LOL

The Linear-Algebra Oriented Programming Language

PROGRAMMING LANGUAGES AND TRANSLATORS FINAL REPORT
SPRING 2019

Professor Ronghui Gu

Jerry Lin, David Wan, Shuqi Chen, Zichuan Wang

May 19, 2019

Contents

1 Introduction 4

2 Language Tutorial
2.1 Project Setup 5
2.2 Tutorial 5

3 Language Reference Manual
3.1 Comments and Whitespace 7
3.1.1 Comments 7
3.1.2 Whitespace 7
3.2 Types, Operators, and Expressions 7
3.2.1 Data Types 7
3.2.1.1 Primitive Data Type 7
3.2.1.2 The Void Keyword 7
3.2.2 Operators 7
3.2.2.1 Precedence and Associativity 8
3.2.2.2 Assignment Operator 8
3.2.2.3 Arithmetic Operators 8
3.2.2.4 Logical Operators 8
3.2.3 Expressions 9
3.3 Variables 9
3.3.1 Variable Naming 9
3.3.2 Scope of Variables 9
3.3.3 Variable Declarations 10
3.3.4 Variable Assignment 10
3.4 Statements 10
3.4.1 Simple Statements 10
3.4.2 The For Loops 10
3.4.3 The While Loops 11
3.4.4 The If Statements 11
3.5 Functions 12

3.5.1 Functions Overview 12
3.5.1.1 Function Declaration 12
3.5.1.2 Anonymous Functions 12
3.5.1.3 Calling Functions 12
3.5.2 Functional Programming in LOL 13
3.5.2.1 Referential Transparency 13
3.5.2.2 First-Class and Higher-Order Functions 14
3.5.2.3 Built-In Functions 15
3.6 Structures 16
3.6.1 List 16
3.6.1.1 Declaring Lists 16
3.6.1.2 Defining Lists 16
3.6.1.3 Lists in Lists 16
3.6.1.4 Accessing List Elements 16
3.6.2 Matrix 17
3.6.2.1 Declaring Matrix. 17
3.6.2.2 Defining Matrix 17
3.6.2.3 The Dimension of Matrix 17
3.6.2.4 Accessing Matrix Elements 17
3.6.2.5 Matrix Functions 17

4 Project Plan
4.1 Process Used for Planning, Specification, and Development 19
4.2 Programming Style Guide 19
4.3 Project Timeline 20
4.4 Roles and Responsibilities 20
4.5 Software Development Environment 21
4.6 Project log 21

5 Architectural Design
5.1 Block Diagram 43
5.2 Scanner 43
5.3 Parser 43
5.4 Semantic Checker 43
5.4 Code Generation 44
5.6 Division of Labor 44

6 Testing
6.1 Testing Suite and Automation 45

6.2 Example Input-Output 45

7 Lessons Learned

7.1 Jerry Lin 47
7.2 David Wan 47
7.3 Shuqi Chen 47
7.4 ZichuanWang 48

8 Appendix

8.1 src Files 49
8.2 test Files 136
8.3 Makefiles 157

1 Introduction

With the rapid development of Machine Learning, Robotics, Computer Vision, and Quantum Computing, Linear Algebra is playing an increasingly important role in our society. Yet, performing linear algebra operations on a computer still requires some barriers of entry. In a mainstream programming language, including Java and Python, the user needs to import external libraries and learn somewhat complicated commands to create classes such as Matrix. Moreover, these libraries might be written in languages that might run slowly on the hardware, affecting the performance of the task.

LinearAlgebra Oriented Language (LOL) is created under one principle: a programming language specifically designed for mathematicians to efficiently perform linear algebra tasks. Although programming languages such as Python or Matlab provide powerful functions, remembering what each function is called or which symbol represents matrix multiplication can be annoying. Thus, a language paradigm is developed that provides powerful and easy-to-remember built-in linear algebra functionalities so that one can focus solely on mathematics without worrying too much about the language details. With LOL, we hope to provide powerful libraries like Python and Matlab with the standard function calling (a.get()), as well as intuitive syntax (m1@m2) so that one can perform complex operations directly from a paper without any hassle. In addition, LOL is created with functional programming in mind, aiming to help mathematicians produce more robust and logical code.

2 Language Tutorial

2.1 Project Setup

To setup LOL, first obtain the source code and then install the following software on your system: LLVM (version 7.0.0 or higher), OCaml (version 4.05.0 or higher), Clang (version 7.0.1 or higher), OPAM and GNU Scientific Library (GSL) (version 2.5 or higher). At this point you can just type "make" to compile all the necessary files for the project.

After make succeeded, you can run the testall.sh script and then make sure all test cases are passed. To run your own LOL program, just use the command "./run.sh yourfilename.lol".

2.2 Tutotial

Although built with support for first class function, LOL uses syntax very familiar to C-like language programmers.

Hello World

```
1   println("Hello world");
2
```

As you can see the code for the hello world program is very close to its equivalent in C-like languages. In LOL there is no need for a main function. LOL has the println function that is essentially the same as the C println function, which takes in a string and output the content to the screen. Notice the semicolon in the end: each statement is ended with a semicolon.

Variable Declaration and Initialization

```
1   int a = 0;
2   List<int> list = [1,2];
3   Matrix mat = Matrix([0.0,-1.0]);
4
```

To declare a variable, write the type name followed by the variable name. For Lists the type is specified in <>. To initialize a baisc type varible, just pass the value to it. List can be either auto initialized to empty list ("List<int> list;" will auto initialize the list to an empty int list) or initialized to contain certain values by following the format shown above ([element1, element2, ...]). Note Matrix only supports floating point numbers.

Function Declaration and Function Call

```
1   float PI = 3.141;
2
3   func float calculate_area (float r) {
4       return r * r * PI;
5   }
6
7   println(str_of_float(calculate_area(10.0)));
8
```

To declare a function, start with the “func” keyword, and then the return type and name of function, and finally the parameter(s). The body of the function is enclosed in {}. The return statement in a function consists of the “return” keyword and the content to be returned.

Control Flow

```
int i = 0;
while (i < 3) {
    println(str_of_int(i));
    i++;
}

func string foo() {
  for (int i = 0; i < 10; i = i+1) {
    if (i == 5){
      return "hello " + str_of_int(i);
    }
  }
}

println(foo());

```

In LOL while loop, for loop and if conditional are supported. The syntax is very close to C-like languages, as shown above. Note for for loops, the initialization, predicate and increment statements can all be omitted (i.e. for (; ;) is allowed).

3 Language Manual

3.1 Comments and Whitespace

3.1.1 Comments

LOL supports both single-line comments and block comments. Single-line comments are denoted with two forward slashes:

```
// this is a line comment
```

Block comments are written in between /* and */. *Block comments can be nested as long as the opening /* and closing */ are matched:

```
/* this is a block comment
*  this is a block comment
*  this is a block comment
*/
```

3.1.2 Whitespace

Whitespace, including newline character, tab, and space, is used only to separate tokens and is otherwise ignored by the LOL compiler.

3.2 Types, Operators, and Expressions

3.2.1 Data Types

3.2.1.1 Primitive Data Type

int: an architecture-dependent signed 32-bit integer type that is made up of a sequence of digits representing a number in base 10. A - is used to denote negative numbers. It is a right-associative operator.
float: an architecture-dependent signed IEEE 64-bit floating-point type. A - is used to denote negative numbers. It is a right-associative operator.

string: a sequence of characters in ASCII enclosed in double-quotes and excludes the double-quote " character. There are no escape sequences. So, for example, you won't be able to print out double-quote characters by escaping the double-quote, but you can print out two single-quotes if you would like something similar to double-quotes.

bool: an expression with a value of true or false.

3.2.1.2 The Void Keyword

The type void has no associated value. It allows us to create functions that either do not require any parameters or do not return a value.

3.2.2 Operators

3.2.2.1 Precedence and Associativity

3.2.2.1.1 Precedence

LOL has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. If there are two or more operators in an expression, then the operator of highest priority will be executed. For example, multiplication and division have a higher precedence than addition and subtraction. In expression 1+2*5, multiplication * operator will be processed first and then addition. It is because multiplication has higher priority of precedence than addition.
We can change the priority of a LOL operator by enclosing the lower order priority operator in parentheses but not the associativity. In the previous example, if we include parentheses in the expression (1+2)*5, then addition will be done first because parentheses has higher priority than multiplication operator.

3.2.2.1.2 Associativity

When an expression has two operators with the same precedence, the expression is evaluated according to its associativity. For example, 2*2*3 is treated as (2*2)*3 since the * operator has right-to-left associativity.

3.2.2.2 Assignment Operator

The equal sign = is used for assignment. It has right-to-left associativity.

3.2.2.3 Arithmetic Operator

- + is used for addition in the traditional mathematical sense for int and float types:

  ```
  int a = 2;
  int b = 3;
  int sum = a + b; // sum: 5
  ```

  In addition, it can be used to concatenate two strings:

  ```
  string first = “hello ”;
  string second = “world”;
  string result = first + second; // result: “hello world”
  ```

- - is used for subtraction in the traditional mathematical sense for int and float types.
- * is used for multiplication in the traditional mathematical sense for int and float types.
- / is used for division in the traditional mathematical sense for int and float types. If the dividend should not be 0.
- @ is used for outer product in the traditional mathematical sense for vectors
- ^ is used for division in the traditional mathematical sense for int and float types, and tensor object.
- == is used for checking equality. This boolean operator works for int, float and string types.
- != is used for checking inequality. This boolean operator works for int, float and string types, and other defined structs.
- <, >, >=, <= work only for int and float types.

3.2.2.4 Logical Operators

! is used for *NOT* in boolean expressions. AND and OR are reserved for keywords in boolean expressions.

3.2.3 Expressions

An expression consists of at least one operand and zero or more operators. Operands are typed objects such as constants, variables, and function calls that return values. Below are some examples:

```
47
2 + 2
```

For parentheses group subexpressions, innermost expressions are evaluated first:

```
(2 * (3 + 5) )
```

3.3 Variables

3.3.1 Variable Naming

All variable names must follow [a-zA-Z][a-zA-Z0-9]*. Reserved words are not allowed. Reserved words are as follows:

int

float

string

bool, true, false

List,Tensor

if, elif, else

for, while

func

return

void

in

struct

AND, OR,

var (optional)

3.3.2 Scope of Variables

LoL is a statically scoped language which supports both local variables and global variables. Variables declared inside blocks (e.g. functions, loops, if statements) exist only inside the block. If there is a variable with the same name already declared, this new variable will override the previous one in the scope of its block.
Variables declared outside of blocks have global scope. The global variables can be accessed anywhere in the program following their declaration.
Multiple variables and functions of the same name, even of different types, can not be declared in the same scope.

3.3.3 Variable Declarations

LOL is a strong typed language. Each variable has a type specified at the time of declaration. The type must be one of the primitive data types or struct specified above.

3.3.4 Variable Assignment

The equal operator = is used for assignment. It sets the the value of the expression on the right-hand side of = to the variable on the left-hand side of =. You can also use +=, -=, *=, /=, for easier assignment when having the same variable on both side.
The type of the expression on the right-hand side must be consistent with the type of the variable on the left-hand side.

3.4 Statements

3.4.1 Simple Statements

Statements are terminated with semicolons ;. A semicolon cannot be by itself. It must be preceded by a statement. A statement can only be assignment, call, increment or decrement.

3.4.2 The For Loops

For loops can be used to specify iteration and looping behavior.
A for loop statement has a header and a body. The header consists of three expressions separated by semicolons. Both parts (the header and the body) are mandatory. The header has an initialization statement, a testing condition, an increment/decrement expression.
The initialization statement is evaluated one time only. It is evaluated before you test the test condition for the first time. You can declare variables in this statement or have an expression here (or even any statement). This initialization statement is optional. If you choose to omit the initialization statement, you still must have one semicolon denoting where it would have ended.
The test condition is tested before you enter the loop the first time and then every time after the increment/decrement expression is evaluated until the test condition is false. This test statement is optional. If you choose to omit the test statement, you still must have one semicolon denoting where it would have ended.
Finally, the increment/decrement expression is evaluated after each loop iteration, before the test condition is tested again. This part is optional. If you choose to omit the increment/decrement expression, you still must have one semicolon denoting where it would have ended.
The loop body can contain any number of statements, including zero statements. The brackets are always required, even if the loop body is empty. A loop iteration consists of evaluating the statements in the loop body.
Example:

```
int sum = 0;
for (int i = 0; i < 10; i = i + 1) {
    sum = sum + i;
}
```

The initialization statement, test condition, and increment/decrement expression can all be omitted to get an infinite loop. Note, there is no break statement so it is often best to write potentially infinite loops inside of functions so you can use return statements to stop them.

```
for ( ; ; ) {
    // infinite loop
}
```

3.4.3 The While Loops

While loops repeatedly execute a block of code as long as the test condition is true. A while loop statement has a header followed by curly braces that contain the body of the while loop, which is the block of code that is executed as long as the condition in the header is true. The header has the keyword while followed by parentheses enclosing the condition to test, which should evaluate to either true or false.

Example:

```
int i = 0;
while (i < 3) {
    i++;
}
```

3.4.4 The If Statements

The if statement supports conditional execution. The if statement has a boolean expression in parenthesis followed by curly braces for the body. The body of the if statement can be zero or more statements. The if statement can then be followed by any number of optional elif statements, which must have a boolean expression specified inside its parenthesis, just like an if statement, and can contain zero or more statement in its body, which must be enclosed by curly braces as well. The elif statement (or the if statement if elif statements were omitted) can be followed by a single, optional else statement. The else statement does not have a boolean expression in parenthesis. The word "else" is simply followed by the curly braces for its body. If the boolean expression for the given statement evaluates to true, then the body associated with that expression is evaluated, and then any following elif/else statements are skipped and the next statement is evaluated. The parentheses and braces are always required.

Example:

```
if (x > 0) {
    println("x is positive"); // println is a built-in function.
} elif (x == 0) {
    println("x is zero");
} else {
    println("x is negative");
}
if (x == 0) {
    x = x + 5;
}
```

3.5 Functions

3.5.1 Functions Overview

3.5.1.1 Function Declaration

The keyword func is reserved for declaring function variables. To declare a variable of type func, you must include the keyword func, followed by the return type of the this function. Please note that if the function does not have any return value, please declare void as the return type.

After the return type is declared, you need to name your function. It is recommended to have function names in lowercase, as some uppercase letters are reserved for mathematical operations. After a function name is given, a set of parentheses needs to follow in which you can specify the parameters to be passed into this function. If a function does not take any parameter, simply leave the parentheses as they are.

In the function body, you can declare variables within the function scope, provide statements, and specify return value with the return keyword, if necessary.

Please note, a function declaration does not require a semicolon after the closing bracket.

A function definition has this form:

```
func return-type function-name (parameter declarations, if any) {
    declarations
    statements
    return statement, if any
}
```

Here are some examples:

```
// a function named “foo”, which takes an integer and returns a new integer
func int foo (int a) {
    return a + 1;
}

// a function named “bar”, which does not take any parameter and always returns the string “lol
func string bar () {
    return “lol”;
}
```

3.5.1.3 Calling Functions

To call a function, put the function name followed by parentheses. Inside the parentheses, put the arguments for the function. The arguments you provide need to match the type and number of arguments that your function expects. Finally, put a ; at the end of this expression.
Here is an example:

```
// declare a function definition
func int addTwo (int a) {
    return a + 2;
}

println(addTwo(3));
// Output: 5
```

3.5.2 Functional Programming in LOL

LOL supports many features for functional programming. Applying functional programming principles in LOL can lead to improved code quality, faster debugging, strong logical support, and ease of testing. Below are functional programming elements supported and recommended by LOL. Functional programming is not strictly enforced in LOL. Programmers can choose between functional and imperative programming depending on their needs.

3.5.2.1 Referential Transparency

Referential transparency is a crucial concept in functional programming. To achieve referential transparency, a function needs to be pure and immutable.
A pure function is a function which:

1. Given the same input, will always return the same output;
2. Produces no side effects.

Consider the code snippet below:

```
float PI = 3.14;

func float calculate_area (float r) {
    return r * r * PI;
}

println(calculate_area(10.0));
```

The above function is impure because it depends on a global variable, which is subject to mutation. When the value for global variable changes, the function calculate_area returns a different value given the same input, which violates the first rule of pure function.
Consider the modified code:

```
float PI = 3.14;

func float calculate_area (float r, float pi) {
    return r * r * pi;
}

println(calculate_area(10.0, PI));
```

The function calculate_area now becomes pure, since it only relies on the parameters passed into the function. You can also observe that this function always returns the same output, given the same inputs. Therefore, the function is referentially transparent.

3.5.2.2 First-Class and Higher Order Functions

First class function is an essential element of functional programming. The idea of functions as first-class entities is that functions are also treated as values and used as data. Functions as first-class entities can:

1. refer to it from constants and variables
2. pass it as a parameter to other functions
3. return it as result from other functions

Suppose we have the following two functions:

```
func float divideNum (float a, float b) {
    return a / b;
}

func float multipleNum (float a, float b) {
    return a * b;
}
```

Since LOL supports first-class functions, meaning that functions can be treated and passed like value, we can combine the above functions into one:

```
func float ops(func f, float a, float b) {
    return f(a, b);
}

println(ops((/), 2.0, 3.0));
println(ops((*), 2.0, 3.0));
```

As you can see, a function f is passed into the parameter of function ops, just like a regular variable. Given the support for First-Class Functions, LOL naturally also supports Higher-Order Functions. A higher-order function is a function that either:

1. takes one or more functions as arguments, or
2. returns a function as its result

Consider the following code snippet:

```
func func(float,float:float) apply_f(func f) {
    // returns an anonymous function that executes function f on two float parameters
    return func float (float i, float j) {
        return f(i) + f(j);
    };
}
```

The apply_f function we implemented above is a higher-order function because it takes an operator function as an argument and uses it.

3.5.2.2 Built-In Functions

Built-in functions are functions that are predefined in the Shoo language. These functions can be called anywhere without the user having to define them first. All built-in functions are also first-class functions.

print

The print() function can be used to print a string. It takes one argument: the string you want to print. This argument cannot be missing.

println

println is the same as print(), except it also prints a newline after the string that is passed in.

dim

dim returns the length of an iterable object. If the given object is list, dim returns the length of the list. If the given object is a tensor, dim returns a int list indicating the shape of the tensor.

str_of_bool

This function takes a boolean and returns a string version of the boolean value by printing true if the boolean is true and false if the boolean is false.

str_of_float

This function takes a float and returns a string version of that float following the C language printing style achieved when using printf with %g in C.

str_of_int

This function takes an integer and returns a string version of the digits in the integer.

int_of_float

This function takes a float and returns an int by rounding the float to its nearest integer representation.

int_of_str

This function takes a string and returns a integer by converting the characters in the string into their corresponding digits. If the string provided doesn't represent a valid base-10 integer, this function returns -1.

float_of_int

This function takes an integer and returns the floating point representation of that integer.

3.6 Structures

3.6.1 List

A list is a container of primitive types, structs, tensors, functions, or other lists.

3.6.1.1 Declaring Lists

The declaration and definition of lists can happen in one statement or two. See next section below about how to do declare and define lists in one statement.
The type of the elements in an list is specified at the time of definition. The size of the list is not fixed and will grow dynamically. Lists are declared using the keyword List, followed by arrow brackets, which contain the type of the elements in the list, <type>, the list name, and a semicolon. The semicolon can be omitted if the list is being declared and defined in one line.

```
/* Declare a variable of type List<String>, automatically initialized as empty list*/
List<string> hello_list;
```

3.6.1.2 Defining Lists

You never define the size of the list when you declare an list. Lists are not fixed in size. To define an list, you simply define the elements of the list explicitly. The elements are put in square brackets and are separated by commas. There can be zero or more elements in the brackets (i.e. the brackets can be empty). The last element in the list of elements is not followed by a comma. The whole statement is terminated by a semicolon.

```
/* Declare an list and specify the elements in the list explicitly during definition all in one
List<string> hello_list = [“Hello”, “World”, “!”];
```

3.6.1.3 Lists in Lists

To declare an list inside an list, you have two options. You can either define the values of all of the lists explicitly using the bracket syntax explained in 4.2 Defining lists. Example:

```
List<List<int>> foo = [
[1,2,3,4],
[11,12,13,14]
];
```

3.6.1.4 Accessing List Elements

To access an element in an list, you use the variable name for that list followed by square brackets with an integer expression inside the square brackets as the index of the desired element. For nested

lists, square brackets with indices in them are concatenated to get to the element at the desired layer of the nested list. Accessing elements beyond the size of the list causes undefined behavior. All indices start at 0.

```
List<int> a = [34, 235, 2534, 435];
// a[2] == 2534 is true
List<List<int>> foo = [
   [1,2,3,4],
   [11,12,13,14]
];
// foo[0][2] == 3 is true
```

3.6.2 Matrix

A matrix is an object that is close to its mathetmatical equivalent.

3.6.2.1 Declaring Matrix

The declaration and definition of lists can happen in one statement or two. See next section below about how to do declare and define lists in one statement.
The type of the elements in a matrix is float. The size of the tensor is fixed at the time of definition. Matrix are declared using the keyword Matrix, followed by the matrix name, and a semicolon.

```
Matrix m = Matrix([1.0]);
```

3.6.2.2 Defining Matrix

You automatically define the size of the matrix when you define a matrix. Matrix are fixed in size. To define a matrix, you simply define the elements of the tensor explicitly. The elements are put in square brackets and are separated by comma. There can be zero or more elements in the brackets (i.e. the brackets can be empty). The whole statement is terminated by a semicolon.

3.6.2.3 Dimension of Matrix

A matrix must have a dimension of two, i.e row by column.

3.6.2.4 Accesing Matrix Elements

To access an element in a tensor, you use the variable name for that tensor followed by a square brackets with two integer expressions inside the square bracket separated by a comma. The first integer expression is the row index, and the second one is the column index. LOL checks the dimension automatically, so if you try to access the dimension that is out of range, it will throw an error before executing the code.

```
Matrix m = Matrix([1.0,2.0,3.0],[3.0,2.0,1.0]); // dimension is (2,3)
m[0,0] == 1.0; // true
```

3.6.2.5 Matrix functions

Below are a list of functions that you can use with matrix:

1. minit(): initialize a new matrix
2. m.get(i, j), mget(m, i, j): retrieve the ith-jth element from the matrix m
3. m.set(i, j), mset(m, i, j, x): set the ith-jth element in the matrix m
4. m1.add(m2), madd(m1, m2): add the elements of m2 to m1
5. m1.sub(m2), msub(m1, m2): subtract the elements of m2 from m1

6. mmulc(m, x): multiply matrix m by a constant x
7. maddc(m, x): add constant x to every element of matrix m
8. mmule(m1, m2): element-wise multiplication of m1 and m2
9. mdive(m1, m2): element-wise division of m1 and m2
10. mdot(r1, r2): take the dot product of two matrices with 1 row
11. mmul(m1, m2): matrix multiplication of m1 and m2
12. mexpe(m): apply exponential function to every element of m
13. mloge(m): apply logarithm function to every element of m
14. mswapr(m, i, j): swap the ith and jth rows in matrix m
15. mswapc(m, i, j): swap the ith and jth columns in matrix m
16. mtrans(m): transpose matrix m
17. mcopy(m1, m2): copy the content of m1 to m2
18. mgetr(m, i): get the ith row of a matrix
19. mgetc(m, i): get the ith column of a matrix
20. mgetsub(m, i1, j1, i2, j2): get the submatrix from i1th row to i2th row and from j1th column to j2th column.

4 Project Plan

4.1 Process used for Planning, Specification, and Development

For the LOL language, the team strictly follows the AGILE development methodology. The features were developed in 3-week sprints and we had weekly meetings to go over the progress on each other's plate. At the very beginning of the semester, we had a meeting which set the goal and the general design of the language. Our second meeting focused on the language reference manual and confirm the language syntax as well as a plan for language implementation. The subsequent meetings were mostly work meetings, when we met in person, went over the progress on each other's end, and solving programming challenges together. After a team member developed a feature, we would go over the feature together and make sure it works properly by running multiple tests. And when a team member makes a proposal for a new feature, we would sit down together and discuss the feasibility of the feature together as a group.

As for development, within our team, we formed two subteams of two to work on more specific parts of the language. For example, at one time we had a team for functional programming and linear algebra operations. Since both teams are developing simultaneously, we organized the repository in the following way: 1. we created a "dev" branch from the master, which contains the latest features from both team, but this is not necessary stable. 2. for each team that is tackling different features. we made different feature branches. 3. For each member who is working on the same feature, he/she creates a new personal branch and submits merge request for the team to review once a new feature is developed.

4.2 Programming Style Guide
(Modified from Java Style Guide: https://javaranch.com/style.jsp)

4.2.1 Formatting

- 1.1 - Indentation
    - All indents are four spaces. All indenting is done with spaces, not tabs.
    - Matching braces always line up vertically in the same column as their construct.
    - All if, while and for statements must use braces even if they control just one statement.
- 1.2 - Spacing
    - All method names should be immediately followed by a left parenthesis.
    - All List dereferences should be immediately followed by a left square bracket.
    - Binary operators should have a space on either side.
    - Unary operators should be immediately preceded or followed by their operand.
    - Commas and semicolons are always followed by whitespace.
    - The keywords if, while, for must be followed by a space.
- 1.4 - Maximum Line Length
    - Avoid making lines longer than 120 characters.
- 1.5 - Parentheses
    - Parentheses should be used in expressions not only to specify order of precedence, but also to help simplify the expression. When in doubt, parenthesize.

4.2.2 Identifiers

All identifiers use letters ('A' through 'Z' and 'a' through 'z') and numbers ('0' through '9') only. No underscores, dollar signs or non-ascii characters.

- 2.1 - Classes and Interfaces
    - All class and interface identifiers will use mixed case. The first letter of each word in the name will be uppercase, including the first letter of the name. All other letters will be in lowercase, except in the case of an acronym, which will be all upper case.
- 2.3 - All Other Identifiers
    - All other identifiers, including (but not limited to) fields, local variables, methods and parameters, will use the following naming convention. This includes identifiers for constants.
    - The first letter of each word in the name will be uppercase, except for the first letter of the name. All other letters will be in lowercase, except in the case of an embedded acronym, which will be all uppercase. Leading acronyms are all lower case.

4.2.3 Coding

- 3.1 - Constructs to Avoid
    - Never use return in the middle of a method.
- 3.2 - Do Not Compound Increment Or Decrement Operators
    - Use a separate line for an increment or decrement.
- 3.3 - Initialization
    - Declare variables as close as possible to where they are used.

4.3 Project Timeline

03/05: Language Reference Manual Done
03/25: Import template from HW2
03/30: Finish most of the scanner and parser
04/08: Confirm the use of GNU Scientific Library and add tests
04/16: Developed List class
04/20: Add List operations, developed First Class Functions
04/28: Add Matrix operations
05/08: Fix bugs and add more tests
05/14: Refine Matrix, List, and First Class Functions
05/15: Fix warnings and final check

4.4. Roles and Responsibilities

Jerry Lin: Manager, responsible for overseeing the overall progress, feature development, and communication with TA, focused on the development of the language syntax and the Matrix class

David Wan: System Architect, responsible for proposing implementation solutions and reviewing features produced by the team members; focused on the First Class Functions, the Matrix class, and language demos

Shuqi Chen: Chief Testing Officer (CTO), responsible for generating the test scripts for the language features and feature development; focused on the List class, the test suite, and the presentation

Zichuan Wang: Language Guru, responsible designing the language syntax, feature development, and feature review; focused on the List class, the LOL syntax, and LOL language demos

4.5 Software Development Environment

The team has both Mac and Windows machines, so the development was conducted in separate operating systems. To compile LOL, several dependencies are required, which can be found for both Mac and Windows. They are:

1. LLVM 7.0.0 or higher
2. OCaml 4.05.0 or higher
3. Clang
4. OPAM
5. GSL

4.6 Project Log

```
commit 53f6717b7486783f65fda096487add4ac1f24f35
Merge: 9e46997 83e2ce9
Author: Jerry Lin <shaokai_lin@126.com>
Date:   Thu May 16 11:10:19 2019 -0400

    Merge pull request #15 from Zichuan-Wang/dev

    Dev

commit 83e2ce9914c11fd97d5ee3dd7e60ed831414c019
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Thu May 16 10:37:29 2019 -0400

    more tests

commit affe30f92bac039c25ba0f299ded383393ec59e1
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Thu May 16 10:19:10 2019 -0400

    more test for matrix

commit 365362e0e4b80cdcb801419fb560ddb9d1b93bf7
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Thu May 16 09:16:33 2019 -0400

    more tests

commit 99072ed496a0c1d25d7b2d591f3c8b50e78c8a46
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Thu May 16 05:05:56 2019 -0400
```

Everything should work.

commit 6905e25472aad1747e8dd14edbac9c898c5cecd4
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Thu May 16 04:58:20 2019 -0400

Redo clean up.

commit 4cc45bc2eb46168c4bef4cb9b16a210e7102d791
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Thu May 16 04:49:51 2019 -0400

Partial fix

commit 2ed27f84dd222695514e841ae20f586f2e76c5ee
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Thu May 16 04:39:41 2019 -0400

Fixed naive

commit 18c8d7cbd04683fcc9a659c0ff6704aeab3b059c
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Thu May 16 04:39:07 2019 -0400

Maybe wrong

commit 9656a2947d29da05ad11dc4cf94847854284b604
Merge: bf85a3f ee6da79
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Thu May 16 03:56:16 2019 -0400

Merge branch 'dev' of https://github.com/Zichuan-Wang/LOL into dev

commit bf85a3f94912a487515dc4396e07480faa1b2cc9
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Thu May 16 03:56:09 2019 -0400

Conflict fix.

commit ee6da791a948651e0d6be299d87e524df3606626
Merge: 4e05fb0 14a4f7f
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Thu May 16 03:44:02 2019 -0400

More Clean up

```
commit 4e05fb00657648df079b4b25cf54affb14985b44
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Thu May 16 03:35:45 2019 -0400

    Clean up

commit 14a4f7f1a02ee5f05e51bb09f351ae39fd87ef64
Author: lsk567 <shaokai_lin@126.com>
Date:   Thu May 16 03:23:56 2019 -0400

    @ operator for MatrixMul

commit 66a9929de0dced5389ae5bb5e06f451b20acfa0c
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Thu May 16 02:31:04 2019 -0400

    Completed mlp_matrix

commit 85aca0326a6afc4bdc9822dbda5fd3e33a9bdad1
Merge: 0d9145a 488dad5
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Thu May 16 02:14:18 2019 -0400

    Merge branch 'dev' of https://github.com/Zichuan-Wang/LOL into dev

commit 0d9145ae221d87fac16095b527a89635a0bdb81b
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Thu May 16 02:14:07 2019 -0400

    Fixed Matrix static checking

commit 488dad5d33e7deb9f10977e9e6082bcad3aae9d9
Merge: ebe6518 337779c
Author: lsk567 <shaokai_lin@126.com>
Date:   Thu May 16 01:51:22 2019 -0400

    Merge branch 'dev' into gsl

commit ebe6518ec364a767cc6fb06a6710f2df7ebc1546
Merge: 2d51dd4 ceeedb9
Author: lsk567 <shaokai_lin@126.com>
Date:   Thu May 16 01:47:57 2019 -0400

    merge dev to gsl
```

```
commit 337779cecda611cfc14bd345284b2eb4540bc3c2
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Thu May 16 01:46:41 2019 -0400

    pow test

commit e43eb05904d40623c302cd4676e70300a9a4085d
Merge: ec3134c ceeedb9
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Thu May 16 01:44:34 2019 -0400

    Merge branch 'dev' of https://github.com/Zichuan-Wang/LOL into dev

commit ec3134c7f6c4c2f9ec5a68e0404b400f82fca75c
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Thu May 16 01:44:22 2019 -0400

    Fixed matrix check dim only takes in int

commit 2d51dd421b14b4f27421df3e655db6aec15b9d9b
Author: lsk567 <shaokai_lin@126.com>
Date:   Thu May 16 01:43:48 2019 -0400

    add max, det, mloge, mexpe

commit ceeedb988cd30bdbe9e6cfd2755a096bd0516cca
Merge: 26956b2 6f83a22
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Thu May 16 01:34:50 2019 -0400

    Merge branch 'dev' of https://github.com/Zichuan-Wang/LOL into dev

commit 6f83a227342cdb0e9432d840fa2f862d597b4dfb
Merge: 1e2012a 6ccdae4
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Thu May 16 01:33:09 2019 -0400

    Merge branch 'dev' of https://github.com/Zichuan-Wang/LOL into dev

commit 1e2012ac0e85ae2bab6ead67508babdd76f1d3c6
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Thu May 16 01:32:51 2019 -0400

    Added float pow

commit 26956b28b3ff2d347419ed59b12a26a759073f1d
```

```
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Thu May 16 01:32:40 2019 -0400

    more tests

commit 6ccdae4446f32f384245201b828df96c4e7e0a18
Merge: 261bd2e 21a3738
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Thu May 16 01:04:43 2019 -0400

    Merge branch 'dev' of https://github.com/Zichuan-Wang/LOL into dev

commit 261bd2e6028b9a2e4ee3920a73916c6e320b3f61
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Thu May 16 01:04:23 2019 -0400

    added more tests

commit 21a3738db097d5f4706335851213958aa98ed06c
Merge: 7fbd4c7 b98e2b3
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Thu May 16 00:39:27 2019 -0400

    Merge branch 'dev' of https://github.com/Zichuan-Wang/LOL into dev

commit 7fbd4c79cd0231070f2c2dfaac000209ade10c70
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Thu May 16 00:39:08 2019 -0400

    Fixed matrix access

commit b98e2b3253d15128729a63043f530fc02621bd5c
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Wed May 15 23:32:49 2019 -0400

    added more tests

commit fc658aa1a1e776da7c80057bbc1abf16143bd18a
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Wed May 15 23:13:19 2019 -0400

    added test for dot

commit 36a5b0f8782122ab61201510fc13ceca29ae7564
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Wed May 15 22:37:47 2019 -0400
```

```
    Fixed mrow and matrix

commit 9b20a0f97b2f8d6634c72e83552467c048d31613
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Wed May 15 22:23:18 2019 -0400

    added more test

commit e04a4c0dfb40f4c9455d8d0d3a027719ba2b440a
Author: lsk567 <shaokai_lin@126.com>
Date:   Wed May 15 21:26:59 2019 -0400

    add matrix multiplication

commit 3fb2fe814cdaa17020e4f62f329593420de4ed4b
Merge: cb341a8 df7e6fe
Author: lsk567 <shaokai_lin@126.com>
Date:   Wed May 15 21:19:13 2019 -0400

    merge gsl to dev

commit cb341a8b2a981dfb019c56bbfeef1f8ba2382433
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Wed May 15 20:44:48 2019 -0400

    Added demo, first demo completed

commit e9d3a3b6046c553131af07748f92a3f4568b3e54
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Wed May 15 20:42:57 2019 -0400

    Added row and col operation

commit df7e6fe557730d4b84c2dc6697823cbcb9b927ed
Author: lsk567 <shaokai_lin@126.com>
Date:   Wed May 15 17:17:13 2019 -0400

    add dot product

commit 1b1150aec1f1699768d254bc40bb8dbd741ef265
Author: lsk567 <shaokai_lin@126.com>
Date:   Wed May 15 16:00:59 2019 -0400

    update .gitignore, add lol programs
```

commit 10fc8bba9cf1e6abd648a1a278cdae2d9fd6b553
Author: lsk567 <shaokai_lin@126.com>
Date: Wed May 15 15:46:20 2019 -0400

add swap, transpose, and matrix view ops

commit 28b24fd6428aa165bdd48f4236d16e9297837dc5
Author: lsk567 <shaokai_lin@126.com>
Date: Wed May 15 12:30:26 2019 -0400

fix merge and testall bugs

commit b9c5ad75e2dd992f10dbdd4bf382eee850332ad5
Merge: a2151a6 1413dd2
Author: lsk567 <shaokai_lin@126.com>
Date: Wed May 15 12:11:04 2019 -0400

initial merge from dev to gsl

commit a2151a6e5e9653e61cf9f7eaa03fe9cb0040c077
Author: lsk567 <shaokai_lin@126.com>
Date: Wed May 15 09:04:26 2019 -0400

fix matrix dimension and add print matrix func

commit 6a41b33f0fa93778574cc6a83cb1f6f31c80abdd
Author: lsk567 <shaokai_lin@126.com>
Date: Wed May 15 08:32:50 2019 -0400

add basic matrix operations

commit 03ea0cb03c14820eb87300815c053f7eaf806dad
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Tue May 14 20:16:08 2019 -0400

Fixed internal checking for matrix dim

commit 9b301ae7367b1ebc2e3a7bf374d71f5e55859baf
Author: lsk567 <shaokai_lin@126.com>
Date: Tue May 14 19:46:42 2019 -0400

fix index out of bound bug

commit 9e469976ee8a253affb43745655c5915720ac7c5
Merge: fbfa356 489af3e
Author: David Wan <zerocstaker@users.noreply.github.com>

Date: Mon May 13 18:25:42 2019 -0400

  Merge pull request #13 from Zichuan-Wang/Zichuan-Wang-patch-1

  Update Language Reference Manual.md

commit b34b89cfa3e0ee2615c90002b6351a5a8b280845
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Mon May 13 18:22:02 2019 -0400

  Added Matrix internal row and col counter for static checking

commit 1413dd2a8cde78bb13c69c54ec8f062f8ac40568
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Sun May 12 16:04:02 2019 -0400

  Fixed tests.

commit 881bae22a21073e980bcf4779345da9958049b90
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Sun May 12 16:03:05 2019 -0400

  Fixed conflicts.

commit 98ee856f20d460bca26fdc231656841eca492d06
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Sun May 12 13:20:06 2019 -0400

  Small fixes.

commit f26a32afab039252c5e8ef046541172b141b976e
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Sat May 11 22:03:50 2019 -0400

  Fixed append bug, access is random

commit 0295f048eaccd8f607bdd9e27a43d50ecee6325f
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Sat May 11 20:22:13 2019 -0400

  Added initial matrix codegen, set elem says incorrect number of arguments

commit 77775afb0c4bdab96fa361122c34165cd39628ed
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Fri May 10 15:56:19 2019 -0400

```
    Added more test.

commit 489af3e9bf1336bb23d361de159ccc4eb9f55076
Author: Zichuan "Max" Wang <zichuan.wang@columbia.edu>
Date:   Fri May 10 15:48:15 2019 -0400

    Update Language Reference Manual.md

commit 1befee49a2823a45749f5162bd77028562b7789c
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Fri May 10 15:46:07 2019 -0400

    Fixed test case

commit 86168c7153a22486b389d4378e6dc30e7b709bc1
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Fri May 10 15:21:32 2019 -0400

    first class func bug needs to be fixed

commit 0c4acbadefa49635a216e81023e72cd451ce2ff7
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Fri May 10 15:01:06 2019 -0400

    added more tests

commit 321e06fb8f536888f116028633dd67e7e939f113
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Fri May 10 14:51:04 2019 -0400

    fixed float bug

commit 30d678f3d2ef23589f12f817c2b8d65186a9eba9
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Fri May 10 12:13:28 2019 -0400

    Added for(i=0;)

commit eb27c572e429b23675b9878f440f14da324c1f80
Merge: c59a123 cd94815
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Fri May 10 11:46:36 2019 -0400

    Merge pull request #12 from Zichuan-Wang/forwhilefix

    Forwhilefix
```

```
commit cd948156717b4e27b3b73d79a749266454dd64f7
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Wed May 8 00:18:29 2019 -0400

    Fixed for loop initialization problems and fixed test cases

commit ee2e1a30fb177baaa890732fc396cf59b9e9f03e
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Wed May 1 17:09:26 2019 -0400

    Fixed declaration for(int i =0;...;...){...}

commit c59a12362a26892f5f68dbf173e2e58e5f16bc16
Merge: 63b7ead 13e24bc
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Wed May 1 00:45:40 2019 -0400

    Merge pull request #11 from Zichuan-Wang/list-funcs

    List funcs

commit 13e24bce2166648b8d8af8627a740faf1e6697d3
Merge: 075bcfb 63b7ead
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Wed May 1 00:45:30 2019 -0400

    Merge branch 'dev' into list-funcs

commit 63b7eada7fb13969f3ced198477cced34d4e5353
Merge: 837b55f 29870ed
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Wed May 1 00:42:53 2019 -0400

    Merge pull request #10 from Zichuan-Wang/func_as_typ

    Func as typ

commit 29870ed24ef7dd7d8435ac2dca5c6d17f35b3494
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Wed May 1 00:41:55 2019 -0400

    Added func tests.

commit bc38e880f0d0cd6401c3beb5c705914221d2c1a7
Author: Zichuan Wang <zichuan.wang@columbia.edu>
```

Date: Wed May 1 00:34:35 2019 -0400

  Completed merging.

commit 075bcfb8d9ff7cc06500adfdaf82d5a4acfa65bc
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Tue Apr 30 23:25:33 2019 -0400

  Disallowed List<void>

commit 1ab947823dd08dc944d872597c81c152f128d1ec
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Tue Apr 30 22:42:14 2019 -0400

  Added more tests.

commit 3c9bca4dae7335a546044d5c1b63d0cdcb948e78
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Sun Apr 28 15:49:48 2019 -0400

  update matrix semantics, started codegen

commit 3acd482058a1f9323b2c08b3353b6e7e4f64ab53
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Sat Apr 27 23:15:40 2019 -0400

  add set & get operations

commit 6473540c2cb48f9199a64fb4a4664d0bbf93ee44
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Sat Apr 27 18:06:08 2019 -0400

  add matrix basic ops to ast, sast, parser, and sem

commit 489e606abdbb5cdd43ce7fdf35af24abd0a3774d
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Fri Apr 26 18:33:43 2019 -0400

  Add matrix basic operations

commit 4d3826152857b623d7af3636c37726f611c92156
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 25 11:12:46 2019 -0400

  fix asat formatting

commit 41aea364e06266077847b6083185a6af293051c8
Merge: 3797d3c 9b7cb92
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 25 09:38:21 2019 -0400

  Merge branch 'gsl' into gsl-new

commit 3797d3c9c97c289fee4425ec7443f766daef19c8
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 25 09:13:37 2019 -0400

  remove comments

commit 761f854704a7db4da5bf4e9a5b67706e421b429e
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 25 09:12:28 2019 -0400

  integrate gsl to new structure

commit e6c9d814c086eb76e30f1af49fb7ed7733d39959
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 25 09:00:06 2019 -0400

  git ignore compiled files

commit cea727746d21f400fef31658bb1ad7f1e236e3f3
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date: Tue Apr 23 23:17:28 2019 -0400

  added test for append

commit 696e403ec7ebe9a31927e6137d95a42ceb067b52
Merge: 920fdb5 837b55f
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Sun Apr 21 23:18:17 2019 -0400

  merge

commit 920fdb5bf35bb5bf1d24fc4b9dec09d42100872c
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Sun Apr 21 23:04:04 2019 -0400

  Fixed bug with func type as a param or return type. Right now only support defining specific func type, needs work with inference at codgen level with SABSTRACT

commit fbfa356f82f54e083405e30fbad8b7b7639c98bc

```
Merge: c8a7259 837b55f
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Sun Apr 21 11:01:00 2019 -0400

    Merge pull request #9 from Zichuan-Wang/dev

    Dev

commit 837b55f0dd50bee2a613ba43c904ffe215ff77aa
Merge: 01e658c 9451aef
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Sun Apr 21 01:24:33 2019 -0400

    Merge pull request #8 from Zichuan-Wang/list-funcs

    List funcs

commit 9451aeffb265bdff60de73f03c4a6cd6f4da1244
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Sun Apr 21 01:13:50 2019 -0400

    Fixed list in list

commit 6d8026ecf933dc513186888d943718560f8f4219
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Sat Apr 20 21:27:13 2019 -0400

    Added length for list

commit 01e658c3953cb9f49e5992f93affbcb5584769af
Merge: d743e48 94f3d4b
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Sat Apr 20 19:23:05 2019 -0400

    Merge pull request #7 from Zichuan-Wang/newstructure

    Newstructure

commit 94f3d4baf46dab6472e6e9c9f574b4136847620b
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Sat Apr 20 19:21:28 2019 -0400

    Added list append

commit 9b7cb920b78f114c93bb1f4249b8232926299532
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
```

Date: Sat Apr 20 14:36:34 2019 -0400

  add message

commit 524db240e3dc1a555f86cd4c18e30cad3f70150b
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Sat Apr 20 00:51:06 2019 -0400

  Fixed List Access

commit f550adfbf934de9bdd9d2f90884e93770c2be027
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Sat Apr 20 00:33:00 2019 -0400

  Add list tests

commit d34121065ff23f95053c3e622fb7c1d108cc94e6
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date: Sat Apr 20 00:30:25 2019 -0400

  Trying to fix bug

commit 5efa74eef08b2d37c23b0ffc1fc236e043eb3d9a
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Fri Apr 19 23:50:22 2019 -0400

  1. Ast, Sast pretty print overhaul
  2. Changed Declare to take expr instead of expr option, None should be changed to Noexpr
  3. Fixed readme unable to remove *.ll files

commit 486f7be2682c0b41d24d3b206373d81787ce4a85
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Fri Apr 19 15:26:35 2019 -0400

  Fixed closure

commit a2de3ea29c316ac96dd1e013788053d1b32160fc
Merge: 7ff7418 663a37b
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date: Fri Apr 19 12:16:49 2019 -0400

  Merge branch 'newstructure' of https://github.com/Zichuan-Wang/LOL into newstructure

commit 663a37be68e5b871f39b646de3ac474169572219
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Fri Apr 19 11:04:59 2019 -0400

Cleaned up ast and sast

commit 0844f1598ca102755ec73701c47655358b329524
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Fri Apr 19 10:18:43 2019 -0400

Added types without initializing

commit 7ff741824b4b966961e97c89ffb69de8f2e87c9e
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date: Fri Apr 19 00:37:38 2019 -0400

added SListAccess, still need to debug

commit 80353a7c4c6a0c82d5ca08184d1a0ce40a989702
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Thu Apr 18 23:12:59 2019 -0400

Fixed expected top-level entity bug

commit 1b051c8c3af6c4a183d4328a7b38fc089c95abcc
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Thu Apr 18 22:03:30 2019 -0400

Fixed and cleaned up Makefile and gitignore

commit fcfbbafd6e35ff3e2f6510d78d868092f4209ffd
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 18 09:13:44 2019 -0400

update gitignore and add program folder

commit db3588493991b3ce410a2c34ddee8087400c02c1
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 18 09:04:18 2019 -0400

update gitignore

commit b6541737a8ccf1d4baba5fbc37a8fe7fadadd826
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 18 08:48:55 2019 -0400

updated gitignore

commit fbbba59d9111f9fd281bef6369b67a486b859ba1

Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Thu Apr 18 08:44:19 2019 -0400

    added print program and update makefile

commit 6f273e7ac6a05366ba051b8f49da90719afaab81
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Wed Apr 17 23:43:28 2019 -0400

    set new line back to string

commit d791cae082f129c45ede7c3ff8d791886fce178b
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Wed Apr 17 21:56:38 2019 -0400

    added gitignore

commit bad0a9d0adf3cdc300d9e89259f0b0a4b0480eb8
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date: Wed Apr 17 15:31:26 2019 -0400

    added list_append declaration

commit c385b487a2541324de67a73ddb8f398ca80da680
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Wed Apr 17 02:39:58 2019 -0400

    Fixed module error

commit f57a5ed8a31dcda7f89784e9d33dd88ce91d2cbb
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Wed Apr 17 01:09:58 2019 -0400

    Fixed error when initializing list with only []

commit 00fc4bf2ecef0be7007698d2caff7825cf3392da
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Wed Apr 17 00:59:30 2019 -0400

    Fixed function expression return error

commit 2a1d208c53756a6e041109facafe329ca9e775ac
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Tue Apr 16 00:14:53 2019 -0400

Fixed some bugs for list, still need to fix the bug of Stored value type does not match pointer operand type!

commit 7c3ecaa78d5af5f949f66bd7cd284e6c922d033d
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Mon Apr 15 22:29:37 2019 -0400

Changed framework, easier to understand, need semantic checking, ++ -- not working need more tests

commit efb459202811a44fa11307c312610ac6b50294b1
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Fri Apr 12 23:12:40 2019 -0400

Added power symbol

commit 11e63e94bc700ac48b706ead0a438d83591c0f8f
Author: zerocstaker <meetdavidwan@gmail.com>
Date: Fri Apr 12 22:58:41 2019 -0400

Added special symbols

commit d743e481921e7d91636a3bfe0c9db481240f1dcd
Merge: 4d7f42c c8a7259
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date: Thu Apr 11 02:05:41 2019 -0400

Merge pull request #5 from Zichuan-Wang/master

Merge pull request #4 from Zichuan-Wang/dev

commit 18fc258569991614d6b7dd7fe2278e3e01a8a7a0
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Tue Apr 9 22:32:56 2019 -0400

added newline function, not working yet

commit 49a492e19c6123220b790d790e244c1e2c7ecc9b
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Tue Apr 9 20:57:28 2019 -0400

add helper functions to debug void param error

commit f61236e5aab16635235a1d5a89b9388ce962ae99
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date: Tue Apr 9 12:48:01 2019 -0400

```
    link gsl during compilation, add gsl_test function

commit f71632f627b23603da3bd08236281f7e00869e95
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date:   Tue Apr 9 12:47:07 2019 -0400

    refactor existing lol programs, add gsl_test

commit 11b86e06964075a71a62c54f6455ffcc7f99c12d
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date:   Tue Apr 9 11:55:17 2019 -0400

    remove gsl in the source, will add later as dep

commit 3d742ed28cbff4d6e1bf94c1d359afca0344ebea
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date:   Mon Apr 8 22:14:13 2019 -0400

    added gitignore

commit b7838c65b6229b073aca8e8f90a01d9edc84629b
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date:   Mon Apr 8 22:04:47 2019 -0400

    added gsl library

commit 049d18147726309670dd1a4834216b87df6ef72d
Author: git config --global user.email "linkaikai28@126.com <shaokai_lin@126.com>
Date:   Mon Apr 8 21:56:55 2019 -0400

    added tests

commit c8a7259085678c2d85a7532b41221fb83f344026
Merge: 8fdc934 4d7f42c
Author: David Wan <zerocstaker@users.noreply.github.com>
Date:   Mon Apr 8 15:01:25 2019 -0400

    Merge pull request #4 from Zichuan-Wang/dev

    Dev

commit 4d7f42c4b124f512f5ca7f02a6f582aa590d124e
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Mon Apr 8 02:11:02 2019 -0400
```

```
    Update README.md

commit 0be4d685874d3567413324045370249525 89dd24
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Mon Apr 8 02:09:41 2019 -0400

    Update README.md

commit a7778440e0f3d29288d1023365574ca5ef46fc68
Merge: e3e88d6 12547d2
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Mon Apr 8 02:07:54 2019 -0400

    Merge pull request #3 from Zichuan-Wang/test

    Added testing infrastructure.

commit 12547d2eac96b8f759e3f9602d2b958ab03a7244
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Mon Apr 8 02:04:52 2019 -0400

    Added new tests.

commit 6efdc3c9cc81dddf8a051329c52b8e25a924f1ee
Author: Zichuan Wang <zichuan.wang@columbia.edu>
Date:   Mon Apr 8 01:28:15 2019 -0400

    Added testing script.

commit 8fdc934f4d79f6a66fc09449ab9078241d0a230a
Merge: fe3fac1 e3e88d6
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Sun Apr 7 23:04:31 2019 -0400

    Merge pull request #2 from Zichuan-Wang/dev

    Hello world milestone

commit e3e88d61f032a9b1a1da879ca83da498ed0fb51c
Merge: fe3fac1 0ea22c1
Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date:   Sun Apr 7 22:32:35 2019 -0400

    Merge pull request #1 from Zichuan-Wang/David

    Hello world milestone
```

```
commit 0ea22c12971b33f1d52f5a346f58c88c18614915
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Mon Apr 1 16:31:29 2019 -0400

    Adds tags and a test file that prints hello world

commit b6febdbfde6581faf0df292d44a2004a75144b01
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Mon Apr 1 16:12:05 2019 -0400

    Fixed bug with function call

commit 7885df12e5878b07bd9f519af4c186eaea4e2f0b
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Mon Apr 1 14:56:16 2019 -0400

    Added how to run program

commit ef093822afe9a1ee994d05546a3edf32c2d5813a
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Sun Mar 31 23:03:38 2019 -0400

    Formatted for clarity

commit 172209b39084af095d801ce809fb19267d2974d2
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Sun Mar 31 02:56:43 2019 -0400

    Finished basic, has bugs when trying out, fixing later

commit 7333c8345e83717dc9b043bb7f4a6ce9d9ed2a43
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Sun Mar 31 00:52:48 2019 -0400

    Completed senment and fixed bugs in previous files, TODO Codegen

commit f2cfd7a746f7cbe8ac328a58b52d7dbf46c610a4
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Sat Mar 30 09:58:54 2019 -0400

    Covered basic scanner parser ast and sast, need to implement codegen and sement

commit fe3fac1b999d972bf1270280612a28e58b882846
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Wed Mar 27 14:00:57 2019 -0400
```

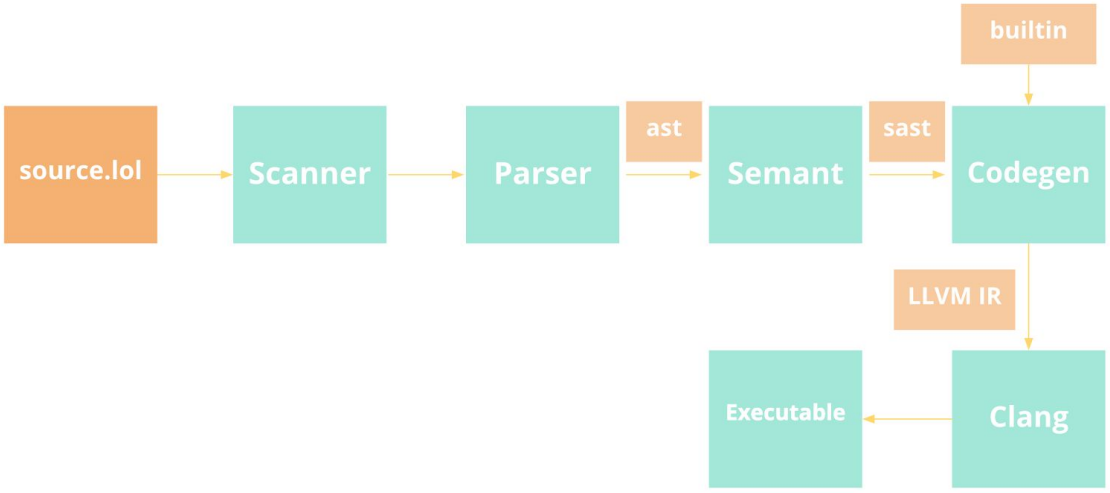
```
    implemented parenthesis

commit c5988794a5276446692321c24eefb551336f66aa
Author: Shuqi Chen <shuqichen0208@gmail.com>
Date:   Tue Mar 26 22:10:21 2019 -0400

    implemented single&multi comment

commit 5ffc4e4e68f510df7de5e403bc5c68db08a55454
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Mon Mar 25 22:06:55 2019 -0400

    Added Makefile and some dummy test files, need to add make test later

commit 557454f14ea118bff021e21249f075b55a0dd996
Author: David Wan <zerocstaker@users.noreply.github.com>
Date:   Mon Mar 25 21:47:41 2019 -0400

    Rename calc.ml to lol.ml

commit 40f837d2be60baf892519e0802cc0add6d010715
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Mon Mar 25 21:28:17 2019 -0400

    Added template from hw2

commit a1f806d250100310898c1f58892f56ed34d089ff
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Wed Mar 6 10:03:44 2019 -0500

    Final additions

commit 0149b11dacaf400573143023d939dbdf4ae44944
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Wed Mar 6 08:01:36 2019 -0500

    Some minor fixes

commit 79761c4e33ebfdd63375f0e10b758e97d66c48b9
Author: zerocstaker <meetdavidwan@gmail.com>
Date:   Tue Mar 5 14:28:17 2019 -0500

    Initial LRM

commit d4a3c520bc1ad0b6541d4a01dda6510f224da555
```

Author: Zichuan "Max" Wang <wang.zi.chuan@hotmail.com>
Date: Tue Feb 26 21:31:48 2019 -0500

Initial commit

5 Architecture Design

5.1 Block Diagram



5.2 Scanner

File: *scanner.mll*

The scanner takes in the program file and "tokenizes" it into literals, identifiers and keywords. All of the white spaces are taken out and tokens are generated for anything that has syntactic meaning in the language. This includes all of keywords, reserved words, variable names, braces, brackets and string, integer, and float literals. The scanner throws an error for unimplemented LOL keywords and syntactically invalid identifiers or literals.

5.3 Parser

File: *parser.mly, ast.ml*

The parser is written in Ocamlyacc. It is responsible for using the stream of tokens out of the scanner to construct an abstract syntax tree. The parser uses the ast.ml to generate an Abstract Syntax Tree. The primary parser follows microC closely.

5.4 Semantic Checker

File: *semant.ml, sast.ml*

The semantic checker performs a depth-first walk over the AST that was generated by the parser, verifying the semantics of the program, and returns a syntactically checked AST with partial type inference and syntax checking. The semantic checking includes a simple type inference system which assigns types to primitives and then propagates these types up the ast to higher level expressions.

5.5 Code Generation

File: *codegen.ml*

The codegen takes in the semantically checked SAST and builds the LLVM.

5.6 Division of Labor

Our primary framework follows micro-C quite closely. David took the responsibility to inherit the micro-C, modify it into a LOL-basic version, laying a reliable foundation for the actual implementations of LOL features. Then each member implementing the exact LOL feature would modify and extend scanner/parser/semant/codegen accordingly. Basically, David led the architecture design, and all team members were responsible for contributing to the whole architecture.

6 Test Plan

Individual features and aspects of LOL were developed in individual branches, including list branch, gsl branch, and many temporary branches . Before being merged into dev branch and then master branch, these features were tested using simple test programs written by the team member/members who were working on that feature. There is one or more tests for each feature, making sure that the newly implemented feature works as expected without breaking main functionality. All team members were responsible for contributing to testing development. In the wrapping-up period, Shuqi added more test cases to test corner cases.

Our testing is consist of both success and fail test cases. There are 70 unit test cases, and 3 comprehensive demo tests.

6.1 Testing Suite and Automation

All unit tests are stored in the /test/ folder. Sucess test cases have test-*.lol, and have a file with the expected output, *.out. Fail cases follow a similar format of fail-*.lol, and have a file with the expected output, *.err.

Testing automation is based off the Micro-C testing suite provided. The testall.sh script compiles and runs all of the *.lol files in the /test folder and compares the output to the corresponding *.out and *.err file respectively. The script is run when LOL is compiled using make.

6.2 Example Input-Output

Example1 test-func-simple-first-class
test:

```
func func(int,int:int) apply_f(func(int:int) f) {
    return func int (int i, int j) {
        return f(i) + f(j);
    };
}

func int double(int x) {
    return x * 2;
}

func(int,int:int) sum_of_double = apply_f(double);

println(str_of_int(sum_of_double(3, 5)));

```

output:

```
16
```

Example2 test-for-loop-missing-all

test:

```
func string infinite(int c) {
    for ( ; ; ) {
        c = c + 1;
        if (c > 100) {
            return "done!";
        }
    }

    return "what??";
}

string s = infinite(2);
println(s);
```

output:

```
done!
```

More test cases are attached in Appendix 8.2.

7 Lessons Learned

7.1 Jerry Lin

Through this project, I have gained many insights into how a modern programming language works, which I think is immensely helpful for my CS career in the future. In this course, there are several chapters that I enjoyed the most. The first one is parser. Prior to taking this class, I had very little idea about how a compiler understands the structure of the string input. Now a know that by leveraging Abstract Syntax Trees and LR(0) Automaton, we can build an effective parser. The other lesson I learned is LLVM. It is always interesting to me to see how things work from a low level perspective. It is fascinating to me that OCaml code and C code can work with each other on the LLVM level and create powerful functionalities. Lastly, I especially enjoyed this course since it provided a unique opportunity to think from the perspective of a programming language creator. As a developer, I have become accustomed to learning new languages but seldom think about why those languages are designed as they are. This course certainly makes me appreciate the features of different programming languages more, which I have taken for granted before.

7.2 David Wan

By implementing our own language, I think I learned a lot of about the internal workings of the most simple programming expressions, which programming languages often hide away the details from the users. I now can appreciate how a simple expression takes so many steps to be converted to TSA or assembly code that Clang can then understand. And during this conversion, there is so many edge cases that could cause a program during semantic checking. It is an really interesting idea that you write code for machine to understand your code in a language that the computer needs to understand. For future teams, I would definitely start early to get a basic infrastructure complete (like fully understand and implement MicroC), and from there building additional functionalities can be easy. Also I would suggest to test the features throughout, even after passing tests due to new features potentially breaking your code.

7.3 Shuqi Chen

7.3.1 Start early

One of the most important lessons I learn from this project is to start early. We started the actual implementation in early March, and it took us almost three months to finally finish implementing LOL language. The implementation were much more complicated and time-costing than we expected. There were always unexpected problems that were hard to debug. Some features, though seemed easy to implement, took us pretty much time.

7.3.2 Fully testing while development

In the beginning, we only wrote simple test cases for basic features which we thought were essential, leaving “unessential” features untested. As more advanced features added to LOL, we found many bugs in previous implementation which prevented the new features from normal function. For example, when implementing List, we focused on append. For the List access functionality, we only wrote naive test case. At that time, we did not realize the importance of List access. The problem appeared later when we added new features and ran a comprehensive test that required to do List access and assignment. Our previous implementation of List access was uncompleted, but our test case was too simple to uncover this problem.

7.4 Zichuan Wang

For the technical part the most important thing I learned is how to define the syntax and semantics for a programming language. Although it might not be the case that I would create a new programming language and define those myself, the project really helps me understand existing languages better.
For me the most important non technical thing I learned is the importance of frequent meetings in teamwork. For the majority of time our team does not have a chance where every team member can sit and code together, so meetings become the most important way for us to coordinate work so that people can know the progress of each other, or otherwise all kinds of problems can arise from lack of communication.
For future teams I suggest they meet and code together in person weekly if possible, otherwise weekly Skype meetings and some real time collaborative programming using tools like Visual Studio Code Live Share are recommended.

8 Appendix

8.1 src Files

ast.ml
Authors: David, Jerry, Shuqi, Zichuan

```

module StringMap = Map.Make(String)

(* types *)
type typ =
    Int
  | Float
  | Bool
  | String
  | Void
  | Func of func_typ
  | List of typ
  (* Linalg *)
  (* | Matrix of expr * expr *)
  | Matrix
  | Abstract

and func_typ = {
  param_typs: typ list;
  return_typ: typ;
}

(* operations *)
and op = Add | Sub | Mul | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or | Mod | Pow | Outer | NoOp

type uop = Neg | Not

and bind = typ * string

```

```
(* expressions *)
type expr =
    IntLit of int
  | FloatLit of string
    | BoolLit of bool
  | StrLit of string
    | Id of string (* var name *)
  | Binop of expr * op * expr
    | Unop of uop * expr
  | Assign of expr * op * expr
    | Call of expr * expr list
  | ListLit of expr list
  | ListAccess of expr * expr
  | ListAppend of expr * expr
  | ListLength of expr
  | FExpr of fexpr
  | Noexpr
  (* Linalg *)
  (* MatrixLit(ListLit(ListLit(FloatLit("1.0"), FloatLit("2.0")), ListLit(
  FloatLit("3.0"), FloatLit("4.0")))) *)
  (* Check matrix rank and row/column count in semantics *)
  (* Matrix Operations *)
  | MatrixLit of expr list
  | MatrixSet of expr * expr * expr * expr (* MatrixSet(Id(m), i, j, x) *)
  | MatrixGet of expr * expr * expr (* MatrixGet(Id(m), i, j) *)
  | MatrixAdd of expr * expr
  | MatrixSub of expr * expr
  | MatrixMulC of expr * expr
  | MatrixAddC of expr * expr
  | MatrixMulE of expr * expr
```

```
  | MatrixMulE of expr * expr
  | MatrixDivE of expr * expr
  | MatrixRow of expr
  | MatrixCol of expr
  | MatrixMul of expr * expr


and fexpr = {
    typ : typ;
    params : bind list;
    body : stmt list;
  }

(* Statements*)
and stmt =
    Block of stmt list
    | Expr of expr
  | Decl of typ * string * expr
    | Return of expr
  | If of expr * stmt * stmt
  | For of stmt * expr * expr * stmt
  | While of expr * stmt
  | Nostmt

type program = stmt list

(* pretty-printing, should return the same code *)
let string_of_op = function

    Add -> "+"
  | Sub -> "-"
```

```
  | Mul -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Pow -> "^"
  | Outer -> "@"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | NoOp -> ""

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

(* map each element in list by function f, and join the string by s *)
let rec string_of_list_stmt l s = String.concat s (List.map string_of_stmt l)
and string_of_list_typ l s = String.concat s (List.map string_of_typ l)
and string_of_list_expr l s = String.concat s (List.map string_of_expr l)
and string_of_list_bind f l s = String.concat s (List.map f l)

and string_of_typ = function
    Void -> "void"
  | Int -> "int"
  | Float -> "float"
  | Bool -> "bool"
```

```
  | String -> "string"
  | Func f -> "func " ^ string_of_typ f.return_typ ^ " (" ^ string_of_list_typ
  f.param_typs ", "  ^ ")"
  | List typ -> "list <" ^ string_of_typ typ ^ ">"
  | Abstract -> "abstract"
  | Matrix -> "Matrix"

and string_of_expr = function
    IntLit(l) -> string_of_int l
  | FloatLit(l) -> l
  | StrLit(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | Binop(e1, o, e2) -> string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^
  string_of_expr e2
  | Unop(uo, e) -> string_of_uop uo ^ " " ^ string_of_expr e
  | Assign(e1, o, e2) -> string_of_expr e1 ^ string_of_op o ^ "= " ^
  string_of_expr e2
  | Call(e, e_list) -> string_of_expr e ^ "(" ^ string_of_list_expr e_list
  ", "  ^ ")"
  (* below actually is parsed with {name = e.name; param = e.params;
   * typ = e.typ; body = e.body}. See test programs for examples. *)
  | FExpr(fexpr) -> string_of_fexpr fexpr
  | Noexpr -> ""
  (* List *)
  | ListLit(expr_list) -> "[" ^ string_of_list_expr expr_list ", " ^"]"
  | ListAccess(e1,e2) -> string_of_expr e1 ^ "[" ^ (string_of_expr e2) ^ "]"
  | ListAppend(e1,e2) -> string_of_expr e1 ^ "Append[" ^ (string_of_expr e2) ^
  "]"
```

```
  | ListLength(e) -> "len(" ^ string_of_expr e ^ ")"
  (* Matrix *)
  | MatrixLit(expr_list) -> "Matrix( " ^ string_of_list_expr expr_list ", "
  ^ " )"
  | MatrixSet(m, i, j, x) -> "MatrixSet( " ^ string_of_expr m ^ ", " ^
  string_of_expr i ^ ", " ^ string_of_expr j ^ ", " ^ string_of_expr x ^ " )"
  | MatrixGet(m, i, j) -> "MatrixGet( " ^ string_of_expr m ^ ", " ^
  string_of_expr i ^ ", " ^ string_of_expr j ^ " )"
  | MatrixAdd(m1, m2) -> "MatrixAdd( " ^ string_of_expr m1 ^ ", " ^
  string_of_expr m2 ^ " )"
  | MatrixSub (m1, m2) -> "MatrixSub( " ^ string_of_expr m1 ^ ", " ^
  string_of_expr m2 ^ " )"
  | MatrixMulC (m, x) -> "MatrixMulC( " ^ string_of_expr m ^ ", " ^
  string_of_expr x ^ " )"
  | MatrixAddC (m, x) -> "MatrixAddC( " ^ string_of_expr m ^ ", " ^
  string_of_expr x ^ " )"
  | MatrixMulE (m1, m2) -> "MatrixMulE( " ^ string_of_expr m1 ^ ", " ^
  string_of_expr m2 ^ " )"
  | MatrixDivE (m1, m2) -> "MatrixDivE( " ^ string_of_expr m1 ^ ", " ^
  string_of_expr m2 ^ " )"
  | MatrixMul (m1, m2) -> "MatrixMul( " ^ string_of_expr m1 ^ ", " ^
  string_of_expr m2 ^ " )"
  | _ -> "string_of_expr not exhausive"

and string_of_fexpr fexpr =
  let string_of_param param = let (typ, s) = param
    in string_of_typ typ ^ " " ^ s
  in
  "func " ^ string_of_typ fexpr.typ ^ " (" ^ string_of_list_bind
  string_of_param fexpr.params ", " ^")"
```

```
  ^ "{\n" ^ string_of_list_stmt fexpr.body "" ^ "}\n"

and string_of_stmt = function
    Block(stmts) -> "{\n" ^ string_of_list_stmt stmts "" ^ "}\n"
  | Expr(e) -> string_of_expr e ^ ";\n";
  | Return(e) -> "return " ^ string_of_expr e ^ ";\n"
  | Decl (t, s, e) ->
    string_of_typ t ^ " " ^ s ^ " = " ^ string_of_expr e ^ ";\n"
  | For (init, e2, e3, s) -> "for (" ^ string_of_stmt init ^ " ; "
  ^ string_of_expr e2 ^ " ; "
    ^ string_of_expr e3 ^ ") " ^ string_of_stmt s ^"\n"
  | While (e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | If(e, s1, fL) -> let then_part = match fL with
        Block([]) -> ""
      | If(_) as s2 -> "elif\n" ^ string_of_stmt s2
      | s2 -> "else\n" ^ string_of_stmt s2
    in
    "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s1 ^ then_part
  | Nostmt -> ""

and string_of_program stmts =
  string_of_list_stmt stmts "" ^ "\n"

```

scanner.mll
Authors: David, Jerry, Shuqi, Zichuan

```
(* Ocamllex scanner for LOL *)

{open Parser}

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

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
(* Comments *)
| "//"                 { linec lexbuf } (* Single-line  Comments *)
| "/*"                 { comment 0 lexbuf} (* Multi-line  Comments *)
(* Member func *)
| ".append"            { DAPPEND }
| ".length"            { DLENGTH }
| ".col"               { DCOL }
| ".row"               { DROW }
| ".set"               { DSET }
| ".get"               { DGET }
| ".add"               { DADD }
| ".sub"               { DSUB }
| ".mulc"              { DMULC }
| ".addc"              { DADDC }
| ".mule"              { DMULE }
| ".dive"              { DDIVE }

(* Symbols *)
| '('                  { LPAREN }
| ')'                  { RPAREN }
| '{'                  { LBRACE }
```

```
| '}'                   { RBRACE }
| '['                   { LSQBRACE }
| ']'                   { RSQBRACE }
| ':'                   { COLON }
| ';'                   { SEMI }
| ','                   { COMMA }
| '.'                   { DOT }
| '"'                   { QUOTE }
| '='                   { ASSIGN }
(* Math *)
| '+'                   { PLUS }
| '-'                   { MINUS }
| '*'                   { TIMES }
| '^'                   { POW }
| '/'                   { DIVIDE }
| '%'                   { MOD }
| '@'                   { MMUL }
| "+="                  { PLUSASN }
| "-="                  { MINUSASN }
| "*="                  { TIMESASN }
| "/="                  { DIVIDEASN }
| "%="                  { MODASN }
| "++"                  { INC }
| "--"                  { DEC }
(* Bool Op *)
| "=="                  { EQ }
| "!="                  { NEQ }
| '<'                   { LT }
| "<="                  { LEQ }
| ">"                   { GT }
```

```
| ">="                  { GEQ }
| "!"                   { NOT }
(* Keywords *)
| "if"                  { IF }
| "elif"                { ELIF }
| "else"                { ELSE }
| "for"                 { FOR }
| "in"                  { IN }
| "while"               { WHILE }
| "func"                { FUNC }
| "return"              { RETURN }
| "int"                 { INT }
| "bool"                { BOOL }
| "float"               { FLOAT }
| "string"              { STRING }
| "void"                { VOID }
| "AND"                 { AND }
| "OR"                  { OR }
| "List"                { LIST }
| "Matrix"              { MATRIX }
| "Tensor"              { TENSOR }
| "true"|"false" as lxm           { BOOLLIT(bool_of_string lxm)  }
| digits as lxm                   { INTLIT(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLOATLIT(lxm) }
| '"' ([^'"']* as lxm) '"'        { STRLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm     { ID(lxm) }
| eof                             { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment level =
```

```
  parse "*/" { if level == 0 then token lexbuf else comment (level - 1) lexbuf}
      | "/*" { comment (level + 1) lexbuf }
      | _ { comment level lexbuf }

and linec =
  parse '\n' { token lexbuf }
      | eof { token lexbuf }
      | _ { linec lexbuf }

```

parser.mly
Authors: David, Jerry, Shuqi, Zichuan

```
%{
  open Ast
%}

%token DAPPEND DLENGTH DGET DSET DADD DSUB DMULC DADDC DMULE DDIVE DROW DCOL
%token SEMI LPAREN RPAREN LBRACE RBRACE LSQBRACE RSQBRACE
%token LIST TENSOR MATRIX
%token QUOTE COLON COMMA DOT
%token PLUS MINUS TIMES DIVIDE MOD MMUL POW ASSIGN
%token PLUSASN MINUSASN TIMESASN DIVIDEASN MODASN INC DEC
%token EQ NEQ LT GT LEQ GEQ AND OR NOT
%token IF ELIF ELSE FOR IN WHILE
%token FUNC RETURN
%token INT BOOL FLOAT STRING VOID
%token <int> INTLIT
%token <bool> BOOLLIT
%token <string> ID STRLIT FLOATLIT
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%nonassoc ELIF
%left COMMA
%left COLON
%right PLUSASN MINUSASN TIMESASN DIVIDEASN MODASN ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS NEG
```

```
%left TIMES DIVIDE MOD DOT MMUL
%left POW
%right NOT

%right INC DEC
%left DAPPEND DLENGTH DROW DCOL

%start program
%type <Ast.program> program

%%

program: stmt_list EOF { List.rev $1 }

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

stmt:
    expr SEMI                                { Expr($1)              }
    /* x = 3; */
  | LBRACE stmt_list RBRACE                  { Block(List.rev $2)    }
  /* Declarations */
  | typ ID SEMI                              { Decl ($1,$2,Noexpr) }
  /* Initialize variables. Can have default init. */
  | typ ID ASSIGN expr SEMI                  { Decl ($1,$2,$4) }
  | fun_decl                                 { $1 }
  /* Function Declaration */
  | RETURN expr_opt SEMI                     { Return($2) }
```

```
  /*  Return a value */
  /* Control Flows */
  | IF LPAREN expr RPAREN stmt false_branch
                                          { If($3, $5, $6) }
                                          /* If, else, elif stuff */
  | FOR LPAREN init_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
                                          { For($3, $5, $7, $9) }
                                          /* For loop; for (;;) */
  | WHILE LPAREN expr RPAREN stmt          { While($3, $5) }
  /* While loop, can be treated as a for loop */

fun_decl: /* fun int add (int i, int j) { i = i+1-1; return i+j; } */
  FUNC ret_typ ID LPAREN params_opt RPAREN LBRACE stmt_list RBRACE
    { Decl( Func({param_typs = List.map (fun (ty, _) -> ty) $5;
        return_typ = $2 }),
    $3,
    FExpr({ typ = $2; params = $5; body = List.rev $8 }))}

/* if stuff */
false_branch:
    elif { $1 }
  | cf_else { $1 }
  | %prec NOELSE { Block([]) }

elif:
  ELIF LPAREN expr RPAREN stmt false_branch { If($3, $5, $6) }

cf_else:
  ELSE stmt { $2 }

```

```
/* Expressions */
expr:
  /* Accesors */
    accessor  {$1}
  /* Arithmeics */
  | MINUS expr %prec NOT { Unop(Neg, $2)      }
  | NOT expr             { Unop(Not, $2)      }
  | expr PLUS   expr { Binop($1, Add, $3)     }
  | expr MINUS  expr { Binop($1, Sub, $3)     }
  | expr TIMES  expr { Binop($1, Mul, $3)     }
  | expr DIVIDE expr { Binop($1, Div, $3)     }
  | expr MOD    expr   { Binop($1, Mod,   $3) }
  | expr POW    expr   { Binop($1, Pow,   $3) }
  | expr MMUL   expr { MatrixMul($1, $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)   }

  /* Function expression*/
  | function_expr { $1 }

  /* Assignmnent */
  | ID ASSIGN    expr { Assign(Id($1), NoOp,$3) }
  | ID PLUSASN   expr { Assign(Id($1), Add, $3) }
  | ID MINUSASN  expr { Assign(Id($1), Sub, $3) }
```

```
121     | ID TIMESASN  expr { Assign(Id($1), Mul, $3)}
122     | ID DIVIDEASN expr { Assign(Id($1), Div, $3) }
123     | ID MODASN    expr { Assign(Id($1), Mod, $3) }
124     | ID INC            { Assign(Id($1), Add, IntLit(1)) }
125     | ID DEC            { Assign(Id($1), Sub, IntLit(1)) }
126
127     /*List*/
128     | LSQBRACE opt_items RSQBRACE { ListLit($2) }
129     | accessor2 ASSIGN expr    { Assign($1, NoOp, $3) }
130     | accessor2 PLUSASN expr   { Assign($1, Add, $3) }
131     | accessor2 MINUSASN expr  { Assign($1, Sub, $3) }
132     | accessor2 TIMESASN expr  { Assign($1, Mul, $3) }
133     | accessor2 DIVIDEASN expr { Assign($1, Div, $3) }
134     | accessor2 MODASN expr    { Assign($1, Mod, $3) }
135
136     /* Brackets for precedence */
137     | LPAREN expr RPAREN   { $2 }
138
139     /* Matrix */
140     | MATRIX LPAREN opt_items RPAREN                        { MatrixLit($3) }
141     | accessor DGET LPAREN expr COMMA expr RPAREN
142     { MatrixGet($1, $4, $6) } /* t.get(1,2) */
143     | accessor DSET LPAREN expr COMMA expr COMMA expr RPAREN
144     { MatrixSet($1, $4, $6, $8) }
145     /* Add, sub, mul, div */
146     | accessor DADD LPAREN expr RPAREN             { MatrixAdd($1, $4) }
147     | accessor DSUB LPAREN expr RPAREN             { MatrixSub($1, $4) }
148     | accessor DMULC LPAREN expr RPAREN            { MatrixMulC($1, $4) }
149     | accessor DADDC LPAREN expr RPAREN            { MatrixAddC($1, $4) }
150     | accessor DMULE LPAREN expr RPAREN            { MatrixMulE($1, $4) }
```

```
  | accessor DDIVE LPAREN expr RPAREN           { MatrixDivE($1, $4) }

/* Accesors, helpful for recursive case */
accessor:
    accessor LPAREN args_opt RPAREN { Call($1, $3)  } /* fun(x) */
  | accessor DAPPEND LPAREN expr RPAREN { ListAppend($1, $4) }
  | accessor DLENGTH LPAREN RPAREN { ListLength($1) }
  | accessor LSQBRACE expr RSQBRACE { ListAccess($1, $3) }
  | accessor DROW LPAREN RPAREN      { MatrixRow($1) }
  | accessor DCOL LPAREN RPAREN      { MatrixCol($1) }
  | accessor LSQBRACE expr COMMA expr RSQBRACE
  { MatrixGet($1, $3, $5) } /* t[1,2] */
  | atom { $1 }

accessor2:
    accessor LPAREN args_opt RPAREN { Call($1, $3)  } /* fun(x) */
  | accessor DAPPEND LPAREN expr RPAREN { ListAppend($1, $4) }
  | accessor DLENGTH LPAREN RPAREN { ListLength($1) }
  | accessor LSQBRACE expr RSQBRACE { ListAccess($1, $3) }
  | accessor DROW LPAREN RPAREN      { MatrixRow($1) }
  | accessor DCOL LPAREN RPAREN      { MatrixCol($1) }
  | accessor LSQBRACE expr COMMA expr RSQBRACE
  { MatrixGet($1, $3, $5) } /* t[1,2] */

atom:
    INTLIT          { IntLit ($1) }
  | FLOATLIT          { FloatLit($1) }
  | BOOLLIT         { BoolLit($1) }
  | STRLIT          { StrLit($1) }
  | ID              { Id($1) }
```

```
function_expr:
  FUNC ret_typ LPAREN params_opt RPAREN LBRACE stmt_list RBRACE {FExpr(
      { typ = $2; params = $4; body = List.rev $7} )}

/* Types */
ret_typ:
    VOID { Void }
  | typ { $1 }

typ:
    INT { Int }
  | FLOAT { Float }
  | BOOL { Bool }
  | STRING { String }
  | FUNC LPAREN typ_opt COLON ret_typ RPAREN { Func (
      { param_typs = $3; return_typ = $5 } )}
  | LIST LT typ GT { List($3) }
  /* | MATRIX LT atom COMMA atom GT { Matrix($3, $5) } */
  | MATRIX { Matrix }

/* Helpers */
typ_opt:
  { [] }
| typ_list { List.rev $1 }

typ_list:
  typ { [$1] }
| typ_list COMMA typ { $3 :: $1 }
```

```
opt_items:
  /* nothing */ { [] }
| item_list { List.rev $1 }

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

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

params_opt:
  /* nothing */ { [] }
| param_list    { List.rev $1 }

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


args_opt:
  /* nothing */ { [] }
| arg_list { List.rev $1 }

init_opt:
  /* nothing */                     { Nostmt }
| typ ID ASSIGN expr                { Decl ($1,$2,$4)}
| ID ASSIGN expr                    { Expr(Assign(Id($1), NoOp, $3))}

```

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

```

sast.ml
Authors: David, Jerry, Shuqi, Zichuan

```
open Ast

(* types *)
type styp =
    SInt
  | SFloat
  | SBool
  | SString
  | SVoid
  | SFunc of sfunc_typ
  | SList of styp
  | SEmpty
  | SABSTRACT
  (* Types only used for builtin fillers*)
  | SAny
  | SListElement of styp
  (* Linalg *)
  | SMatrix of int * int

and sfunc_typ = {
  sparam_typs: styp list;
  sreturn_typ: styp;
}

(* No need for op *)

and sbind = styp * string

(* expressions *)
```

```
and sexpr = styp * sx

and sx =
    SIntLit of int
  | SFloatLit of string
  | SBoolLit of bool
  | SStrLit of string
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of sexpr * op * sexpr
  | SCall of sexpr * sexpr list
  | SFExpr of sfexpr
  (* List *)
  | SListLit of sexpr list
  | SListAccess of sexpr * sexpr
  | SListAppend of sexpr * sexpr
  | SListLength of sexpr
  (* Matrix *)
  | SMatrixLit of smatrix
  | SMatrixRow of sexpr
  | SMatrixCol of sexpr
  | SMatrixSet of sexpr * sexpr * sexpr * sexpr
  | SMatrixGet of sexpr * sexpr * sexpr
  | SMatrixAdd of sexpr * sexpr
  | SMatrixSub of sexpr * sexpr
  | SMatrixMulC of sexpr * sexpr
  | SMatrixAddC of sexpr * sexpr
  | SMatrixMulE of sexpr * sexpr
  | SMatrixDivE of sexpr * sexpr
```

```
  | SMatrixMul of sexpr * sexpr
  (* Other *)
  | SClosure of sclsr
  | SNoexpr

and smatrix = {
  srow : int;
  scol : int;
  scontent : sexpr list
}

and sfexpr = {
    styp : styp;
    sparams: sbind list;
    sbody : sstmt list
}

and sclsr = {
  ind: int;
  free_vars: sbind list;
}

(* statements *)
and sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SDecl of styp * string * sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sstmt * sexpr * sexpr * sstmt
```

```
  | SWhile of sexpr * sstmt
  | SNostmt

type sprogram = sstmt list

(* Helper function for converting between AST and SAST *)
let rec styp_of_typ typ = match typ with
    Int -> SInt
  | Bool -> SBool
  | Float -> SFloat
  | String -> SString
  | Void -> SVoid
  | List ty -> if ty = Void then raise (Failure ("Cannot declare a void list"))
  else SList (styp_of_typ ty)
  | Func func_typ -> SFunc { sreturn_typ = styp_of_typ func_typ.return_typ;
      sparam_typs = List.map styp_of_typ func_typ.param_typs} (* Default SFunc
      to be void and no param*)
  | Matrix -> SMatrix(6666666,6666666)
  | Abstract -> SABSTRACT
  (*| _ -> raise (Failure ("styp_of_typ for " ^ string_of_typ typ ^
  " not implmented")) *)

let rec typ_of_styp styp = match styp with
    SInt -> Int
  | SBool -> Bool
  | SFloat -> Float
  | SString -> String
  | SVoid -> Void
  | SList sty -> List (typ_of_styp sty)
  | SMatrix _ -> Matrix
```

```
  | SFunc sfunc_typ -> Func { return_typ = typ_of_styp sfunc_typ.sreturn_typ;
      param_typs = List.map typ_of_styp sfunc_typ.sparam_typs}
  | SEmpty | SABSTRACT | SAny -> raise(Failure ("typ_of_styp shouldn't happen"))
  | _ -> raise (Failure ("typ_of_styp for " ^ (string_of_styp styp) ^
 " not implemented"))

and sfunc_of_func typ = match typ with
    Func _ -> styp_of_typ typ
  | _ -> raise(Failure ("Not a Func"))

(* Pretty printing *)
and string_of_list_sstmt l s = String.concat s (List.map string_of_sstmt l)
and string_of_list_sexpr l s = String.concat s (List.map string_of_sexpr l)
and string_of_list_sbind f l s = String.concat s (List.map f l)

(* PRETTY PRINTING based off of printer.ml *)
and string_of_styp styp = match styp with
    SInt -> "SInt"
  | SFloat -> "SFloat"
  | SString -> "SString"
  | SBool -> "sbool"
  | SVoid -> "SVoid"
  | SFunc(sfunc_typ) -> "SFunc " ^ string_of_styp sfunc_typ.sreturn_typ ^ "("
    ^ (String.concat "," (List.map string_of_styp sfunc_typ.sparam_typs))
    ^ ")"
  | SList styp -> string_of_styp styp ^ "[]" (* "SList(" ^ string_of_styp styp
 ^ ")" *)
  | SEmpty -> "SEmpty"
  | SABSTRACT -> "SABSTRACT"
  | SAny -> "SAny"
```

```
  | SListElement styp -> "SListelement (" ^ (string_of_styp styp) ^ ")"
  (* | SMatrix(sm, sn) -> "SMatrix" ^ "<" ^ string_of_sexpr sm ^ "," ^
  string_of_sexpr sn ^ ">" *)
  | SMatrix(i,j) -> "SMatrix<" ^ string_of_int i ^ "," ^ string_of_int j ^ ">"

and string_of_sexpr (styp,sx) = "(" ^ string_of_styp styp ^ " : "
  ^ (
    match sx with
     SIntLit(l) -> string_of_int l
   | SFloatLit(l) -> l
   | SStrLit(l) -> l
   | SBoolLit(true) -> "true"
   | SBoolLit(false) -> "false"
   | SId(s) -> s
   | SBinop(e1, o, e2) ->
     (string_of_sexpr e1) ^ " " ^ (string_of_op o) ^ " " ^ (string_of_sexpr e2)
   | SUnop(uo, e) -> string_of_uop uo ^ string_of_sexpr e
   | SAssign(e1, o, e2) -> string_of_sexpr e1 ^ string_of_op o ^
   string_of_sexpr e2
   | SCall(se, se_list) -> string_of_sexpr se ^ string_of_list_sexpr
   se_list ", "
   | SFExpr(sfexpr) -> string_of_sfexpr sfexpr
   | SClosure(clsr) -> "{ ind: " ^ string_of_int clsr.ind ^ ", fvs: ("
     ^ string_of_list_sbind string_of_sparam clsr.free_vars ", " ^ ") }"
   | SNoexpr -> ""
   (* List *)
   | SListLit(sexpr_list) -> string_of_list_sexpr sexpr_list ", "
   | SListAccess(s1,s2) -> string_of_sexpr s1 ^ "[" ^ (string_of_sexpr s2) ^ "]"
   | SListAppend(s1,s2) -> string_of_sexpr s1 ^ "Append[" ^
   (string_of_sexpr s2) ^ "]"
```

```
    | SListLength(e) -> "len(" ^ string_of_sexpr e ^ ")"
    (* Matrix *)
    | SMatrixLit(sm) -> string_of_smatrix sm
    | SMatrixSet(m, i, j, x) -> "SMatrixSet( " ^ string_of_sexpr m ^ ", " ^
    string_of_sexpr i ^ ", "
      ^ string_of_sexpr j ^ ", " ^ string_of_sexpr x ^ " )"
    | SMatrixGet(m, i, j) -> "SMatrixGet( " ^ string_of_sexpr m ^ ", " ^
    string_of_sexpr i ^ ", "
      ^ string_of_sexpr j ^ " )"
    | SMatrixAdd(m1, m2) -> "SMatrixAdd( " ^ string_of_sexpr m1 ^ ", " ^
    string_of_sexpr m2 ^ " )"
    | SMatrixSub (m1, m2) -> "SMatrixSub( " ^ string_of_sexpr m1 ^ ", " ^
    string_of_sexpr m2 ^ " )"
    | SMatrixMulC (m, x) -> "SMatrixMulC( " ^ string_of_sexpr m ^ ", " ^
    string_of_sexpr x ^ " )"
    | SMatrixAddC (m, x) -> "SMatrixAddC( " ^ string_of_sexpr m ^ ", " ^
    string_of_sexpr x ^ " )"
    | SMatrixMulE (m1, m2) -> "SMatrixMulE( " ^ string_of_sexpr m1 ^ ", " ^
    string_of_sexpr m2 ^ " )"
    | SMatrixDivE (m1, m2) -> "SMatrixDivE( " ^ string_of_sexpr m1 ^ ", " ^
    string_of_sexpr m2 ^ " )"
    | SMatrixMul (m1, m2) -> "SMatrixMul( " ^ string_of_sexpr m1 ^ ", " ^
    string_of_sexpr m2 ^ " )"
    | _ -> "string_of_sexpr Not Implemented"
  ) ^ ")"

and string_of_sparam sparam = let (styp, s) = sparam in
  string_of_styp styp ^ " " ^ s

and string_of_sfexpr sfexpr =
```

```
  "sfunc " ^ string_of_styp sfexpr.styp ^ " (" ^ string_of_list_sbind
  string_of_sparam sfexpr.sparams ", " ^")"
  ^ "{\n" ^ string_of_list_sstmt sfexpr.sbody "" ^ "}\n"

(* smatrix<2,3>([1,2],[3,4])*)
and string_of_smatrix sm =
  "smatrix<" ^ string_of_int sm.srow ^ "," ^ string_of_int sm.scol ^ ">(" ^
  string_of_list_sexpr sm.scontent "," ^ ")\n"

and string_of_sstmt = function
      SBlock(sstmts) ->
        "{\n" ^ string_of_list_sstmt sstmts "" ^ "}\n"
    | SExpr(sexpr) -> string_of_sexpr sexpr ^ ";\n";
    | SReturn(sexpr) -> "sreturn " ^ string_of_sexpr sexpr ^ ";\n";
    | SIf(e, s1, fL) -> let then_part = match fL with
          SBlock([]) -> ""
        | SIf(_) as s2 -> "elif\n" ^ string_of_sstmt s2
        | s2 -> "else\n" ^ string_of_sstmt s2
      in
      "sif (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s1 ^ then_part
    | SFor(init, se2, se3, s) -> "sfor (" ^ string_of_sstmt init  ^ " ; "
      ^ string_of_sexpr se2 ^ " ; " ^ string_of_sexpr se3  ^ ") " ^
      string_of_sstmt s
    | SWhile(e, s) -> "swhile (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
    | SDecl(t, id, (SVoid,SNoexpr)) -> string_of_styp t ^ " " ^ id ^ ";\n"
    (* Uninitalized *)
    | SDecl(t, id, v) -> string_of_styp t ^ " " ^ id ^ " = " ^
    string_of_sexpr v ^ ";\n"
    | SNostmt -> ""

let string_of_sprogram sast =
  String.concat "" (List.map string_of_sstmt sast) ^ "\n"
```

semant.ml
Authors: David, Jerry, Shuqi, Zichuan

```
(* Check semantics and convert AST to SAST if semantics is correct *)

open Ast
open Sast

module StringMap = Map.Make (String)

(* The main function in semantics.
* Code can be broken in 5 sections: Helper functions, check expressions,
* check expression list,
* check stmt, and stmt list.
*)
let check statements =
  let empty_func styp = ({ sreturn_typ = styp; sparam_typs = [] }) in
  let built_in_decls = List.fold_left (fun map (name,ty) ->
  StringMap.add name ty map) StringMap.empty Builtins.builtins
  in
  (* Helper function for Id *)
  let type_of_id symbol_table s =
      try StringMap.find s symbol_table
      with Not_found -> raise (Failure ("undeclared identifier " ^ s))
  in

  let add_bind map (ty, name) = match ty with
        Func _ -> StringMap.add name (sfunc_of_func ty) map
      | _ -> StringMap.add name (styp_of_typ ty) map
  in
  (* Helper function for empty init. Match styp to the coresponding sx*)
  let empty_sx styp = match styp with
      SInt -> SIntLit (0)
```

```
    | SFloat -> SFloatLit ("0.0")
    | SBool -> SBoolLit (false)
    | SString -> SStrLit ("")
    | SList _ -> SListLit ([])
    | SMatrix(i,j) -> SMatrixLit ({ srow = i; scol= j; scontent = [] })
    (* | SMatrix -> SMatrixLit ([]) *)
    | _ -> raise (Failure ("Empty of " ^ (string_of_styp styp) ^
    " not implemented or shouldn't happen"))
  in

  (* List Helper Func*)
  let rec check_list_type symbol_table = function
      [] -> SEmpty
    | [x] -> let (t,_) = check_expr symbol_table x in t
    | x :: y :: rest ->
          let (t1,_) = check_expr symbol_table x in
          let (t2,_) = check_expr symbol_table y in
          if t1 = t2 then check_list_type symbol_table (y :: rest)
          else raise (Failure(
              "cannot have elements of differing types " ^ string_of_styp t1 ^
              " and " ^ string_of_styp t2 ^ " in same list"))

  (* Helper function for control flow checking bool *)
  and check_bool_expr symbol_table e =
    let (t',e') = check_expr symbol_table e in
    match t' with
        SBool -> (t',e')
      | SABSTRACT -> (SBool,SBoolLit(true))
      | _ -> raise (Failure ("expected Boolean expression in " ^
      string_of_expr e ))
```

```
61
62    (* Helper function for comparing left and right. If both are the same
63    return the typ*)
64    and infer_typ lt rt= match (lt,rt) with
65          (_, SVoid) -> lt
66        | (SFunc f1, SFunc f2) ->
67          (match (f1.sreturn_typ,f2.sreturn_typ) with
68            (SABSTRACT,_) -> rt
69          | (_,SABSTRACT) -> lt
70          | _ ->
71            let same_ret = infer_typ f1.sreturn_typ f2.sreturn_typ in
72            let same_args = List.fold_left2 (fun l f1' f2' ->
73            (infer_typ f1' f2')::l) [] f1.sparam_typs f2.sparam_typs in
74            let infer_func = { sreturn_typ = same_ret; sparam_typs = same_args }
75            in
76            SFunc (infer_func)
77          )
78        (* List *)
79        | (_, SEmpty) -> lt (* can also apply to matrix *)
80        | (SList t1, SList t2) -> SList (infer_typ t1 t2)
81        | (SABSTRACT, _) -> rt
82        | (_,SABSTRACT) -> lt
83        (* Matrix *)
84        (* Add special rule here to link Matrix and List if we were to remove
85        matrix constructor *)
86        | (SMatrix _ ,SMatrix (i,j)) -> SMatrix (i,j)
87        | _ -> if (lt = rt) then lt
88               else raise (Failure ("illegal assignment " ^ string_of_styp lt ^
89               " = " ^ string_of_styp rt))
90
```

```
(* Check expr. Returns sexpr, a tuple of (styp * sx) *)
and check_expr ?fname symbol_table = function
    IntLit l -> (SInt, SIntLit l)
  | FloatLit l -> (SFloat, SFloatLit l)
  | BoolLit l -> (SBool, SBoolLit l)
  | StrLit l -> (SString, SStrLit l)
  | Noexpr -> (SABSTRACT, SNoexpr)
  (* Ids and Funcs *)
  | Id s -> (type_of_id symbol_table s, SId s)
  (* Func Expressions
  * Checks parameters, the body and finally the return type
  *)
  | FExpr fn ->
    let map = List.fold_left add_bind symbol_table fn.params in
    (* Empty function to be filled *)
    let map' = match fname with Some x -> StringMap.add x (SFunc
    (empty_func SVoid)) map
                                | None -> map
    in
    (* Check body's statments *)
    let ( istmt , _ , _ ) = List.fold_left check_stmt
        ( [] , map', Some(styp_of_typ fn.typ) ) fn.body
    in
    (* Check return. If it is of type function, search type from statments*)
    let get_ret t = match t with
      Func _ -> (* Anoynmous function *)
        (* Helper function for inferring the type*)
        let rec ret_search x stmt= match stmt with
          SBlock stlst -> List.fold_left ret_search x stlst
          | SReturn (t,_) -> t
```

```
121            | SIf (_, b1,b2) -> ret_search (ret_search x b1) b2
122            | SFor ( _, _, _, st) -> ret_search x st
123            | SWhile (_,s) -> ret_search x s
124            | _ -> x
125          in
126          List.fold_left ret_search SVoid istmt
127        | _ -> (styp_of_typ t)
128      in
129      let sty = get_ret fn.typ in
130      let sfunc_typ = {sreturn_typ = sty; sparam_typs = List.map ( fun (ty,_) ->
131          styp_of_typ ty) fn.params} in
132      (SFunc(sfunc_typ), SFExpr ({
133        styp = sty;
134        sparams = List.map (fun (xtyp, str) -> (styp_of_typ xtyp,str)) fn.params;
135        sbody = List.rev istmt
136      }) )
137    (* Call a function. Decompose expr into SFunc *)
138    | Call(expr, args) ->
139      (* May need to check args*)
140      (* Check and extract the type and SExpr of an expr *)
141      (* left is ID *)
142      let (t, se) = (match expr with
143          Id(s) -> (type_of_id symbol_table s, SId s)
144        | _ -> check_expr symbol_table expr)
145      in
146      (match t with
147        SFunc(sfunc_t) -> (* Check if arguments are matching and return *)
148        let args_ = List.fold_left2 (fun l func_rt arg -> let (t,se) =
149        check_expr symbol_table arg in
150        (infer_typ func_rt t,se)::l) [] sfunc_t.sparam_typs args
```

```
      in
      (sfunc_t.sreturn_typ, SCall((t,se), List.rev args_))
    | _ -> raise (Failure "not a function"))
  | Assign(e1,op,e2) ->
    let (lt, le') = check_expr symbol_table e1
    and (rt, re') = check_expr symbol_table e2 in
    ((infer_typ lt rt) , SAssign((lt, le'), op, (rt, re')))
  (* Arithmetics *)
  | Binop(e1, op, e2) as e->
    let (lt, le') = check_expr symbol_table e1
    and (rt, re') = check_expr symbol_table e2
    in
    let infered_typ = infer_typ lt rt in
    ( (match op with
        (* May need to add cross type op *)
          Add | Sub | Mul | Div | Mod | Pow when infered_typ = SInt -> SInt
        | Add | Sub | Mul | Div | Pow when infered_typ = SFloat -> SFloat
        | Add                        when infered_typ = SString -> SString
        | Add                        when (match infered_typ with SList _ ->
        true | _ -> false) -> infered_typ
        | Equal | Neq                 -> SBool
        | Less | Leq | Greater | Geq when infered_typ  = SInt || infered_typ =
        SFloat -> SBool
        | And | Or when infered_typ = SBool -> SBool
        | _ -> raise (Failure ("illegal binary operator " ^ string_of_styp lt ^
        " " ^ string_of_op op ^ " " ^ string_of_styp rt ^ " in " ^
      string_of_expr e))), SBinop((lt, le'), op, (rt, re')) )
  | Unop(op, e) as ex ->
    let (t, e') = check_expr symbol_table e in
    ( (match op with
```

```
          Neg when t = SInt -> SInt
        | Neg when t = SFloat -> SFloat
        | Not when t = SBool -> SBool
        | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^ " "
         ^ string_of_styp t ^
      " in " ^ string_of_expr ex))), SUnop(op, (t, e')) )
  (* List *)
  | ListLit(l) -> (SList (check_list_type symbol_table l),  SListLit (List.map
  (check_expr symbol_table) l))
  | ListAccess(e1,e2) ->
    let (t1, se1) = check_expr symbol_table e1
    and (t2, se2) = check_expr symbol_table e2
    in
    let t3 = match t1 with
        SList(t) -> t
      | _ -> raise (Failure ("Not a list: " ^ string_of_expr e1))
    in (match t2 with
        SInt -> (t3, SListAccess((t1, se1), (t2, se2)))
      | _ -> raise (Failure ("can't access list with non-integer index type")))
  | ListAppend(e1,e2) ->
    let (t1, se1) = check_expr symbol_table e1
    and (t2, se2) = check_expr symbol_table e2
    in
    let t3 = match t1 with
        SList(t) -> t
      | _ -> raise (Failure ("Not a list: " ^ string_of_expr e1))
    in
    if t2 = t3 then (SVoid, SListAppend((t1, se1), (t2, se2))) else raise (
    Failure ("can't append list with different type"))
  | ListLength(e) ->
```

```
    let (t1, se) = check_expr symbol_table e
    in
    let t2 = match t1 with
        SList(t3) -> t3
      | _ -> raise (Failure ("Not a list: " ^ string_of_expr e))
    in
    (SInt, SListLength((t2, se)))

  (* Matrix *)
  (* Need to check dimension *)
  | MatrixLit (e) -> (* [ [Float, Float], [Float, Float], [Float, Float] ] =>
  expr list *)
  (* expects List (List (Float))*)
  (* return sm , expects row to be type List(Float)*)
  let check_row sm row = match row with
      ListLit fl ->
      let col_ct = if sm.scol = 0 then (List.length fl)
        else ( if (List.length fl) = sm.scol then sm.scol else raise( Failure
        ("col size mismatch. Expects" ^ string_of_int sm.scol ^ "but got " ^
        string_of_int (List.length fl))))
      in
      {srow = sm.srow+1; scol= col_ct; scontent =
          (check_expr symbol_table row)::sm.scontent }
    | expr -> raise (Failure (" Row must be type ListLit, got " ^
    string_of_expr expr))
  in
  let sm = List.fold_left check_row { srow = 0; scol=0; scontent = [] } e in
  let sm_rev = { srow = sm.srow; scol = sm.scol; scontent =
      List.rev sm.scontent } in
  (SMatrix(sm_rev.srow,sm_rev.scol), SMatrixLit(sm_rev))
```

```
241     | MatrixRow (m) ->
242       let (t1, se) = check_expr symbol_table m in
243       let t1 = match t1 with
244           SMatrix _ -> t1
245         | _ -> raise (Failure ("Not a matrix: " ^ string_of_sexpr (t1,se)))
246       in
247       (SInt, SMatrixRow((t1,se)))
248     | MatrixCol (m) ->
249       let (t1, se) = check_expr symbol_table m in
250       let t1 = match t1 with
251           SMatrix _ -> t1
252         | _ -> raise (Failure ("Not a matrix: " ^ string_of_sexpr (t1,se)))
253       in
254       (SInt, SMatrixCol((t1,se)))
255     (* since i, j, x are defined by primitive types, we can just build a sexpr
256     on the fly here. *)
257     | MatrixSet (m, i, j, x) ->
258       let (sm,si,sj) = check_matrix_idx m i j symbol_table in
259       (SVoid, SMatrixSet (sm, si, sj, (check_expr symbol_table x)))
260
261     | MatrixGet (m, i, j) ->
262       let (sm,si,sj) = check_matrix_idx m i j symbol_table in
263       (SFloat, SMatrixGet (sm, si, sj))
264
265     | MatrixAdd (m1, m2) ->
266       let (se1, se2) = check_two_matrices_dim symbol_table m1 m2 in
267       (SVoid, SMatrixAdd (se1, se2))
268
269     | MatrixSub (m1, m2) ->
270       let (se1, se2) = check_two_matrices_dim symbol_table m1 m2 in
```

```
      (SVoid, SMatrixSub (se1, se2))

    | MatrixMulC (m, x) ->
      let se1 = check_expr symbol_table m in
      let se2 = check_expr symbol_table x in
      (SVoid, SMatrixMulC (se1, se2))

    | MatrixAddC (m, x) ->
      let se1 = check_expr symbol_table m in
      let se2 = check_expr symbol_table x in
      (SVoid, SMatrixAddC (se1, se2))

    | MatrixMulE (m1, m2) ->
      let (se1, se2) = check_two_matrices_dim symbol_table m1 m2 in
      (SVoid, SMatrixMulE (se1, se2))

    | MatrixDivE (m1, m2) ->
      let (se1, se2) = check_two_matrices_dim symbol_table m1 m2 in
      (SVoid, SMatrixDivE (se1, se2))

    | MatrixMul (m1, m2) ->
      let (t1, sx1) = check_expr symbol_table m1 in
      let (t2, sx2) = check_expr symbol_table m2 in
      let (row1, col1) = match t1 with
      | SMatrix (r1, c1) -> (r1, c1)
      | _ -> raise (Failure "Shoudn't happen") in
      let (row2, col2) = match t2 with
      | SMatrix (r2, c2) -> (r2, c2)
      | _ -> raise (Failure "Shoudn't happen") in
      if col1 = row2 then
```

```
      (SMatrix (row1, col2), SMatrixMul ((t1, sx1), (t2, sx2)))
    else raise (Failure ("Matrix dimensions do not match
    for matrix multiplication."))

(* Helper function for checking equal dimensions *)
and check_two_matrices_dim sym_table m1 m2 =
  let (t1, sx1) = check_expr sym_table m1 in
  let (t2, sx2) = check_expr sym_table m2 in
  let (row1, col1) = match t1 with
    | SMatrix (r1, c1) -> (r1, c1)
    | _ -> raise (Failure "Shoudn't happen") in
  let (row2, col2) = match t2 with
    | SMatrix (r2, c2) -> (r2, c2)
    | _ -> raise (Failure "Shoudn't happen") in
  Printf.printf "%d" row1; Printf.printf "%d" col1;
  Printf.printf "%d" row2; Printf.printf "%d" col2;
  if row1 = row2 && col1 = col2 then
    ((t1, sx1), (t2, sx2))
  else raise (Failure ("Matrix dimensions do not match."))

(* Helper function for checking matrixaccess. If successful,
return checked expr of m,i, and j*)
and check_matrix_idx m i j symbol_table =
  let (t1,se1) = check_expr symbol_table m in
  let (t2,se2) = check_expr symbol_table i in
  let i = match se2 with SIntLit(i) -> i | _ -> -1 in
  let (t3,se3) = check_expr symbol_table j in
  let j = match se3 with SIntLit(j) -> j | _ -> -1 in
  let var = match se1 with SId var -> var | _ ->
  raise (Failure "Shoudn't happen") in
```

```
  let (mi,mj) = match (StringMap.find var symbol_table) with SMatrix(mi,mj) ->
  (mi,mj) | typ -> raise(Failure("Not calling a matrix. Got " ^
  string_of_styp typ)) in
  if i < mi && j < mj
  then
  ((t1,se1),(t2,se2),(t3,se3))
  else raise ( Failure ("Indexing Matrix out of range, expects within (" ^
  string_of_int mi ^ "," ^ string_of_int mj ^ "), but indexing (" ^
  string_of_int i ^ "," ^ string_of_int j ^ ")"))

(* Checks statement
* Keeps a list of SAST, the context (symbol_table), and the return type
*)
and check_stmt (curr_lst, symbol_table,return_typ)  = function
    Block(stmt_list) ->
      let (istmts,_,_) = List.fold_left check_stmt
      (curr_lst,symbol_table,return_typ) stmt_list
      in (SBlock (List.rev istmts) :: curr_lst , symbol_table,return_typ)
  | Expr(e) -> (SExpr (check_expr symbol_table e):: curr_lst, symbol_table,
  return_typ)
  | Decl(t,s,e) as exp -> (match e with
      Noexpr -> let ty = match t with
          Func _ -> raise (Failure
          ("Cannot declare an uninitialized function."))
        | typ -> styp_of_typ typ
      in
      (SDecl (ty, s, (SEmpty,empty_sx ty))::curr_lst, StringMap.add s ty
      symbol_table,return_typ)
    | _ -> (* initialized version *)
      let (tr,er) = match t with (* check right side and for functions check
```

```
      if it a built-in function *)
          Func _  -> check_expr symbol_table e ~fname:s
        | _ -> check_expr symbol_table e
      in
      if StringMap.mem s built_in_decls
      then raise (Failure ("Variable name cannot be a built-in function" ^
      (string_of_stmt exp)))
      else let tl = match t with (* Func with *)
        | Func _ -> sfunc_of_func t
        | typ -> styp_of_typ typ
      in
      (SDecl (infer_typ tl tr,s, (tr,er)):: curr_lst, StringMap.add s tl
      symbol_table,return_typ)
    )
  | Return  e -> let (t1,e1) = check_expr symbol_table e in
    let t = match return_typ with
        Some t2 -> infer_typ t2 t1
      | None -> raise (Failure "Cannot return if not within a function")
    in (SReturn (t,e1) :: curr_lst, symbol_table,return_typ)
  | If(e,st1,st2) ->
      let (st1',_,_) = check_stmt ([],symbol_table, return_typ) st1
      and (st2',_,_) = check_stmt ([],symbol_table, return_typ) st2
      in
      (SIf ( check_bool_expr symbol_table e ,
      List.hd st1', List.hd st2')::curr_lst, symbol_table, return_typ)
  | For(init,e2,e3,st) ->
      let (ist,st1,_) = check_stmt ([],symbol_table,return_typ) init in
      let (sst,_,_) = check_stmt ([],st1,return_typ) st in
      let se2 = check_bool_expr st1 e2 in
      let se3 = check_expr st1 e3 in
```

```
      (SFor(List.hd ist, se2, se3, List.hd sst) :: curr_lst, symbol_table,
      return_typ)
  | While (p,s) -> let (is,_,_) = check_stmt ([],symbol_table,return_typ) s in
      (SWhile(check_bool_expr symbol_table p, List.hd is):: curr_lst,
      symbol_table,return_typ)
  | Nostmt -> (SNostmt::curr_lst,symbol_table,return_typ)
in

let (stmts, _,_) = List.fold_left check_stmt ([],built_in_decls,None) statements
  in SBlock (List.rev stmts) :: []
```

codegen.ml
Authors: David, Jerry, Shuqi, Zichuan

```
(*
Code generation:
Translate takes a semantically checked AST and produces LLVM IR\
*)

module L = Llvm

open Ast
open Sast
open Lift

module StringMap = Map.Make(String)

let translate functions =
  let context = L.global_context () in (* global data container *)
  let llmem = L.MemoryBuffer.of_file "builtins.bc" in
  let llm = Llvm_bitreader.parse_bitcode context llmem in

  (* Get types from the context *)
  let i32_t      = L.i32_type    context (* int *)
  and i1_t       = L.i1_type     context (* nool *)
  and float_t    = L.double_type context (* float *)
  and void_t     = L.void_type   context (* void *)
  and void_ptr_t = L.pointer_type (L.i8_type context)
  and str_t      = L.pointer_type (L.i8_type context) (* string *)

  and list_t     = L.pointer_type (match L.type_by_name llm "struct.List" with
      None -> raise (Failure "Missing implementation for struct List")
    | Some t -> t)
  and lst_element_t = L.pointer_type (match L.type_by_name llm
```

```
31    "struct.List_element" with
32        None -> raise (Failure "Missing implementation for struct List_element")
33      | Some t -> t)
34    and matrix_t   = L.pointer_type (match L.type_by_name llm "struct.gsl_matrix"
35    with
36        None -> raise (Failure "Missing implementation for struct gsl_matrix")
37      | Some t -> t)
38
39    (* Create the LLVM compilation module into which we will generate code *)
40    and the_module = L.create_module context "LOL" in
41
42    (*
43    Return the LLVM type for a LOL function type
44    For non-main functions, add void_ptr in front of parameters
45
46    Parameters:
47    - ret_t: return type as styp
48    - param_ts: list of parameters as styp
49    *)
50    let rec ltype_of_func name (ret_t : styp) param_ts =
51      let param_types = (List.map ltype_of_styp param_ts) in
52      let param_types =
53        if name = "main" then param_types
54        else void_ptr_t :: param_types
55      in L.function_type (ltype_of_styp ret_t) (Array.of_list param_types)
56
57    (* Return the LLVM type for a LOL function type *)
58    and ltype_of_lfexpr name (lfexpr : lfexpr) =
59      ltype_of_func name lfexpr.lreturn_typ (List.map fst lfexpr.lparams)
60
```

```
(* Return the LOL return and parameter types for a LLVM type *)
and typ_of_lfexpr lfexpr = SFunc({
  sreturn_typ = lfexpr.lreturn_typ;
  sparam_typs = List.map fst lfexpr.lparams;
})

(* Return the LLVM type for a LOL function type *)
and ltype_of_sfunction name (sfunc : sfunc_typ) =
  ltype_of_func name sfunc.sreturn_typ sfunc.sparam_typs

and ltype_of_clsr name sfexpr =
  let func_t = L.pointer_type (ltype_of_lfexpr name sfexpr) in
  L.struct_type context [|func_t; void_ptr_t|]

and ltype_of_clsr_func name (sfunc : sfunc_typ) =
  let func_t = L.pointer_type (ltype_of_sfunction name sfunc) in
  L.struct_type context [|func_t; void_ptr_t|]

(* Return the LLVM type for a LOL type *)
and ltype_of_styp = function
    SInt   -> i32_t
  | SBool  -> i1_t
  | SFloat -> float_t
  | SVoid  -> void_t
  | SString -> str_t
  | SFunc(ftype) -> ltype_of_clsr_func "" ftype
  | SEmpty -> void_t
  | SList _ -> list_t
  | SListElement _ -> lst_element_t
  | SMatrix(_,_) -> matrix_t
```

```
    | SAny -> void_ptr_t
    | SABSTRACT -> void_ptr_t
    (*| _ -> raise (Failure "not yet implemented"*)

  (* Helper funciton to retrieve a function from context*)
  and get_func s lmodule =
    match L.lookup_function s lmodule with
        None -> raise( Failure (s ^ "not defined in builtin"))
      | Some f -> f

  in

  (* Creates a %name = insertvalue %agg, %val, %idx*)
  let insert_value builder agg i v = L.build_insertvalue agg v i "tmp__"
  builder in
  let rec generate_seq n = if n >= 0 then (n :: (generate_seq (n-1))) else [] in

  (*
  Builds Built-in functions
  Add each builtin function into Stringmap.
  Function with name length does not gets added.
  *)
  let builtins : (Sast.styp * L.llvalue) StringMap.t =
    List.fold_left (fun m (name, ty) ->
      let ftype = match ty with
          SFunc(f) -> f
        | _ -> (raise (Failure "shouldn't happen"))
      in
      let params = List.map ltype_of_styp ftype.sparam_typs in
      let ltype = L.function_type (ltype_of_styp ftype.sreturn_typ)
```

```
      (Array.of_list params)
      in
      StringMap.add name (ty, (L.declare_function name ltype the_module)) m
    ) StringMap.empty Builtins.builtins in

  (*
  Define each function (arguments and return type) so we can
  call it even before we've created its body

  Like bultins, adds each function into StringMap

  *)
  let function_decls : (L.llvalue * lfexpr) StringMap.t =
    let function_decl m (name, lfexpr) =
      let ftype = ltype_of_lfexpr name lfexpr in
      StringMap.add name (L.define_function name ftype the_module, lfexpr) m in
    List.fold_left function_decl StringMap.empty functions in

  (*
  Fill in the body for a given function

  Unpacking args and env vars,
  skip for main as its env is empty and has no params
  *)
  let build_function_body (name, lfexpr) =
    let (the_function, _) = StringMap.find name function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in

  (*
  Construct the function's "locals": formal arguments and locally
```

```
151     Construct the functions "locals": formal arguments and locally
152     declared variables.  Allocate each on the stack, initialize their
153     value, if appropriate, and remember their values in the "locals" map
154     *)
155     let local_vars =
156       let add_param m (t, n) p =
157         let () = L.set_value_name n p in
158         let local = L.build_malloc (ltype_of_styp t) n builder in
159         let _ = L.build_store p local builder in
160         StringMap.add n (t, local) m
161       in
162       let param_list = Array.to_list (L.params the_function) in
163       let params =
164         if List.length param_list <= 1
165         then StringMap.empty
166         else List.fold_left2 add_param StringMap.empty lfexpr.lparams
167         (List.tl param_list)
168     in
169
170     (* Create the local versions of the free variables. Unpack the environment
171      * pointer and initialize the local free variables using values obtained
172      * from the pointer *)
173     let env =
174       if param_list = []
175       then L.const_null void_ptr_t
176       else List.hd param_list
177     in
178
179     let () = L.set_value_name "env" env in
180     let env_void_ptr = L.build_malloc void_ptr_t "env" builder in
```

```
181     let _ = L.build_store env env_void_ptr builder in
182     let env_p = L.build_load env_void_ptr "env_p" builder in
183
184     let params_of_lfexpr lfexpr = match lfexpr.lfvs with
185         [] -> params
186       | _ ->
187         let ptr_of_fv (t, _) = L.pointer_type (ltype_of_styp t) in
188         let env_struct = L.struct_type context (Array.of_list
189           (List.map ptr_of_fv lfexpr.lfvs)) in
190         let env_ptr_t = L.pointer_type env_struct in
191         let env_ptr = L.build_bitcast env_p env_ptr_t "env_p" builder in
192         let env_val = L.build_load env_ptr "env_val" builder in
193         let add_free_var m (t, n) idx =
194           let free_var = L.build_extractvalue env_val idx "tmp_" builder in
195           StringMap.add n (t, free_var) m
196         in
197         let fvs_count = List.length lfexpr.lfvs in
198         List.fold_left2 add_free_var params lfexpr.lfvs
199             (List.rev (generate_seq (fvs_count - 1)))
200       in
201
202       let params_fvs = match name with
203           "main" -> params
204         | _ -> params_of_lfexpr lfexpr
205       in
206
207       (* Allocate a closure of the function within itself for recursive calls *)
208       let clsr_t = ltype_of_clsr name lfexpr in
209       let clsr_p = L.build_malloc clsr_t lfexpr.lname builder in
210       let clsr_val = List.fold_left2 (insert_value builder)
```

```
211         (L.const_null clsr_t) [0;1] [the_function;env_p] in
212       let _ = L.build_store clsr_val clsr_p builder in
213       let func_t  = typ_of_lfexpr lfexpr in
214       StringMap.add lfexpr.lname (func_t, clsr_p) params_fvs
215     in
216
217     (* Builder *)
218     let rec expr builder (m : (styp * L.llvalue) StringMap.t)
219     ((styp, e) : sexpr) =
220
221
222       let lookup n = let (_, llval) = try StringMap.find n m with
223          Not_found -> raise (Failure ("Variable not found: " ^ n)) in llval
224       in
225
226       (* Helper function for closure*)
227       let build_clsr clsr =
228         let fvs = List.map snd clsr.free_vars in
229         let llfvs = List.map lookup fvs in
230         let fvs_t = List.map ltype_of_styp (List.map fst clsr.free_vars) in
231         let fvs_ptr_t = List.map L.pointer_type fvs_t in
232         let env_struct_t = L.struct_type context (Array.of_list fvs_ptr_t) in
233         let env_struct = L.build_malloc env_struct_t "tmp_" builder in
234         let idxs = List.rev (generate_seq ((List.length fvs) - 1)) in
235         let env_val = List.fold_left2 (insert_value builder)
236               (L.const_null env_struct_t) idxs llfvs in
237         let _ = L.build_store env_val env_struct builder in
238         let env_struct_p = L.build_bitcast env_struct void_ptr_t "env_p" builder
239         in
240
```

```
241         (* Pack the function ptr and the env ptr into the closure struct *)
242         let func_name = "f" ^ (string_of_int clsr.ind) in
243         let (llfunc, sfexpr) = StringMap.find func_name function_decls in
244         let llclosure_struct_t = ltype_of_clsr func_name sfexpr in
245         let clsr_val = List.fold_left2 (insert_value builder)
246         (L.const_null llclosure_struct_t) [0;1] [llfunc; env_struct_p] in
247         clsr_val
248       in
249
250       match e with
251           SIntLit x -> L.const_int i32_t x
252         | SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
253         | SStrLit s -> L.build_global_stringptr s "str" builder
254         | SFloatLit x -> L.const_float_of_string float_t x
255         | SId s -> L.build_load (lookup s) s builder
256         | SClosure clsr -> build_clsr clsr
257         | SNoexpr -> L.const_int i32_t 0
258         | SAssign(e1, op, e2) ->
259           let (_,se1) = e1 in
260           let new_v = match op with
261                     NoOp -> expr builder m e2
262                   | Add -> expr builder m (styp, SBinop(e1, Add, e2))
263                   | Sub -> expr builder m (styp, SBinop(e1, Sub, e2))
264                   | Mul -> expr builder m (styp, SBinop(e1, Mul, e2))
265                   | Div -> expr builder m (styp, SBinop(e1, Div, e2))
266                   | _ -> raise (Failure (string_of_op op ^
267                  " not yet implemented"))
268           in
269           (match se1 with
270              SId s -> ignore(L.build_store new_v (lookup s) builder); new_v
```

```
271          | SListAccess (arr,i) ->
272            let (_,se2) = arr in
273            let arr = match se2 with SId(x) -> x |
274            _-> raise (Failure("ListAccess shoudn't happen")) in
275            let ltype = ltype_of_styp styp in
276            let lst = L.build_load (lookup arr) arr builder in
277            let index = expr builder m i in
278            let data = L.build_malloc ltype "data" builder in
279            ignore(L.build_store new_v data builder);
280            let data = L.build_bitcast data void_ptr_t "data" builder in
281            let list_set_f = get_func "list_set" the_module in
282            ignore(L.build_call list_set_f [| lst; data; index |]
283            "list_set" builder);
284            new_v
285          | SMatrixGet (mat, i, j) ->
286            let (_,se2) = mat in
287            let mat = match se2 with SId(x) -> x |
288            _-> raise (Failure("SMatrixGet shoudn't happen")) in
289            let lst = L.build_load (lookup mat) mat builder in
290            let li = expr builder m i in
291            let lj = expr builder m j in
292            let matrix_set_f = get_func "mset" the_module in
293            ignore(L.build_call matrix_set_f [| lst; li; lj; new_v |]
294            "mset" builder);
295            new_v
296          | _ -> raise (Failure ("assignment for " ^ (string_of_sexpr e2)
297                 ^ "SAssign not implemented in codegen")))
298        | SBinop (e1, op, e2) ->
299            let (t, _) = e1
300            and e1' = expr builder m e1
```

```
            and e2' = expr builder m e2 in
            (match snd e1, snd e2 with
               _ -> (match t with
                   SFloat -> (match op with
                       Add     -> L.build_fadd
                     | Sub     -> L.build_fsub
                     | Mul     -> L.build_fmul
                     | Div     -> L.build_fdiv
                     | Equal   -> L.build_fcmp L.Fcmp.Oeq
                     | Neq     -> L.build_fcmp L.Fcmp.One
                     | Less    -> L.build_fcmp L.Fcmp.Olt
                     | Leq     -> L.build_fcmp L.Fcmp.Ole
                     | Greater -> L.build_fcmp L.Fcmp.Ogt
                     | Geq     -> L.build_fcmp L.Fcmp.Oge
                     | Pow     -> let pow_f = get_func "pow" the_module in
                       let powpowpow e1 e2 str builder = L.build_call pow_f
                       [| e1; e2 |] str builder in
                       powpowpow
                     | _ ->
                       raise (Failure ("internal error: "
                         ^ "semant should have rejected and/or on float"))
                   ) e1' e2' "tmp" builder
                 | SInt -> (match op with
                     | Add     -> L.build_add
                     | Sub     -> L.build_sub
                     | Mul     -> L.build_mul
                     | Div     -> L.build_sdiv
                     | Mod     -> L.build_srem
                     | And     -> L.build_and
                     | Or      -> L.build_or
```

```
                    | Equal   -> L.build_icmp L.Icmp.Eq
                    | Neq     -> L.build_icmp L.Icmp.Ne
                    | Less    -> L.build_icmp L.Icmp.Slt
                    | Leq     -> L.build_icmp L.Icmp.Sle
                    | Greater -> L.build_icmp L.Icmp.Sgt
                    | Geq     -> L.build_icmp L.Icmp.Sge
                    | _ -> raise (Failure( "Not exhausive" ))
                  ) e1' e2' "tmp" builder
                | SBool -> (match op with
                      And     -> L.build_and
                    | Or      -> L.build_or
                    | Equal   -> L.build_icmp L.Icmp.Eq
                    | Neq     -> L.build_icmp L.Icmp.Ne
                    | Less    -> L.build_icmp L.Icmp.Slt
                    | Leq     -> L.build_icmp L.Icmp.Sle
                    | Greater -> L.build_icmp L.Icmp.Sgt
                    | Geq     -> L.build_icmp L.Icmp.Sge
                    | _         -> raise (Failure ("operation "
                    ^ (string_of_op op)
                                                ^ " not implemented for type "
                                                ^ (string_of_styp t)))
                  ) e1' e2' "tmp" builder
                | SString -> (match op with
                      Add -> L.build_call (snd (StringMap.find "string_concat"
                        builtins)) [| e1'; e2'|] "string_concat" builder
                    | Equal -> (L.build_icmp L.Icmp.Ne) (L.const_int i32_t 0)
                                (L.build_call (snd (StringMap.find
                                  "string_equals" builtins)) [| e1'; e2'|]
                                  "string_equals" builder) "tmp" builder
                    | Neq -> (L.build_icmp L.Icmp.Eq) (L.const_int i32_t 0)
```

```
                                (L.build_call (snd (StringMap.find
                                  "string_equals" builtins)) [| e1'; e2'|]
                                  "string_equals" builder) "tmp" builder
                      | _ -> raise (Failure ("operation " ^ (string_of_op op)
                                            ^ " not implemented for type "
                                            ^ (string_of_styp t))))
                  (* NEED STRING IMPLEMENTATION *)
                  | _ -> (match op with
                        Equal -> (L.build_icmp L.Icmp.Eq) e1' e2' "tmp" builder
                      | Neq -> (L.build_icmp L.Icmp.Ne) e1' e2' "tmp" builder
                      | _ -> raise (Failure ("operation " ^ (string_of_op op)
                                            ^ " not implemented for type "
                                            ^ (string_of_styp t))))
    ))
      | SUnop(op, e) ->
        let (t, _) = e in
        let ep = expr builder m e in
        (match op with
           Neg when t = SFloat -> L.build_fneg
         | Neg when t = SInt -> L.build_neg
         | Not when t = SBool -> L.build_not
         | _ -> raise (Failure ("operation " ^ (string_of_uop op) ^
                                " not implemented for type "
                                ^ (string_of_styp t)))) ep "tmp" builder
      | SCall((t, SId(name)), args) when StringMap.mem name builtins ->
          (let func_t = match t with
                SFunc(func_t) -> func_t
              | _ -> raise (Failure "This should never happen")  in
           (match func_t.sreturn_typ with
              SVoid -> (let arg_array = Array.of_list
```

```
391                 (List.map (fun arg -> expr builder m arg) args) in
392                   L.build_call (snd (StringMap.find name builtins))
393                   arg_array "" builder)
394             | _ -> (let arg_array = Array.of_list (List.map
395               (fun arg -> expr builder m arg) args) in
396                 L.build_call (snd (StringMap.find name builtins))
397                 arg_array "_result" builder)))
398       | SCall((t, s), args) ->
399           let func_t = match t with
400               SFunc(func_t) -> func_t
401             | _ -> raise (Failure "wrong type for function call") in
402           let clsr_val = expr builder m (t, s) in
403           let func_ptr = L.build_extractvalue clsr_val 0 "fp" builder in
404           let env_ptr = L.build_extractvalue clsr_val 1 "envp" builder in
405           let llargs = env_ptr :: (List.rev (List.map (expr builder m)
406             (List.rev args))) in
407           let result =
408               (match func_t.sreturn_typ with SVoid -> "" | _ -> "_result") in
409           L.build_call func_ptr (Array.of_list llargs) result builder
410       (* List *)
411       | SListLit(contents) ->
412         let rec list_fill m lst = (function
413             [] -> lst
414           | sx :: rest ->
415             let (typ, _) = sx in
416             let data = (match typ with
417               SList _  -> expr builder m sx
418             | _ -> let data = L.build_malloc (ltype_of_styp typ) "data" builder in
419                 let llvalue = expr builder m sx
420                 in ignore (L.build_store llvalue data builder); data)
```

```
421            in
422            let list_append_f = get_func "list_append" the_module in
423            let data = L.build_bitcast data void_ptr_t "data" builder in
424                      ignore (L.build_call list_append_f [| lst; data |] ""
425                      builder);
426                      list_fill m lst rest)
427        in
428        let list_init_f = get_func "list_init" the_module in
429        let lst = L.build_call list_init_f [||] "list_init" builder in
430              ignore(list_fill m lst contents); lst
431      | SListAccess(arr, i) ->
432        let ltype = ltype_of_styp styp in
433        let arr_var = expr builder m arr in
434        let idx = expr builder m i in
435        let list_access_f = get_func "list_get" the_module in
436        let data_ptr = L.build_call list_access_f [|arr_var; idx|] "list_get"
437        builder in
438        (match styp with
439          SList _ ->
440            L.build_bitcast data_ptr ltype "data" builder
441          | _ ->
442          let data_ptr = L.build_bitcast data_ptr (L.pointer_type ltype) "data"
443          builder in
444          L.build_load data_ptr "data" builder
445        )
446      | SListAppend(arr, it) ->
447        let arr_var = expr builder m arr in
448        let item = expr builder m it in
449        let (typ, _) = it in
450        let data_ptr = L.build_malloc (ltype_of_styp typ) "data_ptr" builder in
```

```
451         ignore(L.build_store item data_ptr builder);
452         let data = L.build_bitcast data_ptr void_ptr_t "data" builder in
453         let list_append_f = get_func "list_append" the_module in
454           L.build_call list_append_f [|arr_var; data|] "" builder
455       | SListLength(arr) ->
456         let arr_var = expr builder m arr in
457         let list_length_f = get_func "list_length" the_module in
458         L.build_call list_length_f [|arr_var|] "list_length" builder
459
460       (* Matrix Operations *)
461       | SMatrixLit(sm) ->
462         (* Function for fold_left for each SFloat, set in matrix *)
463         let matrix_fill_row (m,mat,i,j) sx =
464             let data = expr builder m sx in
465             let matrix_set_elem_f = get_func "mset" the_module in
466             let jl = expr builder m (SInt,SIntLit(j)) in
467             let il = expr builder m (SInt,SIntLit(i)) in
468             ignore (L.build_call matrix_set_elem_f [| mat; il ; jl ; data |]
469             "" builder);
470             (m,mat,i,j+1)
471         in
472         (* Function for fold_left for each row (SList)  *)
473         let matrix_fill (m,mat,i) (_,sx)=
474             match sx with
475                 SListLit l -> let (m,mat,_,_) = List.fold_left matrix_fill_row
476                 (m,mat,i,0) l in (m,mat,i+1)
477               | _ -> raise (Failure ("MatrixFill in SMatrixLit Shoudn't happen"))
478         in
479
480         let matrix_init_f = get_func "minit" the_module in
```

```
481         let row = expr builder m (SInt,SIntLit(sm.srow)) in
482         let col = expr builder m (SInt,SIntLit(sm.scol)) in
483         let mat = L.build_call matrix_init_f [|row; col|] "minit" builder in
484         let (_,mat,_) = List.fold_left matrix_fill (m,mat,0) sm.scontent in
485         mat
486       | SMatrixRow(mat) ->
487         let mat_var = expr builder m mat in
488         let matrix_row_f = get_func "mrow" the_module in
489         L.build_call matrix_row_f [|mat_var|] "mcol" builder
490       | SMatrixCol(mat) ->
491         let mat_var = expr builder m mat in
492         let matrix_col_f = get_func "mcol" the_module in
493         L.build_call matrix_col_f [|mat_var|] "mcol" builder
494       | SMatrixGet (mat,i,j) ->
495         let il = expr builder m i in
496         let jl = expr builder m j in
497         let mat = expr builder m mat in
498         let matrix_get_elem_f = get_func "mget" the_module in
499         L.build_call matrix_get_elem_f [|mat;il;jl|] "" builder;
500
501       | SMatrixSet (mat, i, j, x) ->
502         let matl = expr builder m mat in
503         let il = expr builder m i in
504         let jl = expr builder m j in
505         let xl = expr builder m x in
506         let matrix_set_elem_f = get_func "mset" the_module in
507         L.build_call matrix_set_elem_f [|matl; il; jl; xl|] "" builder
508
509       | SMatrixAdd (mat1, mat2) ->
510         let mat1l = expr builder m mat1 in
```

```
511         let mat2l = expr builder m mat2 in
512         let matrix_add_f = get_func "madd" the_module in
513         L.build_call matrix_add_f [|mat1l; mat2l|] "" builder
514 
515       | SMatrixSub (mat1, mat2) ->
516         let mat1l = expr builder m mat1 in
517         let mat2l = expr builder m mat2 in
518         let matrix_f = get_func "msub" the_module in
519         L.build_call matrix_f [|mat1l; mat2l|] "" builder
520 
521       | SMatrixMulC (mat, x) ->
522         let matl = expr builder m mat in
523         let xl = expr builder m x in
524         let matrix_f = get_func "mmulc" the_module in
525         L.build_call matrix_f [|matl; xl|] "" builder
526 
527       | SMatrixAddC (mat, x) ->
528         let matl = expr builder m mat in
529         let xl = expr builder m x in
530         let matrix_f = get_func "maddc" the_module in
531         L.build_call matrix_f [|matl; xl|] "" builder
532 
533       | SMatrixMulE (m1, m2) ->
534         let mat1l = expr builder m m1 in
535         let mat2l = expr builder m m2 in
536         let matrix_f = get_func "mmule" the_module in
537         L.build_call matrix_f [|mat1l; mat2l|] "" builder
538 
539       | SMatrixDivE (m1, m2) ->
540         let mat1l = expr builder m m1 in
```

```
541         let mat2l = expr builder m m2 in
542         let matrix_f = get_func "mdive" the_module in
543         L.build_call matrix_f [|mat1l; mat2l|] "" builder
544
545       | SMatrixMul (m1, m2) ->
546         let mat1l = expr builder m m1 in
547         let mat2l = expr builder m m2 in
548         let matrix_f = get_func "mmul" the_module in
549         L.build_call matrix_f [|mat1l; mat2l|] "" builder
550
551       | _ as x -> print_endline(string_of_sexpr (styp, x));
552           raise (Failure "expr not implemented in codegen")
553     in
554     (* Each basic block in a program ends with a "terminator" instruction i.e.
555     one that ends the basic block. By definition, these instructions must
556     indicate which basic block comes next -- they typically yield "void" value
557     and produce control flow, not values *)
558     (* Invoke "instr builder" if the current block doesn't already
559        have a terminator (e.g., a branch). *)
560     let add_terminal builder instr =
561       match L.block_terminator (L.insertion_block builder) with
562           None -> ignore (instr builder)
563         | Some _ -> ()
564     in
565
566     (* Build the code for the given statement; return the builder for
567        the statement's successor (i.e., the next instruction will be built
568        after the one generated by this call) *)
569     (* Imperative nature of statement processing entails imperative OCaml *)
570     let rec stmt builder m = function
```

```
571     (* Throw away the scope generated by block
572     * since the block should not be modifying our current scope *)
573       SBlock s1 ->
574       let helper (bldr, map) = stmt bldr map in
575       let (b, _) = List.fold_left helper (builder, m) s1 in
576       (b, m)
577     | SExpr e -> let _ = expr builder m e in (builder, m)
578     (* Right now SVDecl and SFDecl has no difference except se is optional for
579     VDecl*)
580     | SDecl(t, n, se) ->
581       let se' = expr builder m se in
582       let alloc_clsr clsr =
583         let func_name = ("f" ^ (string_of_int clsr.ind)) in
584         let (_, lfexpr) = StringMap.find func_name function_decls in
585         let func_t = L.pointer_type (ltype_of_lfexpr func_name lfexpr) in
586         let llclosure_struct_t = L.struct_type context [|func_t; void_ptr_t|] in
587         L.build_malloc llclosure_struct_t n builder
588       in
589       let (_, ex) = se in
590       let local_var = match ex with
591           SClosure(clsr) -> alloc_clsr clsr
592         | _ -> L.build_malloc (ltype_of_styp t) n builder
593       in
594       let m' = StringMap.add n (t, local_var) m  in
595       let _ = L.build_store se' local_var builder in
596       (builder, m')
597     (* The order that we create and add the basic blocks for an If statement
598     doesnt 'really' matter (seemingly). What hooks them up in the right order
599     are the build_br functions used at the end of the then and else blocks (if
600     they don't already have a terminator) and the build_cond_br function at
```

```
601      the end, which adds jump instructions to the "then" and "else"
602      basic blocks *)
603      | SIf (pred, then_stmts, else_stmts) ->
604        let bool_val = expr builder m pred in
605        (* Add "merge" basic block to our function's list of blocks *)
606        let merge_bb = L.append_block context "merge" the_function in
607        (* Partial function used to generate branch to merge block *)
608        let branch_instr = L.build_br merge_bb in
609
610        (* Same for "then" basic block *)
611        let then_bb = L.append_block context "then" the_function in
612        (* Position builder in "then" block and build the statement *)
613        let (then_builder, _) = stmt (L.builder_at_end context then_bb) m
614        then_stmts in
615        (* Add a branch to the "then" block (to the merge block)
616        * if a terminator doesn't already exist for the "then" block
617        *)
618        let () = add_terminal then_builder branch_instr in
619
620        (* Identical to stuff we did for "then" *)
621        let else_bb = L.append_block context "else" the_function in
622        let (else_builder, _) = stmt (L.builder_at_end context else_bb) m
623        else_stmts in
624        let () = add_terminal else_builder branch_instr in
625
626        (* Generate initial branch instruction perform the selection of "then"
627         or "else". Note we're using the builder we had access to at the start
628         of this alternative. *)
629        let _ = L.build_cond_br bool_val then_bb else_bb builder in
630        (L.builder_at_end context merge_bb, m)
```

```
    | SReturn e ->
      let _ = match lfexpr.lreturn_typ with
          SVoid -> L.build_ret_void builder
        | _ -> L.build_ret (expr builder m e) builder
      in (builder, m)
    | SWhile (predicate, body) ->
      (* First create basic block for condition instructions -- this will
      serve as destination in the case of a loop *)
      let pred_bb = L.append_block context "while" the_function in
      (* In current block, branch to predicate to execute the condition *)
      let _ = L.build_br pred_bb builder in

      (* Create the body's block, generate the code for it, and add a branch
      back to the predicate block (we always jump back at the end of a while
      loop's body, unless we returned or something) *)
      let body_bb = L.append_block context "while_body" the_function in
      let (while_builder, _) = stmt (L.builder_at_end context body_bb) m body in
      let () = add_terminal while_builder (L.build_br pred_bb) in

      (* Generate the predicate code in the predicate block *)
      let pred_builder = L.builder_at_end context pred_bb in
      let bool_val = expr pred_builder m predicate in

      (* Hook everything up *)
      let merge_bb = L.append_block context "merge" the_function in
      let _ = L.build_cond_br bool_val body_bb merge_bb pred_builder in
      (L.builder_at_end context merge_bb, m)
    (* Implement for loops as while loops! *)
    | SFor (init, e2, e3, body) -> stmt builder m
          ( SBlock [init ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
```

```
    | SNostmt -> (builder, m)
    (*| _ -> raise (Failure "stmt not implemented in codegen")*)

  in

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

  (* add a return if the last block falls off the end *)
  add_terminal builder (match lfexpr.lreturn_typ with
      SVoid -> L.build_ret_void
    | SString -> L.build_ret (L.build_global_stringptr "" "str" builder)
    | SFloat -> L.build_ret (L.const_float_of_string float_t "0.0")
    | t -> L.build_ret (L.const_int (ltype_of_styp t) 0))

in
List.iter build_function_body functions;
the_module

```

lol.ml
Authors: David

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

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./lol.native [-a|-s|-l|-c] [file.lol]" 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
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ ->
    let sast = Semant.check ast in
    let lsast = Lift.lift sast in
    match !action with
      Ast     -> print_string (Ast.string_of_program ast)
    | Sast    -> print_string (Sast.string_of_sprogram sast)
```

```
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate lsast))
    | Compile -> let m = Codegen.translate lsast in
      Llvm_analysis.assert_valid_module m;
      print_string (Llvm.string_of_llmodule m)

```

lift.ml
Authors: David

```
open Sast

module StringMap = Map.Make(String)

(* Some types for lifting *)
type environment = {
  variables: styp StringMap.t;
  parent: environment option;
}

(* Lifted function *)
type lfexpr = {
  lname: string;
  lfvs: sbind list; (* Free variables *)
  lreturn_typ: styp;
  lparams: sbind list;
  lbody: sstmt list;
}

(* For inbuilt functions, create empty function types *)
let built_in_decls = List.fold_left (fun map (name,ty)
-> StringMap.add name ty map) StringMap.empty Builtins.builtins

(* Look up function: traverse up the tree until we encounter a symbol *)
let rec lookup (e : environment) name =
  try
    StringMap.find name e.variables
  with Not_found -> match e.parent with
      Some(parent) -> lookup parent name
    | None ->
```

```
31        try StringMap.find name built_in_decls
32        with Not_found -> raise (Failure ("Lift: undeclared identifier " ^ name))
33
34  let add_bind m (t, id) = StringMap.add id t m
35
36  let rec dfs_sstmt funcs env sstmt =
37    (* Perform dfs of statements.
38     * HANDLING SCOPE:
39     * When we pass scope to a block,
40     * the block can see all the variables in the current scope,
41     * but its local variables are invisible to us, so we simply
42     * pass in our environment and ignore the environment that
43     * the block generates.
44     * The only time where we actually change our environment is
45     * when we encounter a declaration.
46     * ---
47     * HANDLING FREE VARIABLES:
48     * The dfs functions return the free variables from the statement
49     * or expression being explored. These free variables were not in
50     * the immediate scope for them, but it might be in scope for us --
51     * consider the case when there is a function expression nested within
52     * a function expression. Therefore we have to check if the variables
53     * are indeed not in our immediate scope.
54     * ---
55     * Note that in general scopes persist, the only case where we get a
56     * actual new scope is when we get a function expression, where we need
57     * to create a new scope for that function expression to detect free
58     * variables. *)
59      let (funcs', fvs', env', sstmt') =
60      match sstmt with
```

```
61          SBlock (stmts) ->
62          let (funcs1, fvs1, _, stmts1) = dfs_sstmts funcs env stmts
63          in (funcs1, fvs1, env, SBlock(stmts1))
64        | SDecl(styp, id, sexpr) ->
65          let (funcs1, fvs1, sexpr1) = dfs_sexpr funcs env sexpr ~fname:id in
66          let (styp1, _) = sexpr1 in
67          let new_typ = match (styp, styp1) with
68              SFunc(_), SFunc(_) -> styp1
69            | _ -> styp
70          in
71          let new_env = {variables = StringMap.add id new_typ env.variables;
72                         parent = env.parent}
73          in
74          (funcs1, fvs1, new_env, SDecl(new_typ, id, sexpr1))
75        | SReturn e ->
76          let (funcs1, fvs1, e1) = dfs_sexpr funcs env e in
77          (funcs1, fvs1, env, SReturn(e1))
78        | SIf(e, s1, s2) ->
79          let (funcs1, fvs1, e') = dfs_sexpr funcs env e in
80          let (funcs2, fvs2, _, s1') = dfs_sstmt funcs1 env s1 in
81          let (funcs3, fvs3, _, s2') = dfs_sstmt funcs2 env s2 in
82          (funcs3, List.concat [fvs1; fvs2; fvs3], env, SIf(e', s1', s2'))
83        | SExpr e ->
84          let (funcs1, fvs1, e1) = dfs_sexpr funcs env e in
85          (funcs1, fvs1, env, SExpr(e1))
86        | SFor (init, e2, e3, body) ->
87          let (funcs1, fvs1, env1, init') = dfs_sstmt funcs env init in
88          let (funcs2, fvs2, e2') = dfs_sexpr funcs1 env1 e2 in
89          let (funcs3, fvs3, e3') = dfs_sexpr funcs2 env1 e3 in
90          let (funcs4, fvs4, _, body') = dfs_sstmt funcs3 env1 body in
```

```
91          (funcs4, List.concat [fvs1; fvs2; fvs3; fvs4], env,
92          SFor(init', e2', e3', body'))
93        | SWhile(e, s) ->
94          let (funcs1, fvs1, e') = dfs_sexpr funcs env e in
95          let (funcs2, fvs2, _, s') = dfs_sstmt funcs1 env s in
96          (funcs2, List.concat [fvs1; fvs2], env, SWhile(e', s'))
97        | SNostmt -> (funcs, [], env, SNostmt)
98        (*| _ -> print_endline(string_of_sstmt sstmt);
99        raise (Failure "not implemented in lifter")*)
100     in
101     let check_scope (_, fv) = not (StringMap.mem fv env.variables) in
102     let fvs' = List.filter check_scope fvs' in
103     (funcs', fvs', env', sstmt')
104
105 and dfs_sstmts funcs env sstmts = match sstmts with
106     sstmt :: rest ->
107     let (funcs1, fvs1, env1, sstmts1) = dfs_sstmt funcs env sstmt in
108     let new_env = { variables = List.fold_left add_bind env1.variables fvs1;
109                     parent = env1.parent}
110     in
111     let (funcs2, fvs2, env2, sstmts2) = dfs_sstmts funcs1 new_env rest in
112     (funcs2, List.concat [fvs1; fvs2], env2, sstmts1::sstmts2)
113   | [] -> (funcs, [], env, sstmts)
114
115 and dfs_sexpr ?fname funcs env (t, expr) =
116   let check_scope (_, fv) = not (StringMap.mem fv env.variables) in
117   let (funcs', fvs', expr') = match expr with
118       SFExpr(sfexpr) ->
119       (* Build a closure from the function expression
120       * and return the list of free variables from
```

```
121         * the function expression. Then do another check to
122         * see if the free variables from the scope of the function
123         * expression are still in the current scope. If not, then
124         * add that variable to the free variable list of the current
125         * scope *)
126         let (funcs1, fvs1, (st, clsr)) = match fname with
127             Some x -> build_closure funcs env sfexpr ~fname:x
128           | None -> build_closure funcs env sfexpr
129         in
130         let fvs1' = List.filter check_scope fvs1 in
131         (funcs1, fvs1', (st, SClosure(clsr)))
132       | SAssign(se1, op, se2) ->
133         let (funcs1, fvs1, se1') = dfs_sexpr funcs env se1 in
134         let (funcs2, fvs2, se2') = dfs_sexpr funcs1 env se2 in
135         (funcs2, List.concat [fvs1; fvs2], (t, SAssign(se1', op, se2')))
136       | SId(sid) -> let fv =
137           if StringMap.mem sid env.variables || StringMap.mem sid built_in_decls
138           then []
139           else [lookup env sid, sid]
140         in
141         (funcs, fv, (t, SId(sid)))
142       | SBinop(se1, op, se2) ->
143         let (funcs1, fvs1, se1') = dfs_sexpr funcs env se1 in
144         let (funcs2, fvs2, se2') = dfs_sexpr funcs1 env se2 in
145         (funcs2, List.concat [fvs1; fvs2], (t, SBinop(se1', op, se2')))
146       | SUnop(op, se) ->
147         let (funcs1, fvs1, se') = dfs_sexpr funcs env se in
148         (funcs1, fvs1, (t, SUnop(op, se')))
149       | SCall((lt, se), args) ->
150         (match se with
```

```
151          SId(s1) -> let fv' =
152            if StringMap.mem s1 env.variables || StringMap.mem s1 built_in_decls
153            then None
154            else Some(lookup env s1, s1)
155          in
156          let (funcs1, fvs1, args') = dfs_sexprs funcs env (args) in
157          let fvs' = match fv' with
158              Some(x) -> x :: fvs1
159            | _ -> fvs1
160          in (funcs1, fvs', (t, SCall((lt, se), args')))
161        | _ -> (* Need this for recursion. *)
162          let (funcs1, fvs1, _) = dfs_sexpr funcs env (lt, se) in
163          let (funcs2, fvs2, args') = dfs_sexprs funcs1 env args in
164          (funcs2, fvs1@fvs2, (t, SCall((lt, se), args'))))
165      | _ as x -> (funcs, [], (t, x))
166    in
167    let fvs' = List.filter check_scope fvs' in
168    (funcs', fvs', expr')
169
170  and dfs_sexprs funcs env sexprs= match sexprs with
171      sexpr :: rest ->
172      let (funcs1, fvs1, sexpr1) = dfs_sexpr funcs env sexpr in
173      let new_env = {
174        variables = List.fold_left add_bind env.variables fvs1;
175        parent = env.parent;
176      } in
177      let (funcs2, fvs2, rest) = dfs_sexprs funcs1 new_env rest in
178      (funcs2, List.concat [fvs1; fvs2], sexpr1 :: rest)
179    | [] -> (funcs, [], sexprs)
180
```

```
181 and build_closure ?fname funcs env fexpr =
182   let vars = List.fold_left add_bind StringMap.empty fexpr.sparams in
183   let name = match fname with Some x -> x | None -> "" in
184   let vars_rec = match name with
185       "" -> vars
186     | _ -> StringMap.add name SABSTRACT vars in
187   let new_env = { variables = vars_rec; parent = Some env } in
188   let (funcs', fvs, _, body') = dfs_sstmts funcs new_env fexpr.sbody in
189   let clsr = { ind = List.length funcs'; free_vars = fvs; } in
190   let new_func = {
191     lname = name;
192     lfvs = fvs;
193     lreturn_typ = fexpr.styp;
194     lparams = fexpr.sparams;
195     lbody = body'
196   } in
197   let func_t = {
198     sparam_typs = List.map fst fexpr.sparams;
199     sreturn_typ = fexpr.styp;
200   } in
201   (new_func :: funcs', fvs, (SFunc(func_t), clsr))
202
203 (* Lift takes a list of sast stmts, and converts to a list of (fname, func) *)
204 (* sstmt list -> (string * lfunc) list *)
205 let lift sstmts =
206   let default_env = { variables = StringMap.empty; parent = None } in
207   let (funcs, _, _, sstmts') = dfs_sstmts [] default_env sstmts in
208   let main_func = {
209     lname = "main";
210     lfvs = [];
```

```
    lreturn_typ = SInt;
    lparams = [];
    lbody = sstmts'
  } in
  let name i func = ("f" ^ string_of_int i, func) in
  let named_funcs = List.mapi name (List.rev funcs) in
  (("main", main_func) :: named_funcs)

(* Pretty print *)
let fmt_lfunc f = String.concat "\n" [
    " -fvs: " ^ String.concat ""
      (List.map (fun (t, n) -> (string_of_styp t) ^ " " ^ n) f.lfvs);
    " -return_t: " ^ string_of_styp f.lreturn_typ;
    " -params: " ^ String.concat ""
      (List.map (fun (t, n) -> (string_of_styp t) ^ " " ^ n) f.lparams);
    " -lbody: \n" ^ string_of_list_sstmt f.lbody ", ";
  ]

let helper (name, f) = name ^ ":\n" ^ (fmt_lfunc f)

let rec string_of_lsast = function
    [] -> ""
| item :: rest -> String.concat "\n" [(helper item);(string_of_lsast rest)]

```

builtins.c
Authors: Jerry, David, Shuqi, Zichuan

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

// GSL header files
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_sf.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_permutation.h>

#define MAXFLOATSIZE 50

typedef struct List_element {
    struct List_element *next;
    struct List_element *prev;
    void *data;
} list_element;

typedef struct List {
    int32_t length;
    list_element *head;
    list_element *tail;
} list;

/*
==========================================================
                    LIST FUNCTIONS
==========================================================
```

```
32  */
33  //init list
34  list* list_init() {
35      list* l = (list *) malloc(sizeof(list));
36      l->length = 0;
37      l->head = NULL;
38      l->tail = NULL;
39      return l;
40  }
41
42  //access element
43  void* list_get(list *l, int index) {
44      if(index >= l->length) {
45          return NULL;
46      }
47      list_element *curr = l->head;
48      int count = 0;
49      while(count < index) {
50          curr = curr->next;
51          count++;
52      }
53
54      return curr->data;
55  }
56
57  //set element
58  list_element* list_set(list *l, void *data, int index) {
59      if(index >= l->length) {
60          return NULL;
```

```
61      }
62      list_element *curr = l->head;
63      int count = 0;
64      while(count < index) {
65          curr = curr->next;
66          count++;
67      }
68      free(curr->data);
69      curr->data = data;
70      return curr;
71  }
72
73  //append element: can be any type
74  void list_append(list *l, void *data) {
75      list_element *le = (list_element*) malloc(sizeof(list_element));
76      le->data = data;
77      le->next = NULL;
78      le->prev = l->tail;
79      if(l->tail) {
80          l->tail->next = le;
81      }
82      else { //empty list
83          l->head = le;
84      }
85      l->tail = le;
86      l->length += 1;
87
88      //return le;
89  }
90
```

```
91  //concatenate lists
92  list* list_concat(list *l1, list *l2) {
93      list *new_list = list_init();
94      list_element *curr = l1->head;
95      while(curr) {
96          list_append(new_list, curr->data);
97          curr = curr->next;
98      }
99      curr = l2->head;
100     while(curr) {
101         list_append(new_list, curr->data);
102         curr = curr->next;
103     }
104     return new_list;
105 }
106
107 /* assumes that the list contains the element */
108 void list_remove(list *l, list_element *e) {
109     if(l->head == e) {
110         l->head = e->next;
111     }
112     else {
113         e->prev->next = e->next;
114     }
115     if(l->tail == e) {
116         l->tail = e->prev;
117     }
118     else {
119         e->next->prev = e->prev;
120     }
```

```
121     l->length--;
122     free(e);
123 }
124
125 int32_t list_length(list *l) {
126     return l->length;
127 }
128
129 // assumes the str that's based in is correctly null terminated
130 void println(char * str){
131   printf("%s\n",str);
132   return;
133 }
134
135 // assumes the str that's based in is correctly null terminated
136 void print(char * str){
137   printf("%s",str);
138   return;
139 }
140
141 char* str_of_int(int x) {
142   int length = snprintf( NULL, 0, "%d", x );
143   char* str = malloc( length + 1 );
144   snprintf( str, length + 1, "%d", x );
145   return str;
146 }
147
148 char* str_of_bool(int x) {
149   // int length;
150   char* str;
```

```
151   if (x) {
152     str = malloc(sizeof("true") + 1);
153     snprintf(str, sizeof("true") + 1, "%s", "true");
154   } else {
155     str = malloc(sizeof("false") + 1);
156     snprintf(str, sizeof("false") + 1, "%s", "false");
157   }
158   return str;
159 }
160
161 char* string_concat(char * str1, char* str2) {
162   int totalLength = strlen(str1) + strlen(str2) + 1;
163   char* result = calloc( totalLength, 1 );
164   strcat(result, str1);
165   strcat(result, str2);
166   return result;
167 }
168
169 int string_equals(char * str1, char* str2) {
170   int res = strcmp(str1, str2);
171   if (res == 0) {
172     return 1;
173   }
174   return 0;
175 }
176
177 //BUT TECHNICALLY IT"S A C DOUBLE TYPE NOT C FLOAT TYPE
178 char* str_of_float(double x) {
179   char* str;
180   str = (char *) malloc( sizeof(char) * MAXFLOATSIZE);
```

```c
  snprintf(str, MAXFLOATSIZE, "%g", x);
  return str;
}

// rounds float to nearest int
int int_of_float(double x){
  return (int) round(x);
}

double float_of_int(int x){
  return x * 1.0;
}

// returns -1 if the string is not a valid integer.
// expects a null terminated string -- if the string is not
// null terminated, undefined behavior will occur.
// If the integer is beyond INT_MAX as defined by C, that is, 2147483647,
// overflow may occur and the result is undefined.
int int_of_str(char *str){
  // reference: https://www.geeksforgeeks.org/write-your-own-atoi/
  int res = 0;  // Initialize result
  int sign = 1;  // Initialize sign as positive
  int i = 0;  // Initialize index of first digit
  // If number is negative, then update sign
  if (str[0] == '-'){
    sign = -1;
    i++;  // Also update index of first digit
  }

  // Iterate through all digits and update the result
```

```
211   for (; str[i] != '\0'; ++i) {
212     if(str[i]<48 || str[i]>57) {
213        // not a digit
214        return -1;
215     }
216     res = res*10 + str[i] - '0';
217   }
218   // Return result with sign
219   return sign*res;
220 }
221
222 /*
223 ==========================================================
224                         Math
225 ==========================================================
226 */
227
228 double exp(double x) {
229   return gsl_sf_exp(x);
230 }
231
232 double log(double x) {
233   return gsl_sf_log(x);
234 }
235
236 /*
237 ==========================================================
238                         Matrices
239 ==========================================================
240 */
```

```c
/*
The following is an attempt to create an interface between GSL and OCaml.
We will see if we can provide simplified version of GSL functions here for OCaml
to call and store the GSL objects.

To-do:
1. calling native GSL functions
*/

int printm(const gsl_matrix *m)
{
  int status, n = 0;

  for (size_t i = 0; i < m->size1; i++) {
    printf("||\t");
    for (size_t j = 0; j < m->size2; j++) {
      if ((status = printf("%g\t", gsl_matrix_get(m, i, j))) < 0)
        return -1;
      n += status;
    }
    printf("||");

    if ((status = printf("\n")) < 0)
      return -1;
    n += status;
  }
  printf("\n");

  return n;
}
```

```
271
272 // initializes an empty matrix with a certain number of rows and columns
273 gsl_matrix * minit(size_t m, size_t n) {
274   // allocate matrix memory
275   return gsl_matrix_calloc(m, n);
276 }
277
278 int mrow(gsl_matrix * m){
279   return m->size1;
280 }
281
282 int mcol(gsl_matrix * m){
283   return m->size2;
284 }
285
286 double mmax(gsl_matrix * m) {
287   return gsl_matrix_max(m);
288 }
289
290 // double mdet(gsl_matrix * m) {
291 //   return gsl_linalg_LU_det(m, 1);
292 // }
293
294 double mdet(gsl_matrix * A) {
295   double det;
296   int signum;
297   gsl_permutation *p = gsl_permutation_alloc(A->size1);
298
299   gsl_matrix *tmpA = gsl_matrix_alloc(A->size1, A->size2);
300   gsl_matrix_memcpy(tmpA , A);
```

```
301
302   gsl_linalg_LU_decomp(tmpA , p, &signum);
303   det = gsl_linalg_LU_det(tmpA , signum);
304   gsl_permutation_free(p);
305   gsl_matrix_free(tmpA);
306
307   return det;
308 }
309
310 // get matrix element
311 // double gsl_matrix_get(const gsl_matrix * m, const size_t i, const size_t j)
312 double mget(const gsl_matrix * m, const size_t i, const size_t j) {
313   return gsl_matrix_get(m, i, j);
314 }
315
316 // set matrix element
317 // void gsl_matrix_set(gsl_matrix * m, const size_t i, const size_t j, double x)
318 double mset(gsl_matrix * m, const size_t i, const size_t j, double x) {
319   //printf("<%d,%d>\n",m->size1,m->size2);
320   //printf("%d,%d\n",i,j);
321   gsl_matrix_set(m, i, j, x);
322   //print_matrix(m);
323   return x;
324 }
325
326 // Add. sub. mul. div
327 int madd(gsl_matrix * a, const gsl_matrix * b) {
328   return gsl_matrix_add(a, b);
329 }
330
```

```c
int msub(gsl_matrix * a, const gsl_matrix * b) {
  return gsl_matrix_sub(a, b);
}

int mmulc(gsl_matrix * a, const double x) {
  return gsl_matrix_scale(a, x);
}

int maddc(gsl_matrix * a, const double x) {
  return gsl_matrix_add_constant(a, x);
}

int mmule(gsl_matrix * a, const gsl_matrix * b) {
  return gsl_matrix_mul_elements(a, b);
}

int mdive(gsl_matrix * a, const gsl_matrix * b) {
  return gsl_matrix_div_elements(a, b);
}

int mexpe(gsl_matrix * a) {
  for (int i = 0; i < a->size1; i++) {
    for (int j = 0; j < a->size2; j++) {
      gsl_matrix_set(a, i, j, gsl_sf_exp(gsl_matrix_get(a, i, j)));
    }
  }
  return 0;
}

int mloge(gsl_matrix * a) {
```

```
  for (int i = 0; i < a->size1; i++) {
    for (int j = 0; j < a->size2; j++) {
      gsl_matrix_set(a, i, j, gsl_sf_log(gsl_matrix_get(a, i, j)));
    }
  }
  return 0;
}

// swap rows and columns, transpose, copy
int mswapr(gsl_matrix * m, size_t i, size_t j) {
  return gsl_matrix_swap_rows(m, i, j);
}

int mswapc(gsl_matrix * m, size_t i, size_t j) {
  return gsl_matrix_swap_columns(m, i, j);
}

gsl_matrix * mtrans(gsl_matrix * m) {
  gsl_matrix * newMatrix = gsl_matrix_calloc(m->size2, m->size1);
  gsl_matrix_transpose_memcpy(newMatrix, m);
  return newMatrix;
}

gsl_matrix * mcopy(gsl_matrix * m) {
  gsl_matrix * newMatrix = gsl_matrix_calloc(m->size1, m->size2);
  gsl_matrix_memcpy(newMatrix, m);
  return newMatrix;
}

// Matrix view
```

```c
gsl_matrix * mgetr(gsl_matrix * m, size_t row) {
  gsl_matrix * newMatrix = gsl_matrix_calloc(1, m->size2);
  gsl_matrix_view v = gsl_matrix_submatrix(m, row, 0, 1, m->size2);
  gsl_matrix_memcpy(newMatrix, &v.matrix);
  return newMatrix;
}

gsl_matrix * mgetc(gsl_matrix * m, size_t col) {
  gsl_matrix * newMatrix = gsl_matrix_calloc(m->size1, 1);
  gsl_matrix_view v = gsl_matrix_submatrix(m, 0, col, m->size1, 1);
  gsl_matrix_memcpy(newMatrix, &v.matrix);
  return newMatrix;
}

gsl_matrix * mgetsub(gsl_matrix * m, size_t x1, size_t y1, size_t x2, size_t y2)
{
  if (x1 > x2) {
    printf("Invalid x1 or x2! Expected x1 is less than or equal to x2.\n");
    return NULL;
  }
  if (y1 > y2) {
    printf("Invalid y1 or y2! Expected y1 is less than or equal to y2.\n");
    return NULL;
  }
  int n1 = x2 - x1 + 1;
  int n2 = y2 - y1 + 1;
  gsl_matrix * newMatrix = gsl_matrix_alloc(n1, n2);
  gsl_matrix_view v = gsl_matrix_submatrix(m, x1, y1, n1, n2);
  gsl_matrix_memcpy(newMatrix, &v.matrix);
  return newMatrix;
```

```
421 }
422
423 // BLAS
424 double mdot(gsl_matrix * m1, gsl_matrix * m2) {
425   // Both matrices need to have same number of columns and one row
426   double* result = malloc(sizeof(float));
427   gsl_vector_view v1_view = gsl_matrix_row(m1, 0);
428   gsl_vector_view v2_view = gsl_matrix_row(m2, 0);
429   gsl_vector * v1 = gsl_vector_alloc(v1_view.vector.size);
430   gsl_vector * v2 = gsl_vector_alloc(v2_view.vector.size);
431   gsl_vector_memcpy(v1, &v1_view.vector);
432   gsl_vector_memcpy(v2, &v2_view.vector);
433   gsl_blas_ddot(v1, v2, result);
434   double res = *result;
435   free(result);
436   free(v1);
437   free(v2);
438   return res;
439 }
440
441 gsl_matrix * mmul ( gsl_matrix * m1, gsl_matrix * m2) {
442   if (m1->size2 != m2->size1) {
443     printf("Matrix dimensions do not match!\n");
444     return NULL;
445   }
446   gsl_matrix * newMatrix = gsl_matrix_calloc(m1->size1, m2->size2);
447   gsl_blas_dgemm(CblasNoTrans, CblasNoTrans, 1.0, m1, m2, 0.0, newMatrix);
448   return newMatrix;
449 }
450
```

```
451 // pipe the operator into SCall to a builtin
452 // M + N => SCall (concat, M, N) concat \in builtin
453 // keep Tensor as a class
454 // use bitcode to translate
455
```

builtins.ml
Authors: David, Jerry

```
(* Builtin function declarations as SFunc(sfunc_typ) *)

open Sast

module StringMap = Map.Make (String)

let builtins = [

  (* Printing *)
  ("print", SFunc({sparam_typs = [SString]; sreturn_typ = SVoid}));
  ("println", SFunc({ sparam_typs = [SString]; sreturn_typ = SVoid }));
  ("printm", SFunc ({sparam_typs = [SMatrix(0, 0)]; sreturn_typ = SVoid}));

  (* Casting *)
  ("int_of_float", SFunc({ sparam_typs = [SFloat]; sreturn_typ = SInt }));
  ("float_of_int", SFunc({ sparam_typs = [SInt]; sreturn_typ = SFloat }));

  (* String *)
  ("str_of_int", SFunc({ sparam_typs = [SInt]; sreturn_typ = SString }));
  ("int_of_str", SFunc({ sparam_typs = [SString]; sreturn_typ = SInt }));
  ("str_of_bool", SFunc({ sparam_typs = [SBool]; sreturn_typ = SString }));
  ("str_of_float", SFunc({ sparam_typs = [SFloat]; sreturn_typ = SString }));
  ("string_concat", SFunc({ sparam_typs = [SString; SString]; sreturn_typ = SString }));
  ("string_equals", SFunc({ sparam_typs = [SString; SString]; sreturn_typ = SInt }));

  (* List functions *)
  ("list_init", SFunc({sparam_typs = []; sreturn_typ = SList (SAny) }));
  ("list_append", SFunc({sparam_typs = [SList (SAny); SAny]; sreturn_typ = SVoid }));
  ("list_get", SFunc({sparam_typs = [SList (SAny); SInt]; sreturn_typ = SListElement (SAny) }));
  ("list_length", SFunc({sparam_typs= [SList (SAny)]; sreturn_typ = SInt }));
```

```
  ("list_set", SFunc({sparam_typs= [SList (SAny); SAny ; SInt]; sreturn_typ = SAny }));

  (* Math *)
  ("exp", SFunc({sparam_typs = [SFloat]; sreturn_typ = SFloat }));
  ("log", SFunc({sparam_typs = [SFloat]; sreturn_typ = SFloat }));

  (* Matrix functions *)
  (* Basic *)
  ("minit", SFunc({sparam_typs = [SInt; SInt]; sreturn_typ = SMatrix(0,0) }));
  ("mget", SFunc({sparam_typs = [SMatrix(0, 0); SInt; SInt]; sreturn_typ = SFloat }));
  ("mset", SFunc({sparam_typs = [SMatrix(0, 0); SInt; SInt; SFloat]; sreturn_typ = SFloat }));
  (* Matrix Property *)
  ("mrow", SFunc({sparam_typs = [SMatrix(0,0)]; sreturn_typ = SInt}));
  ("mcol", SFunc({sparam_typs = [SMatrix(0,0)]; sreturn_typ = SInt}));
  ("mmax", SFunc({sparam_typs = [SMatrix(0,0)]; sreturn_typ = SFloat}));
  ("mdet", SFunc({sparam_typs = [SMatrix(0,0)]; sreturn_typ = SFloat}));
  (* Add, sub, mul, div *)
  ("madd", SFunc({sparam_typs = [SMatrix(0, 0); SMatrix(0, 0)]; sreturn_typ = SVoid }));
  ("msub", SFunc({sparam_typs = [SMatrix(0, 0); SMatrix(0, 0)]; sreturn_typ = SVoid }));
  ("mmulc", SFunc({sparam_typs = [SMatrix(0, 0); SFloat]; sreturn_typ = SVoid }));
  ("maddc", SFunc({sparam_typs = [SMatrix(0, 0); SFloat]; sreturn_typ = SVoid }));
  ("mmule", SFunc({sparam_typs = [SMatrix(0, 0); SMatrix(0, 0)]; sreturn_typ = SVoid }));
  ("mdive", SFunc({sparam_typs = [SMatrix(0, 0); SMatrix(0, 0)]; sreturn_typ = SVoid }));
  ("mexpe", SFunc({sparam_typs = [SMatrix(0, 0)]; sreturn_typ = SVoid }));
  ("mloge", SFunc({sparam_typs = [SMatrix(0, 0)]; sreturn_typ = SVoid }));
  (* Swap rows & cols, transpose *)
  ("mswapr", SFunc({sparam_typs = [SMatrix(0, 0); SInt; SInt]; sreturn_typ = SVoid }));
  ("mswapc", SFunc({sparam_typs = [SMatrix(0, 0); SInt; SInt]; sreturn_typ = SVoid }));
  ("mtrans", SFunc({sparam_typs = [SMatrix(0, 0)]; sreturn_typ = SMatrix(0,0) }));
  ("mcopy", SFunc({sparam_typs = [SMatrix(0, 0)]; sreturn_typ = SMatrix(0,0) }));
```

```
61    (* Matrix view *)
62    ("mgetr", SFunc({sparam_typs = [SMatrix(0, 0); SInt]; sreturn_typ = SMatrix(0,0) }));
63    ("mgetc", SFunc({sparam_typs = [SMatrix(0, 0); SInt]; sreturn_typ = SMatrix(0,0) }));
64    ("mgetsub", SFunc({sparam_typs = [SMatrix(0, 0); SInt; SInt; SInt; SInt]; sreturn_typ = SMatrix(0,0) }))
65    (* BLAS *)
66    ("mdot", SFunc({sparam_typs = [SMatrix(0, 0); SMatrix(0, 0)]; sreturn_typ = SFloat }));
67    ("mmul", SFunc({sparam_typs = [SMatrix(0, 0); SMatrix(0, 0)]; sreturn_typ = SMatrix(0, 0) }));
68
69    (* Advanced binop*)
70    ("pow", SFunc({ sparam_typs = [SFloat; SFloat]; sreturn_typ = SFloat }));
71
72  ]
73
```

8.2 test Files

testall.sh
Authors: Shuqi, Zichuan

```
#!/bin/bash

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

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

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
GCC="gcc"

# Path to the lol compiler.  Usually "./lol.native"
# Try "_build/lol.native" if ocamlbuild was unable to create a symbolic link.
LOL="./lol.native"

# Set time limit for all operations
ulimit -t 30

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

passed=0
total=0
keep=0

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

SignalError() {
```

```
    if [ $error -eq 0 ] ; then
            echo "FAILED"
            error=1
    fi
    echo "  $1"
}

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

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

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

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\//
                        s/.lol//'`
    reffile=`echo $1 | sed 's/.lol$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
```

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

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

   generatedfiles=""

   generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
   Run "$LOL" "$1" ">" "${basename}.ll" &&
   Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
   # Run "$CC" "-o" "${basename}.exe" "${basename}.s" "builtins.o" "-lm" &&
   Run "$GCC" "-o" "${basename}" "${basename}.s" "-L/usr/local/lib" "builtins.o" "-lgsl
-lgslcblas -lm" &&
   Run "./${basename}" > "${basename}.out" &&
   Compare ${basename}.out ${reffile}.out ${basename}.diff

   # Report the status and clean up the generated files

   total=$((total+1))

   if [ $error -eq 0 ] ; then
         if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
         fi
         echo "OK"
         echo "###### SUCCESS" 1>&2
   passed=$((passed+1))
   else
         echo "###### FAILED" 1>&2
         globalerror=$error
   fi
}

CheckFail() {
   error=0
   basename=`echo $1 | sed 's/.*\\//
                        s/.lol//'`
   reffile=`echo $1 | sed 's/.lol$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

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

```
    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$LOL" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    total=$((total+1))

    if [ $error -eq 0 ] ; then
            if [ $keep -eq 0 ] ; then
               rm -f $generatedfiles
            fi
            echo "OK"
            echo "###### SUCCESS" 1>&2
    passed=$((passed+1))
    else
            echo "###### FAILED" 1>&2
            globalerror=$error
    fi
}

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

shift `expr $OPTIND - 1`

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

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


if [ $# -ge 1 ]
```

```
then
   files=$@
else
   files="tests/test-*.lol tests/fail-*.lol"
fi

for file in $files
do
   case $file in
         *test-*)
            Check $file 2>> $globallog
            ;;
         *fail-*)
            CheckFail $file 2>> $globallog
            ;;
         *)
            echo "unknown file type $file"
            globalerror=1
            ;;
   esac
done
echo $passed" out of "$total" tests passed!"
exit $globalerror
```

/tests
*.lol
Authors: Shuqi, Zichuan

fail-assign1.lol

```
int i;
bool b;

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

fail-elif-without-if.lol

```
float b = 0.23;

elif (b < 0) {
   println("what?");
} else {
   println("how?");
}

println("nothing");
```

fail-else-without-if.lol

```
float b = 0.23;

else {
   println("how?");
}

println("nothing");
```

fail-list-access-expr.lol

```
array<string> a = ["hi", "bye"];
a[1.234];
```

fail-list-void.lol

```
List<void> a;
```

fail-loops-init.lol

```
// failiure because while only takes boolean in field
```

```
while(1){
        println("hello");
}
```

test-AND.lol

```
bool h = true;
bool i = true;
bool j = false;

bool t1 = h AND i;
bool t2 = h AND j;

if (t1) {
        println("T AND T is T");
} else {
        println("T AND T is F");
}

if (t2) {
        println("T AND F is T");
} else {
        println("T AND F is F");
}
```

test-assign1.lol

```
int i = 2;
bool b = true;
float k = 3.14;

i = 42;
i = 10;
b = true;
b = false;
k = 0.0;
println("OK");
```

test-binop-eq.lol

```
int i = 0;
int j = 0;
if (i == j) {
        println("LOL");
}
```

test-binop.lol

```
int x = 5+6;
println(str_of_int(x));
int y = 2*3;
println(str_of_int(y));

int me = 5/2;
println(str_of_int(me));

float hello = 4.5/2.0;
println(str_of_float(hello));
```

test-comment-block.lol

```
/* this
is
a
block
comment */

println("LOL");
```

test-comment-line.lol

```
// this is a comment line

println("LOL");
```

test-comment-nested.lol

```
/* this is a /* nested */ comment */

println("LOL");
```

test-compare-bool.lol

```
bool h = true;
bool i = true;

if (h == i) {
   println("bool works");
} else {
   println("bool doesn't work");
}

bool j = false;
if (h == j) {
   println("bool doesn't work");
} else {
   println("bool works");
}
```

test-compare-float.lol

```
float d = 1.098;
float e = 1.098;

if (d == e) {
   println("float works");
} else {
   println("float doesn't work");
}

float f = 2.45678900;
if (d == f) {
   println("float doesn't work");
} else {
   println("float works");
}
```

test-compare-int.lol

```
int a = 5;
int b = 5;

if (a == b) {
   println("int works");
} else {
   println("int doesn't work");
}
```

test-compare-neq-float.lol

```
float a = 5.1;
float b = 6.1;

if (a <= b) {
   println("int works");
} else {
   println("int doesn't work");
}

float c = 7.1;
if (c >= b) {
   println("int works");
} else {
   println("int doesn't work");
```

```
}
```

test-compare-neq-int.lol

```
int a = 5;
int b = 6;

if (a <= b) {
   println("int works");
} else {
   println("int doesn't work");
}

int c = 7;
if (c >= b) {
   println("int works");
} else {
   println("int doesn't work");
}
```

test-for-loop-missing-all.lol

```
func string infinite(int c) {
   for ( ; ; ) {
      c = c + 1;
      if (c > 100) {
         return "done!";
      }
   }

   return "what??";
}

string s = infinite(2);
println(s);
```

test-for-loop-no-incr.lol

```
for (int i = 0; i < 10;  ) {
   println("hello " + str_of_int(i));
   i = i + 2;
}
```

test-for-loop-no-init.lol

```
int i = 0;
for (; i < 5; i = i + 1) {
```

```
    println(str_of_int(i));
}
```

test-for-loop-no-predicate.lol

```
func int hi() {
    for (int k = 1; ; k = k + 7) {
        if (k > 100) {
            return 100;
        }
    }
    return 1;
}

println(str_of_int(hi()));
```

test-for-loop-return.lol

```
func string foo() {
  for (int i = 5; i < 10; i = i+1) {
    if (i == 5){
      return "hello " + str_of_int(i);
    }
  }
}

println(foo());
```

test-func-int.lol

```
func int foo (){
        println("foo");
        return 666;
}


println(str_of_int(foo()));
```

test-func-list.lol

```
func int addOne (int num){
        return num + 1;
}
```

```
List<func(int:int)> lst;
lst.append(addOne);
println(str_of_int(lst[0](1)));
```

test-func-mult-parameters.lol

```
float PI = 3.14;


func float calculate_area (float r, float pi) {
   return r * r * pi;
}

println(str_of_float(calculate_area(10.0, PI)));
```

test-func-simple-first-class.lol

```
func func(int,int:int) apply_f(func(int:int) f) {
   return func int (int i, int j) {
      return f(i) + f(j);
   };
}

func int double(int x) {
   return x * 2;
}

func(int,int:int) sum_of_double = apply_f(double);

println(str_of_int(sum_of_double(3, 5)));
```

test-func-simple-first-class2.lol

```
func func(float,float:float) apply_f(func(float:float) f) {
   return func float (float i, float j) {
      return f(i) + f(j);
   };
}

func float double(float x) {
   return x * 2.0;
}
```

```
func(float,float:float) sum_of_double = apply_f(double);

println(str_of_float(sum_of_double(3.4, 5.3)));
```

test-test-for-loop-no-incr.lol

```
for (int i = 0; i < 10;  ) {
   println("hello " + str_of_int(i));
   i = i + 2;
}
```

test-func-single-parameter.lol

```
float PI = 3.141;

func float calculate_area (float r) {
   return r * r * PI;
}

println(str_of_float(calculate_area(10.0)));
```

test-func-void.lol

```
func void foo (){
        println("Void func");
}


foo();
```

test-helloworld.lol

```
println("Hello World!");
```

test-if-else.lol

```
bool flag1 = false;
if (flag1){
        println("233");
}
else{
        println("333");
}
```

```
bool flag2 = true;
if (flag2){
	println("666");
}
else{
	println("777");
}
```

test-if-nested.lol

```
bool flag1 = false;
bool flag2 = true;
if (flag1){
	if (flag2) {
		println("both true");
	} else {
		println("flag1 true, flag2 false");
	}
}
else{
	if (flag2) {
		println("flag1 false, flag2 true");
	} else {
		println("both false");
	}
}
```

test-if-simple.lol

```
bool flag = true;
if (flag){
	println("233");
}
```

test-list-append-length.lol

```
List<int> a = [34, 235, 2534, 435];
println(str_of_int(a.length()));
a.append(123);
a.append(134);
a.append(123);
a.append(123);
println(str_of_int(a.length()));
```

test-list-append.lol

```
List<int> a = [1, 2, 3];
a.append(4);
println(str_of_int(a[3]));
```

test-list-empty.lol

```
List<int> a = [];
```

test-list-if.lol

```
func void printTrue(){
  println("true");
}
List<int> a = [34, 235, 2534, 435];
if (a[2] == 2534)
{
printTrue();
}
```

test-list-length.lol

```
List<int> a = [34, 235, 2534, 435];
println(str_of_int(a.length()));
```

test-list-of-list.lol

```
List<List<int>> foo = [[0,1],[2,3]];
println(str_of_int(foo[0][0]));
println(str_of_int(foo[0][1]));
println(str_of_int(foo[1][0]));
println(str_of_int(foo[1][1]));
```

test-list-simple.lol

```
List<int> a = [34, 235, 2534, 435];
println(str_of_int(a[2]));
```

test-matrix-dot.lol

```
Matrix t1 = Matrix([1.0, 2.0, 3.0]);
Matrix t2 = Matrix([2.0, 3.0, 4.0]);

float result = mdot(t1, t2);
println(str_of_float(result));
```

test-matrix-get.lol

```
Matrix t = Matrix([1.0, 2.0,1.0],[3.0, 4.0, 3.0],[4.0,5.0, 4.0],[6.0,7.0,6.0]);
println(str_of_float(t.get(0,1)));
```

test-matrix-init.lol

```
Matrix t1 = Matrix([5.0]);

printm(t1);
```

test-matrix-madd.lol

```
Matrix t1 = Matrix([1.1,1.1]);
Matrix t2 = Matrix([2.2,2.2]);
madd(t1, t2);

printm(t1);
```

test-matrix-maddc.lol

```
Matrix t = Matrix([1.0,2.0]);
maddc(t, 2.0);

printm(t);
```

test-matrix-mcopy.lol

```
Matrix t = Matrix([1.0,2.0]);
Matrix t2 = mcopy(t);

printm(t2);
```

test-tmatrix-mdive.lol

```
Matrix t1 = Matrix([1.1,1.1]);
Matrix t2 = Matrix([2.2,2.2]);
mdive(t1, t2);

printm(t1);
```

test-matrix-mdot.lol

```
Matrix t1 = Matrix([1.0, 2.0, 3.0]);
Matrix t2 = Matrix([2.0, 3.0, 4.0]);

float result = mdot(t1, t2);
println(str_of_float(result));
```

test-matrix-mexpe.lol

```
Matrix t = Matrix([1.0,2.0]);
mexpe(t);

printm(t);
```

test-matrix-mget.lol

```
Matrix t = Matrix([1.0, 2.0,1.0],[3.0, 4.0, 3.0],[4.0,5.0, 4.0],[6.0,7.0,6.0]);
println(str_of_float(mget(t,0,1)));
```

test-matrix-mgetc.lol

```
Matrix t1 = Matrix([1.0, 2.0, 3.0], [2.0, 3.0, 4.0]);
Matrix t3 = mgetc(t1, 1);
printm(t3);
```

test-matrix-mgetr.lol

```
Matrix t1 = Matrix([1.0, 2.0, 3.0], [2.0, 3.0, 4.0]);
Matrix t3 = mgetr(t1, 1);
printm(t3);
```

test-matrix-mgetsub.lol

```
Matrix t1 = Matrix([1.0, 2.0, 3.0], [2.0, 3.0, 4.0]);
Matrix t2 = mgetsub(t1, 0, 0, 1, 1);
printm(t2);
```

test-matrix-mloge.lol

```
Matrix t = Matrix([10.0,20.0]);
mloge(t);

printm(t);
```

test-matrix-mmulc.lol

```
Matrix t = Matrix([1.0,2.0]);
mmulc(t, 2.0);

printm(t);
```

test-matrix-mmule.lol

```
Matrix t1 = Matrix([1.1,1.1]);
Matrix t2 = Matrix([2.2,2.2]);
mmule(t1, t2);

printm(t1);
```

test-matrix-mset.lol

```
Matrix t = Matrix([1.0, 2.0,1.0],[3.0, 4.0, 3.0],[4.0,5.0, 4.0],[6.0,7.0,6.0]);
mset(t, 0, 1, 8.0);
println(str_of_float(t[0, 1]));
```

test-matrix-msub.lol

```
Matrix t1 = Matrix([1.1,1.1]);
Matrix t2 = Matrix([2.2,2.2]);
msub(t2,t1);

printm(t2);
```

test-matrix-mtrans.lol

```
Matrix t1 = Matrix([1.0, 2.0, 3.0], [2.0, 3.0, 4.0]);
Matrix t3 = mtrans(t1);
printm(t3);
```

test-matrix-set.lol

```
Matrix t = Matrix([1.0, 2.0,1.0],[3.0, 4.0, 3.0],[4.0,5.0, 4.0],[6.0,7.0,6.0]);
```

```
t.set(0, 1, 8.0);
println(str_of_float(t[0, 1]));
```

test-matrix-swap-columns.lol

```
Matrix t1 = Matrix([1.0, 2.0, 3.0], [2.0, 3.0, 4.0]);
mswapc(t1, 0, 1);
printm(t1);
```

test-matrix-swap-rows.lol

```
Matrix t1 = Matrix([1.0, 2.0, 3.0], [2.0, 3.0, 4.0]);
mswapr(t1, 0, 1);
printm(t1);
```

test-OR.lol

```
bool h = true;
bool i = true;
bool j = false;

bool t1 = h OR i;
bool t2 = h OR j;

if (t1) {
        println("T OR T is T");
} else {
        println("T OR T is F");
}

if (t2) {
        println("T OR F is T");
} else {
        println("T OR F is F");
}
```

test-power.lol

```
println(str_of_float(2.0^2.0));
```

test-print.lol

```
int a = 5;
println(str_of_int(a));
println(str_of_int(5));

float b = 1.234567;
```

```
println(str_of_float(b));
println(str_of_float(1.234567));
```

test-return-after-for.lol

```
func string foo() {
  for (int i = 5; i < 10; i = i+1) {
      int x = 5;
      x = 10;
  }
  return "Hello World";
}

println(foo());
```

test-unop.lol

```
bool x=true;
if (!x) {
 println("THIS SHOULDNT PRINT");
} else {
 println("Somebody once told me");
}

int y = -400;
if (y+400 == 0) {
 println("the world is gonna roll me");
}

float z = -4.5;
float c = -0.3;
float b = z+c;
if(b==-4.8){
  println("I ain't the sharpest tool in the shed");
}
```

test-var-scope.lol

```
int q = 7;
if (q == 7) {
    int q = 9;
    println(str_of_int(q));
} else {
    println("no");
}
```

```
int i;
for (i = 0; i < 10; i = i+1) {
   int q = 9;
         q = q + 1;
   println(str_of_int(q));
}

func int foo (int a) {
   int q = 21;
   return a;
}

int a = foo(q);

println(str_of_int(a));
println(str_of_int(q));
```

test-while-loop-int-incre.lol

```
int i = 0;
while (i < 3) {
   println(str_of_int(i));
   i++;
}
```

8.3 *Makefile*

Authors: David, Jerry

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

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

# "make all" builds the executable as well as the "printbig" library designed
# to test linking external code

.PHONY : all
all : lol.native builtins.o

.PHONY : lol.native
lol.native:
    rm -f *.o
    ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,llvm.bitreader -cflags -w,+a-4 lol.native

builtins.o :
    gcc -Wall -I/usr/local/include -c builtins.c
    clang -emit-llvm -o builtins.bc -c builtins.c -Wno-varargs


.PHONY : clean
clean :
    ocamlbuild -clean
    rm -rf lol.native scanner.ml parser.ml parser.mli
    rm -rf ./*.ll
    rm -rf *.cmx *.cmi *.cmo *.cmx *.o *.s *.bc *.out *.exe

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

```
31
32 OBJS = ast.cmx sast.cmx codegen.cmx parser.cmx scanner.cmx semant.cmx lol.cmx
33
34 lol : $(OBJS)
35     ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o lol
36
37 scanner.ml : scanner.mll
38     ocamllex scanner.mll
39
40 parser.ml parser.mli : parser.mly
41     ocamlyacc parser.mly
42
43 %.cmo : %.ml
44     ocamlc -c $<
45
46 %.cmi : %.mli
47     ocamlc -c $<
48
49 %.cmx : %.ml
50     ocamlfind ocamlopt -c -package llvm $<
51
52 ### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
53 ast.cmo :
54 ast.cmx :
55 codegen.cmo : ast.cmo
56 codegen.cmx : ast.cmx
57 lol.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
58 lol.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
59 parser.cmo : ast.cmo parser.cmi
60 parser.cmx : ast.cmx parser.cmi
```

```
61 scanner.cmo : parser.cmi
62 scanner.cmx : parser.cmx
63 semant.cmo : ast.cmo
64 semant.cmx : ast.cmx
65 parser.cmi : ast.cmo
66
```