
        (match fdecl.S.styp with
            A.Void -> ""
            | _ -> f ^ "_result") in
    L.build_call fdef (Array.of_list actuals)
       result builder
| S.SNoexpr                -> L.const_int i32_t 0
| S.SMatrixAccess (s, se1, se2, _) ->
    let i = expr builder se1 and j = expr builder
       se2 in
        (build_matrix_access s i j builder false)
| S.SMatrixLit (smlist, t) ->
    let numtype = match t with
    A.Float -> float_t
```

```
        | A.Int -> i32_t
        | _ -> i32_t
        in
        let flipped = List.map List.rev smlist in
        let lists = List.map (List.map (expr
           builder)) flipped in
        let listArray = List.map Array.of_list lists
           in
        let listArray2 = List.rev (List.map
           (L.const_array numtype) listArray) in
        let arrayArray  = Array.of_list listArray2 in
        L.const_array (array_t numtype (List.length
           (List.hd smlist))) arrayArray
    | S.SRows(r) -> L.const_int i32_t r
    | S.SCols(c) -> L.const_int i32_t c
    | S.STranspose(s,t) ->
        let alloctype = match t with
            Matrix(Int, c, r) -> i32_t |
               Matrix(Float, c, r) -> float_t| _ ->
               i32_t in
        (match t with
            Matrix(Int, c, r) | Matrix(Float, c, r) ->
                let r_tr = (match c with IntLit(n) ->
                   n | _ -> -1) in
                let c_tr = (match r with IntLit(n) ->
                   n | _ -> -1) in
                let tmp_tr = L.build_alloca (array_t
                   (array_t alloctype c_tr) r_tr)
                   "tmpmat" builder in
                for i=0 to (r_tr-1) do
                    for j=0 to (c_tr-1) do
                        let mtr = build_matrix_access
                           s (L.const_int i32_t i)
                           (L.const_int i32_t j)
                           builder false in
                        let ld = L.build_gep tmp_tr
                           [| L.const_int i32_t 0;
                           L.const_int i32_t j;
                           L.const_int i32_t i |]
                           "tmpmat" builder in
                        ignore(build_store mtr ld
                           builder);
                    done
                done;
                L.build_load (L.build_gep tmp_tr [|
                   L.const_int i32_t 0 |] "tmpmat"
                   builder) "tmpmat" builder
          | _ -> const_int i32_t 0)
```

```
| S.SSubMatrix(s, r1, r2, c1, c2, t) ->
    (let alloctype = match t with
        Matrix(Int, c, r) -> i32_t |
           Matrix(Float, c, r) -> float_t| _ ->
           i32_t in
        match t with Matrix(Int, c, r)|
           Matrix(Float, c, r) ->
            let r1' = (match r1 with
               SNumLit(SIntLit(n)) -> n | _ ->
               -1) in
            let r2' = (match r2 with
               SNumLit(SIntLit(n)) -> n | _ ->
               -1) in
            let c1' = (match c1 with
               SNumLit(SIntLit(n)) -> n | _ ->
               -1) in
            let c2' = (match c2 with
               SNumLit(SIntLit(n)) -> n | _ ->
               -1) in
            let tmp_tr = L.build_alloca (array_t
               (array_t alloctype (c2' - c1' +
               1)) (r2' - r1' + 1)) "tmptr"
               builder in
            for i=r1' to (r2') do
                for j=c1' to (c2') do
                    let mtr = build_matrix_access
                       s (L.const_int i32_t i)
                       (L.const_int i32_t j)
                       builder false in
                    let ld = L.build_gep tmp_tr
                       [| L.const_int i32_t 0;
                       L.const_int i32_t (i -
                       r1'); L.const_int i32_t (j
                       - c1') |] "tmpmat" builder
                       in
                    ignore(build_store mtr ld
                       builder);
                done
            done;
            L.build_load (L.build_gep tmp_tr [|
               L.const_int i32_t 0 |] "tmptr"
               builder) "tmptr" builder
        | _ -> const_int i32_t 0)
| S.STrace(s, t) ->
    (let alloctype = match t with
        Matrix(Int, c, r) -> i32_t
        | Matrix(Float, c, r) -> float_t
        | _ -> i32_t
        in
        let buildtype = match t with
            Matrix(Int, c, r) -> L.build_add
            | Matrix(Float, c, r) -> L.build_fadd
            | _ -> L.build_add
        in
        let initial_val = match t with
            Matrix(Int, c, r) -> L.const_int i32_t 0
            | Matrix(Float, c, r) -> L.const_float
               float_t 0.0
            | _ -> L.const_int i32_t 0
        in
        match t with
            Matrix(Int, c, r)
            | Matrix(Float, c, r) ->
                let c_tr = (match c with IntLit(n) ->
                   n | _ -> -1) in
                let tmp_sum = L.build_alloca
                   alloctype "tmpsum" builder in
                ignore(L.build_store initial_val
                   tmp_sum builder);
                for i=0 to (c_tr-1) do
                        let result =
                           build_matrix_access s
                           (L.const_int i32_t i)
                           (L.const_int i32_t i)
                           builder false in
                        ignore(L.build_store
                           (buildtype result
                           (L.build_load tmp_sum
                           "addtmp" builder) "tmp"
                           builder) tmp_sum builder);
                done;
                L.build_load tmp_sum "restmp" builder
            | _ -> raise(Failure "Cannot calculate
               trace!"))

in


let add_terminal builder f =
    match L.block_terminator (L.insertion_block
       builder) with
        Some _ -> ()
        | None -> ignore (f builder)
in

let rec stmt builder = function
```

```
    S.SBlock sl -> List.fold_left stmt builder sl
    | S.SExpr e -> ignore (expr builder e); builder
    | S.SReturn e ->
       ignore(match fdecl.S.styp with
           A.Void  -> L.build_ret_void builder
           | _                  -> L.build_ret (expr
              builder e) builder); builder
    | S.SIf (predicate, then_stmt, else_stmt) ->
       let bool_val = expr builder predicate in
       let merge_bb = L.append_block context
           "merge" the_function in
       let then_bb = L.append_block context
           "then" the_function in
       add_terminal
           (stmt (L.builder_at_end context then_bb)
              then_stmt)
           (L.build_br merge_bb);
       let else_bb = L.append_block context
           "else" the_function in
       add_terminal
           (stmt (L.builder_at_end context else_bb)
              else_stmt)
           (L.build_br merge_bb);
       ignore (L.build_cond_br bool_val then_bb
          else_bb builder);
       L.builder_at_end context merge_bb
    | S.SWhile (predicate, body) ->
       let pred_bb = L.append_block context
           "while" the_function in
       ignore (L.build_br pred_bb builder);
       let body_bb = L.append_block context
           "while_body" the_function in
       add_terminal (stmt (L.builder_at_end context
          body_bb) body)
       (L.build_br pred_bb);
       let pred_builder = L.builder_at_end context
          pred_bb in
       let bool_val = expr pred_builder predicate in
       let merge_bb = L.append_block context
           "merge" the_function in
       ignore (L.build_cond_br bool_val body_bb
          merge_bb pred_builder);
       L.builder_at_end context merge_bb
    | S.SFor (op1, op2, e3, body) -> stmt builder
       (S.SBlock [S.SExpr op1 ;
           S.SWhile (op2, S.SBlock [body ;
               S.SExpr e3]) ])
in
```

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

        add_terminal builder (match fdecl.S.styp with
            A.Void -> L.build_ret_void;
            | t -> L.build_ret (L.const_int (ltype_of_typ t)
               0))
    in

    List.iter build_function_body functions;
    the_module
```

9.7 m2.ml

```
(*
        Jeffrey Monahan         - jm4155
        Christine Pape          - cmp2223
        Montana St. Pierre      - mrs2296
        Shelley Zhong           - sz2699
        Tengyu Zhou             - tz2338
*)
open Scanner
open Parser
open Ast
open Codegen
open Semant
open Llvm

type action = Ast | LLVM_IR | Compile

let _ =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the
       generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./m2.native [-a|-l|-c] [file.mc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in
     filename) usage_msg;
  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in
  let sast = Semant.check ast in
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule
     (Codegen.translate sast))
  | Compile -> let m = Codegen.translate sast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

9.8 Makefile

```
MAKE = ./scripts/build.sh
CLEAN = ./scripts/clean.sh

default:
	@$(MAKE)

clean:
	@$(CLEAN)
```

9.9 build.sh

```
#!/bin/bash

cp ./src/scanner.mll ./scanner.mll
cp ./src/parser.mly ./parser.mly
cp ./src/ast.ml ./ast.ml
cp ./src/sast.ml ./sast.ml
cp ./src/semant.ml ./semant.ml
cp ./src/codegen.ml ./codegen.ml
cp ./src/m2.ml ./m2.ml

ocamlbuild -j 0 -r -use-ocamlfind -pkgs
   str,llvm,llvm.analysis,llvm.bitwriter,llvm.bitreader,llvm.linker,
llvm.target m2.native
```

9.10 clean.sh

```
#!/bin/bash

rm -rf _build build.log m2.native *.cmo *.cmi *.ml* *.ll
```

9.11 Test1

9.11.1 Pass tests

test-add1.m2

```
int add(int x, int y)
{
  return x + y;
}
int main()
{
  printInt( add(17, 25) );
  return 0;
}
---------------
42
```

test-arith1.m2

```
int main()
{
  printInt(39 + 3);
  return 0;
}
---------------
42
```

test-arith2.m2

```
int main()
{
  printInt(1 + 2 * 3 + 4);
  return 0;
}
---------------
11
```

test-arith3.m2

```
int foo(int a)
{
  return a;
}
int main()
{
  int a;
  a = 42;
  a = a + 5;
  printInt(a);
  return 0;
}
---------------
47
```

test-arithbinops.m2

```
int main() {
    int i;
    int j;
    int k;
    float f;
    float g;
    float h;
    i = 2;
    j = 4;
    f = 2.2;
    g = 3.3;
    k = i+j;
    printInt(i);
    printInt(j);
    printInt(k);
    printFloat(f);
    printFloat(g);
    printFloat(h);
    printInt(k);
    return 0;
}
---------------
2 4 6 2.200000 3.300000 0.000000 6
```

test-fib.m2

```
int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}
int main()
{
  printInt(fib(0));
  printInt(fib(1));
  printInt(fib(2));
  printInt(fib(3));
  printInt(fib(4));
  printInt(fib(5));
  return 0;
}
---------------
1 1 2 3 5 8
```

test-for1.m2

```
int main()
```

```
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    printInt(i);
  }
  printInt(42);
  return 0;
}
---------------
0 1 2 3 4 42
```

test-for2.m2

```
int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    printInt(i);
    i = i + 1;
  }
  printInt(42);
  return 0;
}
---------------
0 1 2 3 4 42
```

test-func1.m2

```
int add(int a, int b)
{
  return a + b;
}
int main()
{
  int a;
  a = add(39, 3);
  printInt(a);
  return 0;
}
---------------
42
```

test-func2.m2

```
int fun(int x, int y)
{
  return 0;
}
int main()
{
  int i;
  i = 1;

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

  printInt(i);
  return 0;
}
---------------
2
```

test-func3.m2

```
void printem(int a, int b, int c, int d)
{
  printInt(a);
  printInt(b);
  printInt(c);
  printInt(d);
}
int main()
{
  printem(42,17,192,8);
  return 0;
}
---------------
42 17 192 8
```

test-func4.m2

```
int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}
int main()
{
  int d;
  d = add(52, 10);
  printInt(d);
  return 0;
}
---------------
62
```

test-func5.m2

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

test-func6.m2

```
void foo() {}

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

int main()
{
  printInt(bar(17, false, 25));
  return 0;
}
---------------
42
```

test-func7.m2

```
int a;

void foo(int c)
{
  a = c + 42;
}
int main()
{
  foo(73);
  printInt(a);
  return 0;
}
---------------
115
```

test-func8.m2

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

```
int main()
{
  foo(40);
  return 0;
}
---------------
43
```

test-gcd.m2

```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}
int main()
{
  printInt(gcd(2,14));
  printInt(gcd(3,15));
  printInt(gcd(99,121));
  return 0;
}
---------------
2 3 11
```

test-gcd2.m2

```
int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}
int main()
{
  printInt(gcd(14,21));
  printInt(gcd(8,36));
  printInt(gcd(99,121));
  return 0;
}
---------------
7 4 11
```

test-global1.m2

```
int a;
int b;

void printa()
{
  printInt(a);
}
void printb()
{
  printInt(b);
}
void incab()
{
  a = a + 1;
  b = b + 1;
}
int main()
{
  a = 42;
  b = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
  return 0;
}
---------------
42      21      43      22
```

test-global2.m2

```
bool i;

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

  i = 42;
  printInt(i + i);
  return 0;
}
---------------
84
```

test-global3.m2

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

```
int j;

int main()
{
  i = 42;
  j = 10;
  printInt(i + j);
  return 0;
}
---------------
52
```

test-hello.m2

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

test-if1.m2

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

test-if2.m2

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

test-if3.m2

```
int main()
```

```
{
  if (false) printInt(42);
  printInt(17);
  return 0;
}
---------------
17
```

test-if4.m2

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

test-if5.m2

```
int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}
int main()
{
 printInt(cond(true));
 printInt(cond(false));
 return 0;
}
---------------
42 17
```

test-local1.m2

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

  i = 42;
  printInt(i + i);
}
```

```
int main()
{
  foo(true);
  return 0;
}
---------------
84
```

test-local2.m2

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

  c = a;

  return c + 10;
}

int main() {
 printInt(foo(37, false));
 return 0;
}
---------------
47
```

test-ops1.m2

```
int main()
{
  printInt(1 + 2);
  printInt(1 - 2);
  printInt(1 * 2);
  printInt(100 / 2);
  printInt(99);
  printBool(1 == 2);
  printBool(1 == 1);
  printInt(99);
  printBool(1 != 2);
  printBool(1 != 1);
  printInt(99);
  printBool(1 < 2);
  printBool(2 < 1);
  printInt(99);
  printBool(1 <= 2);
  printBool(1 <= 1);
  printBool(2 <= 1);
  printInt(99);
  printBool(1 > 2);
  printBool(2 > 1);
  printInt(99);
  printBool(1 >= 2);
  printBool(1 >= 1);
  printBool(2 >= 1);
  return 0;
}
---------------
3 -1 2 50 99 0 1 99 1 0 99 1 0 99 1 1 0 99 0 1 99 0 1 1
```

test-print.m2

```
int main(){
        printStr("my");
        printInt(5);
        return 0;
}
---------------
my
5
```

test-var1.m2

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

test-var2.m2

```
int a;

void foo(int c)
{
  a = c + 42;
}
int main()
{
  foo(73);
  printInt(a);
  return 0;
}
---------------
115
```

test-while1.m2

```
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    printInt(i);
    i = i - 1;
  }
  printInt(42);
  return 0;
}
---------------
5 4 3 2 1 42
```

test-while2.m2

```
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}
int main()
{
  printInt(foo(7));
  return 0;
}
---------------
14
```

9.11.2 Fail tests

fail-assign1.m2

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

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; /* Fail: assigning a bool to an integer */
}
---------------
Fatal error: exception Failure("Mismatched assignment type")
```

fail-assign2.m2

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

  b = 48; /* Fail: assigning an integer to a bool */
}
---------------
Fatal error: exception Failure("Mismatched assignment type")
```

fail-assign3.m2

```
void myvoid()
{
  return;
}
int main()
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}
---------------
Fatal error: exception Failure("Mismatched assignment type")
```

fail-expr1.m2

```
int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}
int main()
{
  return 0;
}
---------------
Fatal error: exception Failure("Invalid type for arithmetic
   operators")
```

fail-expr2.m2

```
int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
}
int main()
{
  return 0;
}
---------------
Fatal error: exception Failure("Invalid type for arithmetic
   operators")
```

fail-for1.m2

```
int main()
{
  int i;

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

  return 0;
}
---------------
Fatal error: exception Failure("Undefined ID j")
```

fail-for2.m2

```
int main()
```

```
{
  int i;

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

  return 0;
}
---------------
Fatal error: exception Failure("Undefined ID j")
```

fail-for3.m2

```
int main()
{
  int i;

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

  return 0;
}
---------------
Fatal error: exception Failure("Unrecognized function")
```

fail-func1.m2

```
int foo() {}

int bar() {}

int baz() {}

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

int main()
{
  return 0;
}
---------------
Fatal error: exception Failure("duplicate function bar")
```

fail-func2.m2

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

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

```
int main()
{
  return 0;
}
---------------
Fatal error: exception Failure("duplicate formal a in bar")
```

fail-func3.m2

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

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

int main()
{
  return 0;
}
---------------
Fatal error: exception Failure("illegal void formal b in bar")
```

fail-func4.m2

```
int foo() {}

void bar() {}

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

void baz() {}

int main()
{
  return 0;
}
---------------
Fatal error: exception Failure("function printStr may not be
   defined")
```

fail-func5.m2

```
int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;
  return 0;
}
int main()
{
  return 0;
}
---------------
Fatal error: exception Failure("illegal void local b in bar")
```

fail-func6.m2

```
void foo(int a, bool b)
{
}
int main()
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}
---------------
Fatal error: exception Failure("Incorrect number of
   arguments")
```

fail-func7.m2

```
void foo(int a, bool b)
{
}
int main()
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}
---------------
Fatal error: exception Failure("Incorrect number of
   arguments")
```

fail-global1.m2

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

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

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

fail-global2.m2

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

int main()
{
  return 0;
}
---------------
Fatal error: exception Failure("duplicate global b")
```

fail-if1.m2

```
int main()
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}
---------------
Fatal error: exception Failure("Undefined ID foo")
```

fail-if2.m2

```
int main()
{
  if (true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}
---------------
Fatal error: exception Failure("Undefined ID bar")
```

fail-nomain.m2

```

---------------
Fatal error: exception Failure("Unrecognized function")
```

fail-while.m2

```
int main()
{
  int i;

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

  while (true) {
    foo(); /* foo undefined */
  }
}
---------------
Fatal error: exception Failure("Unrecognized function")
```

9.12 Test2

9.12.1 Pass tests

test-controlFlow.m2

```
int main(){
        float a;
        int b;

        a = 9.3;

        if (a > 5.1){
                printStr("If statement");
        }
        else{
                printStr("Else statement");
        }

        while (a > 8.0){
                a = a - 2.0;
                printStr("While loop");
        }

        for (b = 0; b < 2; ++b){
                printStr("For loop");
        }

        return 0;
}
---------------
```

```
If statement
While loop
For loop
For loop
```

test-dataTypes.m2

```
int main(){
        int a;
        float b;
        String c;
        bool d;

        a = 1;
        b = 2.0;
        c = "ah";
        d = true;

        printInt(a);
        printFloat(b);
        printStr(c);
        printBool(d);

        return 0;
}
---------------
1 2.000000 ah
1
```

test-floatDet22.m2

```
int main(){
        matrix float [2][2] m;
        float det;
        m = [[1.0,2.0];
                 [3.0,4.0]];
        det = m[0][0] * m[1][1] - m[1][0] * m[0][1];

        printStr("The determinant of matrix");
        myprint(m);
        printStr("is");
        printFloat(det);

        return 0;
}
void myprint(matrix float [2][2] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
---------------
The determinant of matrix
1.000000 2.000000
3.000000 4.000000
is
-2.000000
```

test-floatDet33.m2

```
int main(){
        matrix float [3][3] m;
        float det;

        m = [[1.0,2.0,3.0];
                 [4.0,5.0,6.0];
                 [7.0,8.0,9.0]];

        det = det = m[0][0] * (m[1][1] * m[2][2] - m[1][2] *
           m[2][1])
                - m[0][1] * (m[1][0] * m[2][2] - m[1][2] *
                   m[2][0])
                + m[0][2] * (m[1][0] * m[2][1] - m[1][1] *
                   m[2][0]);

        printStr("The determinant of matrix");
        myprint(m);
        printStr("is");
        printFloat(det);

        return 0;
}
void myprint(matrix float [3][3] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
```

```
---------------
The determinant of matrix
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
is
0.000000
```

test-floatMatrixAccess.m2

```
int main(){
        int a;
        int b;
        matrix float [2][1] m;
        m = [[2.0];
         [3.2]];
    a = 0;
    b = 0;
    printFloat(m[a][b]);

        return 0;
}
---------------
2.000000
```

test-floatMatrixBinop1.m2

```
int main() {
  matrix float [2][2] m;
  m = [[2.0,1.0];
        [0.0,3.0]];

  printStr("Matrix m is");
  myprint(m);

  printStr("m + m is");
  myprint(m + m);

  printStr("m - m is");
  myprint(m - m);

  printStr("m * m is");
  myprint(m * m);

  return 0;
}
void myprint(matrix float [2][2] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
---------------
Matrix m is
2.000000 1.000000
0.000000 3.000000
m + m is
4.000000 2.000000
0.000000 6.000000
m - m is
0.000000 0.000000
0.000000 0.000000
m * m is
4.000000 5.000000
0.000000 9.000000
```

test-floatMatrixBinop2.m2

```
int main() {
  float a;
  matrix float [2][3] M;
  a = 9.0;
  M = [[0.0,0.0,0.0];
      [0.0,0.0,0.0]];

  printStr("Matrix M is");
  myprint(M);
  printStr("Float a is");
  printFloat(a);
  printStr("");

  printStr("a * M is");
  myprint(a * M);


  return 0;
}
void myprint(matrix float [2][3] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
---------------
Matrix M is
0.000000 0.000000 0.000000
0.000000 0.000000 0.000000
Float a is
9.000000
a * M is
0.000000 0.000000 0.000000
0.000000 0.000000 0.000000
```

test-floatMatrixFunc.m2

```
int main() {
  matrix float [2][2] m;
  m = [[1.0,1.0];
       [0.0,2.0]];

  printStr("Matrix m is");
  myprint(m);

  printStr("Rows of m");
  printInt(m:rows);
  printStr("");

  printStr("Columns of m");
  printInt(m:cols);
  printStr("");

  printStr("Transpose of m");
  myprint(m:transpose);

  printStr("Trace of m");
  printFloat(m:trace);
  printStr("");

  printStr("Submatrix (0,1,0,1) of m");
  printFloat(m:subMatrix[0,0,0,0]);
  printStr("");

  printStr("Submatrix (0,1,0,1) of m");
  myprint(m:subMatrix[0,1,0,1]);

  return 0;
}
void myprint(matrix float [2][2] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
---------------
Matrix m is
1.000000 1.000000
0.000000 2.000000
Rows of m
2
Columns of m
2
Transpose of m
1.000000 0.000000
1.000000 2.000000
Trace of m
3.000000
Submatrix (0,1,0,1) of m
1.000000
Submatrix (0,1,0,1) of m
1.000000 1.000000
0.000000 2.000000
```

test-for.m2

```
int main(){
        int i;
        i = 1;
        for (;i<2;){
                i = i + 1;
                printInt(i);
        }

        return 0;
}
---------------
2
```

test-graph.m2

```
int main() {
    matrix int [8][8] m;
    int i;
    int j;
    int k;
    int sum;
    int sumM;
    int new;
    m = [[21,9,1,13,12,13,30,11];
        [0,2,5,14,1,13,1,13];
        [30,9,12,31,54,16,18,8];
        [3,44,13,15,19,7,8,18];
        [3,45,45,20,31,7,8,18];
        [3,45,45,20,19,7,2,1];
        [23,13,13,13,23,7,4,1];
        [12,44,13,31,11,7,8,8]
        ];
    myprint(m);
    for(i = 0; i < 6; ++i) {
        sum = 0;
        sumM = 0;
        for(j = 0; j < 8; ++j) {
            for(k = 0; k < 8; ++k) {
                if(m[j][k] > 0) {
                    m[j][k] = m[j][k] - 1;
                    ++sum;
                }
                sumM = sumM + m[j][k];

            }
        }
        for(j = 0; j < 8; ++j) {
            for(k = 0; k < 8; ++k) {
                new = sum * m[j][k] / sumM;
                m[j][k] = m[j][k] + new;
            }
        }
        myprint(m);
        printStr("");

    }
    return 0;
}
void myprint(matrix int [8][8] M) {
    int i;
    int j;
    for(i = 0; i < M:rows; ++i) {
        for(j = 0; j < M:cols; ++j) {
            printInt(M[i][j]);
            printInt(999);
        }
        printStr("");
    }
}
---------------
21      999     9       999     1       999     13
   999     12      999     13      999     30      999
   11      999
0       999     2       999     5       999     14
   999     1       999     13      999     1       999
   13      999
30      999     9       999     12      999     31
   999     54      999     16      999     18      999
   8       999
3       999     44      999     13      999     15
   999     19      999     7       999     8       999
   18      999
3       999     45      999     45      999     20
   999     31      999     7       999     8       999
   18      999
3       999     45      999     45      999     20
   999     19      999     7       999     2       999
   1       999
23      999     13      999     13      999     13
   999     23      999     7       999     4       999
   1       999
12      999     44      999     13      999     31
   999     11      999     7       999     8       999
   8       999
21      999     8       999     0       999     12
   999     11      999     12      999     30      999
   10      999
0       999     1       999     4       999     13
   999     0       999     12      999     0       999
   12      999
30      999     8       999     11      999     31
   999     56      999     15      999     18      999
   7       999
2       999     45      999     12      999     14
   999     19      999     6       999     7       999
   18      999
2       999     46      999     46      999     20
   999     31      999     6       999     7       999
   18      999
2       999     46      999     46      999     20
   999     19      999     6       999     1       999
   0       999
```

```
23       999      12       999      12       999      12
   999      23       999      6        999      3        999
   0        999
11       999      45       999      12       999      31
   999      10       999      6        999      7        999
   7        999

21       999      7        999      0        999      11
   999      10       999      11       999      30       999
   9        999
0        999      0        999      3        999      12
   999      0        999      11       999      0        999
   11       999
30       999      7        999      10       999      31
   999      58       999      14       999      18       999
   6        999
1        999      46       999      11       999      13
   999      19       999      5        999      6        999
   18       999
1        999      47       999      47       999      20
   999      31       999      5        999      6        999
   18       999
1        999      47       999      47       999      20
   999      19       999      5        999      0        999
   0        999
23       999      11       999      11       999      11
   999      23       999      5        999      2        999
   0        999
10       999      46       999      11       999      31
   999      9        999      5        999      6        999
   6        999

21       999      6        999      0        999      10
   999      9        999      10       999      30       999
   8        999
0        999      0        999      2        999      11
   999      0        999      10       999      0        999
   10       999
30       999      6        999      9        999      31
   999      60       999      13       999      18       999
   5        999
0        999      47       999      10       999      12
   999      19       999      4        999      5        999
   18       999
0        999      48       999      48       999      20
   999      31       999      4        999      5        999
   18       999
0        999      48       999      48       999      20
```

```
    999     19      999     4       999     0       999
    0       999
23      999     10      999     10      999     10
    999     23      999     4       999     1       999
    0       999
9       999     47      999     10      999     31
    999     8       999     4       999     5       999
    5       999

21      999     5       999     0       999     9
    999     8       999     9       999     30      999
    7       999
0       999     0       999     1       999     10
    999     0       999     9       999     0       999
    9       999
30      999     5       999     8       999     31
    999     62      999     12      999     18      999
    4       999
0       999     48      999     9       999     11
    999     19      999     3       999     4       999
    18      999
0       999     49      999     49      999     20
    999     31      999     3       999     4       999
    18      999
0       999     49      999     49      999     20
    999     19      999     3       999     0       999
    0       999
23      999     9       999     9       999     9
    999     23      999     3       999     0       999
    0       999
8       999     48      999     9       999     31
    999     7       999     3       999     4       999
    4       999

21      999     4       999     0       999     8
    999     7       999     8       999     30      999
    6       999
0       999     0       999     0       999     9
    999     0       999     8       999     0       999
    8       999
30      999     4       999     7       999     31
    999     64      999     11      999     18      999
    3       999
0       999     49      999     8       999     10
    999     19      999     2       999     3       999
    18      999
0       999     50      999     50      999     20
    999     31      999     2       999     3       999
```

```
   18       999
0        999      50       999      50       999      20
   999      19       999      2        999      0        999
   0        999
23       999      8        999      8        999      8
   999      23       999      2        999      0        999
   0        999
7        999      49       999      8        999      31
   999      6        999      2        999      3        999
   3        999

21       999      3        999      0        999      7
   999      6        999      7        999      30       999
   5        999
0        999      0        999      0        999      8
   999      0        999      7        999      0        999
   7        999
30       999      3        999      6        999      31
   999      66       999      10       999      18       999
   2        999
0        999      50       999      7        999      9
   999      19       999      1        999      2        999
   18       999
0        999      52       999      52       999      20
   999      31       999      1        999      2        999
   18       999
0        999      52       999      52       999      20
   999      19       999      1        999      0        999
   0        999
23       999      7        999      7        999      7
   999      23       999      1        999      0        999
   0        999
6        999      50       999      7        999      31
   999      5        999      1        999      2        999
   2        999
```

test-intDet22.m2

```
int det22(matrix int [2][2] input){
        int det;
        det = input[0][0] * input[1][1] - input[1][0] *
           input[0][1];
        return det;
}
int main(){
        matrix int [2][2] m;
        m = [[1,2];
                 [3,4]];
        printStr("The determinant of matrix");
        myprint(m);
        printStr("is");
        printInt(det22(m));

        return 0;
}
void myprint(matrix int [2][2] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printInt(M[i][j]);
    }
    printStr("");
  }
}
---------------
The determinant of matrix
1 2
3 4
is
-2
```

test-intDet33.m2

```
int det33(matrix int [3][3] input){
        int det;
        det = input[0][0] * (input[1][1] * input[2][2] -
           input[1][2] * input[2][1])
                - input[0][1] * (input[1][0] * input[2][2] -
                   input[1][2] * input[2][0])
                + input[0][2] * (input[1][0] * input[2][1] -
                   input[1][1] * input[2][0]);

        return det;
}
int main(){
        matrix int [3][3] m;
        m = [[1,2,3];
                 [4,5,6];
                 [7,8,9]];

        printStr("The determinant of matrix");
        myprint(m);
        printStr("is");

        printInt(det33(m));
        return 0;
}
void myprint(matrix int [3][3] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printInt(M[i][j]);
    }
    printStr("");
  }
}
---------------
The determinant of matrix
1 2 3
4 5 6
7 8 9
is
0
```

test-intMatrixAccess.m2

```
int main(){
        int a;
        int b;
        matrix int [4][4] m;
        m = [[2,9,1,1];
        [0,2,1,1];
        [1,1,1,1];
        [3,1,1,1]];
    a = 0;
    b = 0;
    printInt(m[a][b]);
        return 0;
}
---------------
2
```

test-intMatrixBinop1.m2

```
int main() {
  matrix int [4][4] m;
  m = [[2,9,1,1];
        [0,2,1,1];
        [1,1,1,1];
        [3,1,1,1]];
  printStr("Matrix m is");
  myprint(m);

  printStr("m + m is");
  myprint(m + m);

  printStr("m - m is");
  myprint(m - m);

  printStr("m * m is");
  myprint(m * m);

  return 0;
}
void myprint(matrix int [4][4] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printInt(M[i][j]);
    }
    printStr("");
  }
}
---------------
Matrix m is
2       9       1       1
0       2       1       1
1       1       1       1
3       1       1       1
m + m is
4       18      2       2
0       4       2       2
2       2       2       2
6       2       2       2
m - m is
0       0       0       0
0       0       0       0
0       0       0       0
0       0       0       0
m * m is
8       38      13      13
4       6       4       4
6       13      4       4
10      31      6       6
```

test-intMatrixBinop2.m2

```
int main() {
  int a;
  matrix int [2][3] M;
  a = 9;
  M = [[0,0,0];
      [0,0,0]];

  printStr("Matrix M is");
  myprint(M);
  printStr("Int a is");
  printInt(a);
  printStr("");

  printStr("a * M is");
  myprint(a * M);


  return 0;
}
void myprint(matrix int [2][3] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printInt(M[i][j]);
    }
    printStr("");
  }
}
---------------
Matrix M is
0       0       0
0       0       0
Int a is
9
a * M is
0       0       0
0       0       0
```

test-intMatrixFunc.m2

```
int main() {
  matrix int [2][2] m;
  m = [[1,1];
      [0,2]];

  printStr("Rows of m");
  printInt(m:rows);
  printStr("");

  printStr("Columns of m");
  printInt(m:cols);
  printStr("");

  printStr("Transpose of m");
  myprint(m:transpose);

  printStr("Trace of m");
  printInt(m:trace);
  printStr("");

  printStr("Submatrix (0,1,0,1) of m");
  printInt(m:subMatrix[0,0,0,0]);
  printStr("");

  printStr("Submatrix (0,1,0,1) of m");
  myprint(m:subMatrix[0,1,0,1]);

  return 0;
}
void myprint(matrix int [2][2] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printInt(M[i][j]);
    }
    printStr("");
  }
}
---------------
Rows of m
2
Columns of m
2
Transpose of m
1       0
1       2
Trace of m
3
Submatrix (0,1,0,1) of m
1
Submatrix (0,1,0,1) of m
1       1
0       2
```

test-inverse.m2

```
int main(){
        matrix float [2][2] m;
        matrix float [2][2] inverse;
        float det;

        m = [[1.0,2.0];
                 [3.0,4.0]];

        det = m[0][0] * m[1][1] - m[1][0] * m[0][1];

        if(det != 0.0){
        inverse[0][0] = m[1][1] / det;
            inverse[0][1] = - m[0][1] /det;
            inverse[1][0] = - m[1][0] /det;
            inverse[1][1] = m[0][0] / det ;
            printStr("The inverse of matrix");
                myprint(m);
                printStr("is");
                myprint(inverse);
        }
        else{
                printStr("Cannot find inverse");
        }

        return 0;
}
void myprint(matrix float [2][2] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
---------------
The inverse of matrix
1.000000        2.000000
3.000000        4.000000
is
-2.000000       1.000000
1.500000        -0.500000
```

test-matrixChainMult.m2

```
int main(){
        matrix int [1][7] p;
        matrix int[6][6] DP;
        int n;
        int i;
        int l;
        int j;
        int k;
        int q;
        int tmp1;
        int tmp2;
        int tmp3;

        n = DP:cols;

        p = [[30,35,15,5,10,20,25]];

        for (i = 0; i < DP:cols; ++i){
                DP[i][i] = 0;
        }

        for (l = 2; l < n+1; ++l){
                for(i = 0; i < n-l+1; ++i){
                        j = i + l - 1;
                        DP[i][j] = 100000;
                        for(k = i; k < j; ++k){
                                tmp1 = k+1;
                                tmp2 = i-1;
                                tmp3 = j+1;

                                q = DP[i][k] + DP[tmp1][j] +
                                   p[0][i] * p[0][tmp1] *
                                   p[0][tmp3];

                                if (q < DP[i][j]){
                                        DP[i][j] = q;
                                }
                        }
                }
        }
        myprint(DP);

        return 0;
}
void myprint(matrix int [6][6] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
        printInt(M[i][j]);
      }
      printStr("");
    }
}
---------------
0       15750   7875    9375    11875   15125
0       0       2625    4375    7125    10500
0       0       0       750     2500    5375
0       0       0       0       1000    3500
0       0       0       0       0       5000
0       0       0       0       0       0
```

test-norm1.m2

```
int norm1(matrix int [4][4] input)
{
        int row;
        int col;
        int maxSum;
        int Sum;
        int finalIndex;
        int index;
        int index2;
        maxSum = -10000000;
        Sum = 0;
        row = input:rows;
        col = input:cols;


        for(index = 0; index < col; index = index + 1) {
                Sum = 0;
                for(index2 = 0; index2 < row; index2 = index2
                   + 1) {
                        Sum = Sum + input[index2][index];
                }
                if(Sum > maxSum) {
                        maxSum = Sum;
                        finalIndex = index;
                }
        }

        return maxSum;
}
int main() {
  matrix int [4][4] m;
  m = [[2,9,1,1];
        [0,2,1,1];
        [1,1,1,1];
        [3,1,1,1]];

  printInt(norm1(m));

  return 0;
}
---------------
13
```

test-pow.m2

```
void myprint(matrix int [4][4] M) {
    int i;
    int j;
    for(i = 0; i < M:rows; ++i) {
        for(j = 0; j < M:cols; ++j) {
            printInt(M[i][j]);
        }
        printStr("");
    }
}

void pow(matrix int[4][4] input, int N) {
    matrix int[4][4] res;
    matrix int[4][4] ret;
    int flag;
    flag = 1;
    res = input;
    while(N > 1) {
        if(N - N / 2 * 2 == 0) {
            res = res * res;
            N = N / 2;
        }
        else {
            if(flag == 1) {
                ret = res;
                flag = -1;
            }
            else {
                ret = ret * res;
            }
            res = res * res;
            N = N / 2;
        }
    }
    if(flag == 1) {
        myprint(res);
    }
    else {
        ret = ret * res;
        myprint(ret);
    }
}
int main() {
    matrix int [4][4] m;
    int time;
    time = 2;
    m = [[2,9,1,1];
         [0,2,1,1];
         [1,1,1,1];
         [3,1,1,1]
    ];
    pow(m, 6);
    return 0;
}
---------------
13984   39720   13788   13788
4272    11848   4176    4176
5960    16604   5760    5760
10744   30196   10360   10360
```

test-print.m2

```
int main() {
        printInt(5);
        printFloat(9.9);
        printBool(true);
        return 0;
}
---------------
5       9.900000        1
```

test-solveEqn2by2.m2

```
int main() {
  matrix float [2][2] m1;
  matrix float [2][1] m2;

  m1 = [[2.0,1.0];
        [3.0,3.0]];

  m2 = [[4.0];
        [9.0]];

  solveEqn(m1,m2);
  return 0;
}

void solveEqn(matrix float [2][2] m1, matrix float [2][1] m2){
  float det;
  matrix float [2][2] inverse;
  matrix float [2][1] result;
  int i;
  int j;

  det = m1[0][0] * m1[1][1] - m1[1][0] * m1[0][1];

  if (det != 0.0){
    inverse[0][0] = m1[1][1] / det;
    inverse[0][1] = - m1[0][1] /det;
    inverse[1][0] = - m1[1][0] /det;
    inverse[1][1] = m1[0][0] / det ;

    result = inverse * m2;

    printStr("Input equation is:");
    printFloat(m1[0][0]);
    printStr("x + ");
    printFloat(m1[0][1]);
    printStr("y = ");
    printFloat(m2[0][0]);
    printStr("");
    printStr("");

    printFloat(m1[1][0]);
    printStr("x + ");
    printFloat(m1[1][1]);
    printStr("y = ");
    printFloat(m2[1][1]);
    printStr("");
    printStr("");

    printStr("Solutions are:");
    printStr("x = ");
    printFloat(result[0][0]);
    printStr("");
    printStr("y = ");
    printFloat(result[1][0]);
  }
  else{
    printStr("Cannot find a solution");
  }
```

```
}
---------------
Input equation is:
2.000000        x +
1.000000        y =
4.000000

3.000000        x +
3.000000        y =
2.000000

Solutions are:
x =
1.000000
y =
2.000000
```

test-solveEqn3by3.m2

```
int main() {
  matrix float [3][3] m;
  matrix float [3][1] m2;
  matrix float [3][3] tr;
  matrix float [3][3] adj;
  matrix float [3][3] inverse;
  matrix float [3][1] result;
  float det;
  int i;
  int j;

  m = [[1.0, 1.0, 1.0];
       [2.0, 3.0, 1.0];
       [1.0, 1.0, 5.0]];

  m2 = [[3.0];
        [6.0];
        [7.0]];

  tr = m:transpose;

  det = m[0][0] * (m[1][1] * m[2][2] - m[1][2] * m[2][1])
    - m[0][1] * (m[1][0] * m[2][2] - m[1][2] * m[2][0])
    + m[0][2] * (m[1][0] * m[2][1] - m[1][1] * m[2][0]);

  if (det != 0.0){
    adj[0][0] = tr[1][1]*tr[2][2] - tr[1][2]*tr[2][1];
    adj[0][1] = tr[1][0]*tr[2][2] - tr[1][2]*tr[2][0];
    adj[0][2] = tr[1][0]*tr[2][1] - tr[1][1]*tr[2][0];
    adj[1][0] = tr[0][1]*tr[2][2] - tr[0][2]*tr[2][1];
    adj[1][1] = tr[0][0]*tr[2][2] - tr[0][2]*tr[2][0];
    adj[1][2] = tr[0][0]*tr[2][1] - tr[0][1]*tr[2][0];
    adj[2][0] = tr[0][1]*tr[1][2] - tr[0][2]*tr[1][1];
    adj[2][1] = tr[0][0]*tr[1][2] - tr[0][2]*tr[1][0];
    adj[2][2] = tr[0][0]*tr[1][1] - tr[0][1]*tr[1][0];


    adj[0][1] = adj[0][1] * (0.0-1.0);
    adj[1][0] = adj[1][0] * (0.0-1.0);
    adj[1][2] = adj[1][2] * (0.0-1.0);
    adj[2][1] = adj[2][1] * (0.0-1.0);

    for(i = 0; i < inverse:rows; ++i){
      for (j = 0; j < inverse:cols; ++j){
        inverse[i][j] = adj[i][j] / det;
      }
    }

    result = inverse * m2;

    printStr("Input equations are:");

    for (i = 0; i < m:rows; ++i){
      printFloat(m[i][0]);
      printStr("x + ");
      printFloat(m[i][1]);
      printStr("y ");
      printFloat(m[i][2]);
      printStr("z = ");
      printFloat(m2[i][0]);
      printStr("");
      printStr("");
    }

    printStr("Solutions are:");
    printStr("x = ");
    printFloat(result[0][0]);
    printStr("");
    printStr("y = ");
    printFloat(result[1][0]);
    printStr("");
    printStr("z = ");
    printFloat(result[2][0]);

  }
  else{
    printStr("Cannot find a solution");
  }
  return 0;
}
void myprint(matrix float [3][1] M){
  int i;
  int j;
  for(i = 0; i < M:rows; ++i){
    for(j = 0; j < M:cols; ++j){
      printFloat(M[i][j]);
    }
    printStr("");
  }
}
---------------
Input equations are:
1.000000        x +
1.000000        y
1.000000        z =
3.000000

2.000000        x +
3.000000        y
1.000000        z =
6.000000

1.000000        x +
1.000000        y
5.000000        z =
7.000000

Solutions are:
x =
1.000000
y =
1.000000
z =
1.000000
```

9.12.2 Fail tests

fail-duplicateGlobal.m2

```
int main(){
        int a;
        int a;
        return 0;
}
---------------
Fatal error: exception Failure("duplicate local a in main")
```

fail-equality.m2

```
int main(){
        matrix int [1][1] a;
        matrix int [1][1] b;
        a = [[1]];
        b = [[2]];
        a == b;
        return 0;
}
---------------
Fatal error: exception Failure("Illegal type for equality
   operators")
```

fail-functionArgument.m2

```
int main(){
        int a;
        int b;
        myfunc(a);

        return 0;
}
void myfunc(int a, int b){
}
---------------
Fatal error: exception Failure("Incorrect number of
   arguments")
```

fail-logical.m2

```
int main(){
        matrix float [1][1] a;
        matrix float [1][1] b;
        a = [[1.1]];
        b = [[2.2]];
        if (a && b){
                printStr(1);
        }
        return 0;
}
---------------
Fatal error: exception Failure("Illegal type for logical
   operators")
```

fail-matrixAccess.m2

```
int main(){
        int m;
        m[0][0] = 1;

        return 0;
}
---------------
Fatal error: exception Failure("Cannot operate on nonmatrix")
```

fail-matrixBinop.m2

```
int main(){
        matrix int [3][2] m;
        matrix int [1][1] m2;
        m = [[2,9];
        [0,2];
        [1,1]];
    m2 = [[1]];
    m * m2;

    return 0;
}
---------------
Fatal error: exception Failure("Incorrect dimention/type for
   matrix multiplication")
```

fail-matrixBinop2.m2

```
int main(){
        matrix int [3][2] m;
        matrix int [1][1] m2;
        m = [[2,9];
        [0,2];
        [1,1]];
    m2 = [[1]];
    m + m2;

    return 0;
}
---------------
Fatal error: exception Failure("Incorrect dimention/type for
   matrix addition/subtraction")
```

fail-matrixBinop3.m2

```
int main(){
        float a;
        matrix int [3][2] m;
        a = 2.2;
        m = [[2,9];
        [0,2];
        [1,1]];
    a/m;

    return 0;
}
---------------
Fatal error: exception Failure("Invalid type for arithmetic
   operators")
```

fail-matrixBinop4.m2

```
int main(){
        String a;
        matrix int [3][2] m;
        a = "hello";
        m = [[2,9];
        [0,2];
        [1,1]];
    a + m;

    return 0;
}
---------------
Fatal error: exception Failure("Invalid type for arithmetic
   operators")
```

fail-matrixBinop5.m2

```
int main(){
        matrix int [3][2] m;
        matrix float [3][2] m2;
        m = [[2,9];
        [0,2];
        [1,1]];
    m2 = [[2.0,9.0];
        [0.0,2.0];
        [1.0,1.0]];
    m + m2;

    return 0;
}
---------------
Fatal error: exception Failure("Incorrect dimention/type for
   matrix addition/subtraction")
```

fail-matrixBinop6.m2

```
int main(){
        matrix int [1][2] m;
        float a;
        m[0][0] = 1;
        m[0][1] = 2;
        a = 9.0;
        a - m;

        return 0;
}
---------------
Fatal error: exception Failure("Invalid type for arithmetic
   operators")
```

fail-matrixDim.m2

```
int main(){
        matrix int [2.1][9.2] m;

        return 0;
}
---------------
Fatal error: exception Failure("Integer required for matrix
   dimension")
```

fail-mismatchedAssign.m2

```
int main(){
        String a;
        a = 2;
        return 0;
}
---------------
Fatal error: exception Failure("Mismatched assignment type")
```

fail-moreThanOneTypeInMat.m2

```
int main(){
        matrix int [2][1] a;
        a = [[1];
                [1.2]];
        return 0;
}
---------------
Fatal error: exception Failure("More than one datatype in a
   matrix")
```

fail-print.m2

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

}
---------------
Fatal error: exception Failure("function printInt may not be
   defined")
```

fail-stringBinop.m2

```
int main(){
        String a;
        String b;
        a = "he";
        b = "ld";
        a - b;

        return 0;
}
---------------
Fatal error: exception Failure("Invalid operation on String")
```

fail-subMatrix.m2

```
int main() {
  matrix float [3][2] m;
  m = [[2.1, 1.9];
       [0.3, 9.7];
       [1.12, 9.21]];
  m:subMatrix[9.2,0,0,0];

  return 0;
}
---------------
Fatal error: exception Failure("Integer required for matrix
   dimension/index")
```

fail-trace.m2

```
int main() {
  matrix int [3][1] m;
  m = [[2];
       [0];
       [1]];
  printStr("Trace of m");
  printInt(m:trace);
  printStr("");

  return 0;
}
---------------
Fatal error: exception Failure("Trace only operates on square
   matrices")
```

fail-transpose.m2

```
int main(){
        float a;
        a:transpose;

        return 0;
}
---------------
Fatal error: exception Failure("Cannot operate on nonmatrix")
```

fail-unop.m2

```
int main(){
        String a;
        a = "js";

        ++a;

        return 0;
}
---------------
Fatal error: exception Failure("Invalid datatype for unop")
```