Final Report

1. Introduction

2. Language Manual
   2.1 Data Types
      2.1.1 Primitive types (call by value)
      2.1.2 Built-in types (call by reference, non-primitive types)
   2.2 Comments
      2.2.1 Single-line comment
      2.2.2 Multi-line comment
   2.3 Program
   2.4 Functions
      2.4.1 Define a Function
      2.4.2 Function Return Statements
      2.4.3 main Function
      2.4.4 Built-in Functions
   2.5 Variables
      2.5.1 Define a Variable
   2.6 Loops
      2.6.1 While-loop
      2.6.2 For-loop
   2.7 If-else
   2.8 Operators
      2.8.1 Assignment Operator
      2.8.2 Arithmetic Operator
      2.8.3 Comparison Operators
      2.8.4 Logical Operators
   2.9 Keywords & Separators
   2.10 Memory
   2.11 Scope
3. Language Tutorial
   3.1 Initial Setup
   3.2 Get Started
      3.2.1 Program
      3.2.2 Function
      3.2.3 Variable
      3.2.4 Statements and Expressions
      3.2.5 Matrix
      3.2.6 Comment
   3.3 Sample Code
4. Project Plan
   4.1 Planning Process
   4.2 Specification Process
   4.3 Development Process
   4.4 Testing Process
   4.5 Team Responsibilities
   4.6 Project Timeline
   4.8 Software Development Environment
   4.9 Programming Style Guide
5. Architecture Design
   5.1 Architecture Diagram
   5.2 Scanner
   5.3 Parser
   5.4 Semantic Checker
   5.5 Code Generator
6. Test Plan
   6.1 Source Language Programs
      6.1.1 Source
      6.1.2 Target
   6.2 Test Suite
      6.2.1 Passing Tests (27)
      6.2.2 Failing Tests (19)
Marble is a programming language that incorporates matrix manipulation functionalities natively so that the compiled code can solve linear algebra problems efficiently. With Marble, developers can define matrices using Matlab–like literal syntax, i.e. \( M = \begin{bmatrix} 0,0,0;0,0,0 \end{bmatrix} \) or \( M = \text{zeros}(3,2) \). Marble includes a bare minimum number of matrix manipulation functions in the language to speed up compiling. Since language is flexible, developers can write functions for other matrix operations that are not provided.
2. Language Manual

2.1 Data Types

In order to accomplish certain operations and functions efficiently, we create the following primitive types as building blocks (each of which contains simple values of a kind).

2.1.1 Primitive types (call by value)

- **int**: Integer under a range of $-2^{30}$ to $2^{30} - 1$
- **float**: OCaml float type (IEEE 754 with a 53-bit mantissa and exponents from $-1022$ to $1023$)
- **bool**: true/false
- **null**: type of variables after declaration and before assignment, type of defined functions, type of variables assigned to functions without return statements

2.1.2 Built-in types (call by reference, non-primitive types)

- **matrix**:
  - accessor: `a[1,0]`
  - dimension: `rows(a) cols(a)`
  - initialization: `matrix A = [1.0,2.0;3.0,4.0]; matrix B = zeros(3,2);`
  - operators:
    - addition, substraction (+, -): `[1.0;2.0] + [1.0;2.0]`
    - scalar multiplication (*): `3 * [1.0;2.0]` or `[1.0;2.0] * 5.0`
    - matrix multiplication (*): `[1.0,2.0] * [1.0;2.0]`
  - Note that all entries in the matrix need to be float.

Example:

```java
1    // initialize a matrix
2    matrix A = [1.0,2.0;3.0,4.0];
3
4    // access an element in the matrix
5    A[1,0]; // returns 3.0
6
7    // getting the matrix dimension, number of rows, number of cols
8    rows(A); // returns 2
9    cols(A); // returns 2
```
2.2 Comments

2.2.1 Single-line comment
The content after the symbol `//` within a line is recognized as a comment in our language and our interpreter will skip the content during the execution.

Example:

```
1  // This is a comment
```

2.2.2 Multi-line comment
Any content after `/*` and before `*/` is recognized as a comment in our language and our interpreter will skip the content during the execution.

Example:

```
1  /*
2  This is also a comment
3  */
```

2.3 Program
When developers write code in Marble, the file that contains the code is a Marble program. A program consists of a collection of function declarations, variable declarations, and one and only one `main()`.

- Function declarations and variable declarations are optional and can be in any order before the `main()` function
- One `main()` function is required for every program

Example:
A function is a collection of input parameters and statements. A function declaration creates one function and binds the corresponding identifier to it.

- A function must have a name; adding parenthesis () to the end of its name will invoke the function.
- A function must have a return type. It can be any data type.
- Input parameters are optional and multiple input parameters are separated by commas ,. Each input parameter must have a type and a name.
- Inside the curly braces {} is a collection of 0 or more statements.
- A function can have 0 or more return statements and these return statements must have the match the declared return type.
- Within the same scope, functions must have different names.

Example:
2.4.2 Function Return Statements

- Once any `return` statement is executed, the function will terminate
- The value returned by the function will be available in the context where the function is invoked

Example:

```java
int function fib(int n){
    if(n == 1){
        return 1;
    }
    if(n == 2){
        return 2;
    }
    return fib(n-1) + fib(n-2);
}
```

2.4.3 `main` Function

`main` function is a special type of function. In particular, its name must be "main" and it lacks input parameters.

- One program must have one `main` function.
- Return statements are required in the `main` function and they will terminate the program. The value returned from the `main` function is useless
- Code will start executing from `main`.

Example:

```java
int function get(matrix m, int row, int col){
    return m[row,col];
}
int function set(matrix m, int row, int col, int val){
    m[row,col] = val;
    return 0;
}
matrix m = [1,2;1,2];
float a = get(m,0,0); // a = 1.0
bool b = set(m,1,1,2); // b = null
```
2.4.4 Built-in Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print</code></td>
<td><code>int i</code></td>
<td>Print out an integer</td>
</tr>
<tr>
<td><code>printb</code></td>
<td><code>bool b</code></td>
<td>Print out a boolean as 0 or 1</td>
</tr>
<tr>
<td><code>printf</code></td>
<td><code>float f</code></td>
<td>Print out a float</td>
</tr>
<tr>
<td><code>printmf</code></td>
<td><code>matrix m</code></td>
<td>Print out a matrix</td>
</tr>
<tr>
<td><code>rows</code></td>
<td><code>matrix m</code></td>
<td>Return number of rows of the matrix</td>
</tr>
<tr>
<td><code>cols</code></td>
<td><code>matrix m</code></td>
<td>Return number of cols of the matrix</td>
</tr>
<tr>
<td><code>zeros</code></td>
<td><code>int r, int c</code></td>
<td>Return a $r$ rows by $c$ cols matrix, filled with 0.0</td>
</tr>
</tbody>
</table>

2.5 Variables

2.5.1 Define a Variable

A variable has a type, a name, and an optional value. A variable declaration creates one variable, binds corresponding identifiers to it, and gives it a type and an initial value.

- One variable can only have one type in its lifetime. There’s no way to change its type. A runtime error will be thrown if the variable and the value it is assigned to have mismatched types. See the "2.1 Data Types" section for more details.
- A variable declared without assigning an initial value will have a null value. Variables can be reassigned later in the program.
- Variables must be declared before assigning a value to it or before using it.
- Within the same scope, variables must have different names.
- Variables in the global scope cannot
2.6 Loops

2.6.1 While–loop

The format for while–loop is `while(expr){stmts}`. The expression is the condition part of the loop. The expression is of type boolean and the type check will be done during runtime. The loop–body is a statement list.

Example:

```java
int i = 0;
while(i < 10){
    i = i + 1;
}
```

2.6.2 For–loop

The format for for–loop is `for(assignstmt;expr;assignstmt){stmts}`. The assignment statement, excluding the matrix assignment, is the init part, such as `int i = 0, i = 0, i += 1` or `i -= 1`. The expression with type boolean is the condition part and the type check will be done during runtime. The part after the condition part is also an assignment statement, which excludes the matrix assignment and will be executed after each iteration. The loop–body is a statement list.

Example:
2.7 If–else

The format is `if(expr){stmts}else{stmts}`.

The if–branch is required and can only have one. The else–branch is optional and can have zero or one else. `elif` can be filled by adding another if–else in the if–branch/else–branch.

Example:

```java
int a = 10;
if (a <= 10) {
    if (a <= 20) {
        // this is identical to "else if"
    }
}
else {
    // ...
}
```

2.8 Operators

2.8.1 Assignment Operator

The equal sign `=` is used to indicate storing values in variables with the format `type ID = expr;` or `ID = expr;`. Type checking will be done during the runtime.

We also support `+=, -=` and the syntax is `expr += expr;` or `expr -= expr;`. This shortcut is only available for int and float.

If a variable is assigned a value before the declaration, the error will be caught during the compilation.

Example:

```java
int x = 1;
x += 2;
```

2.8.2 Arithmetic Operator

```java
int n = 1;
for(int i = 0; i < 10; i = i + 1){
    n = n * 10;
}
```
The following standard arithmetic operators are provided (only applies to int/float):

- addition +
- subtraction and sign negation -
- multiplication *
- division /
- modular %

```
1  int a = 1;
2  int b = 3;
3  int c = a + b;
4  print(c); //out 4
5  c = b - a;
6  print(c); //out 2
7  c = b * c;
8  print(c); //out 6
9  c = c / b;
10 print(c); //out 2
11 c = c % b;
12 print(c); //out 2
```

2.8.3 Comparison Operators

The following comparison operators are provided:

- greater than >
- less than <
- greater than or equal to >=
- less than or equal to <=
- equal to ==
- not equal !=

All comparison operators will be performed on the values of the operands, not the reference addresses.

Example:
The following logical operators are provided:

- negate `!`
- and `&&`
- or `||`

Only boolean expressions are allowed. Any other expressions will cause runtime errors.

Example:

```java
// The above code will produce 0 1 1 0 1 1 1
#
bool isPositive = 1 < 0;
bool isNegative = !isPositive;
printb(isPositive && isNegative); // 0
printb(isPositive || isNegative); // 1
```

### 2.8.4 Logical Operators

#### 2.9 Keywords & Separators

The following keywords are reserved. If used as a variable name, the compiler will throw an error indicating that the keyword cannot be used.
<table>
<thead>
<tr>
<th>Keywords</th>
<th>Format</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| if, else    | if(expr)  
{stmts}  
else  
{stmts} | Reserved for conditional statements         |
| for, while  | for(assignstmt; expr; assignstmt ){stmts}  
while(expr){stmts} | Reserved for flow control                    |
| main        | int function main(){stmts}                 | main function is used to indicate the starting point to execute the program |
| function    | function foo(){}                          | Reserved for functions                       |
| return      | return expr;                               | Reserved for function return statements      |
| null        |                                             | Evaluates to \textit{false} when used as a boolean |
| int, float, | type ID;                                   | Built-in datatypes                           |
| bool, matrix|                                             |                                              |
|             | ( ) [ ] { } , ;                           | separators                                   |

### 2.10 Memory
We use call–by–value for both primitive and non–primitive types.

Example:
We choose to use static scoping in our language since we want to facilitate modular coding. In this scoping, a variable always refers to its top-level environment.

Example:

```java
int function swap(int a, int b){
    int c = a;
    a = b;
    b = c;
    return 0;
}

int function main(){
    int a = 1;
    int b = 2;
    swap(a,b); // a = 1, b = 2
    return 0;
}
```

### 2.11 Scope

We choose to use static scoping in our language since we want to facilitate modular coding. In this scoping, a variable always refers to its top-level environment.

Example:

```java
int function meth(){
    int a = 0;
    return 0;
}

int function main(){
    // Invalid since "a" is not declared in meth2's scope
    // Error: "undeclared identifier a"
    int b = a + 2;
    return 0;
}
```
To use Marble, please install OCaml and OCaml LLVM binding library, Clang and llvm on local machine first. Then follow the instructions below.

```
1 cd <project folder>
2 opam init
3 opam install llvm.<version number>
4 eval 'opam config env
```

Note: Do not forget to link lli and llc to OCaml.
For example:

```
1 ln -s /usr/bin/lli-6.0 /usr/bin/lli
```

To run our tests, please run the command `make test`.

3.2 Get Started

3.2.1 Program
Each program has and only has one function called main, which is the entry point of this program. Besides the main function, each program can also have optional global variables and functions.

### 3.2.2 Function

All functions, including main, are defined as follows.

```plaintext
<return type> function <function name> (<param1 type> <param1 name>, ....) {
    <a list of statements>
    <return statement>
}
```

### 3.2.3 Variable

Global variables are defined as `<var type> <var name>;`. Local variables are defined as `<var type> <var name>; or <var type> <var name> = <init value>;

### 3.2.4 Statements and Expressions

 Statements always end with a semi-colon ( ; ). Most statements and expressions allowed in mainstream languages are also allowed in Marble. As for a full list of statements and expressions, please refer to our language manual.

### 3.2.5 Matrix

Matrix is the most important part that distinguishes Marble from other languages. Follow the instructions to learn how to declare and manipulate matrices in Marble!
3.2.6 Comment

Single-line comments always start with //.

Multiple-line comments always start with /* and end with */.

3.3 Sample Code
4. Project Plan

4.1 Planning Process

We first set our main goals on a set of features we wanted to implement. Then, we categorized features into basic language features, matrix-related features, and nice-to-have features. We set up milestone deadlines based on these features, the structure of compilers, and suggestions from meetings with our project advisor, Maxwell. We created short-term goals as we work through each milestone. The milestones we set and our full project log are outlined in the following sections 8.1.

We've encountered situations when one member's task was blocked by another member's. We found that constant communication and being flexible could help mitigate the situation. We also did pair programming when implementing the first Hello World milestone and found it extremely useful.

4.2 Specification Process

The initial specification of Marble was based on the set of language features and the main building blocks required to build an end-to-end compiler. Marble is inspired by Java and Python.
The first specification was ambitious and we later cut down these features: dynamic function return type, \texttt{elseif}, and operator \texttt{ref} equal for checking variable memory address equal. We first finalized lexical and syntax specifications and implemented the lexer and the parser. Then, we drafted our Language Reference Manual with code samples. In both stages, each member was responsible for a subset of features and a holistic review was conducted with the team and our project advisor before moving forward to the next stage. During the development process, when revisions to the Language Reference Manual and/or parsers are needed, we would weigh choices together as a team.

### 4.3 Development Process

As suggested by Professor Edwards, we first implemented the lexer and parser based on our context-free grammar, then the AST, SAST, and the semantic checker, then the code generator. We referred to examples from class and past projects to implement our basic Hello World features, including global variables, global functions, the \texttt{main} function, and basic integer additions and subtractions. Once our basic compiler ran end-to-end, we developed more features in parallel, such as function type checking, build-in printing functions, function scoping, matrix operations. We also strive for providing intuitive error messages for syntax and runtime errors.

### 4.4 Testing Process

As we implement features, we wrote multiple success and error unit tests for each feature to ensure code quality. We added functions to visualize syntax errors, the Marble code, and intermediate data structures to debug unexpected test results. We’ve also built test automation to increase work efficiency.

### 4.5 Team Responsibilities

As the project progressed, the roles of each team member became more fluid than the initially assigned roles of \textit{Project Manager}, \textit{System Architect}, \textit{Language Guru}, and \textit{Tester}. In order to streamline the development, each team member is heavily involved with at least two major features of the Marble language, from coding to documentation, as shown below.

- Huaxuan Gao: Infrastructure and matrix-related operations
- Qiwen Luo: Statements and expressions
- Yixin Pan: Statements and expressions
- Xindi Xu: Variables, functions, and matrix initialization

### 4.6 Project Timeline
<table>
<thead>
<tr>
<th>Date</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct. 28</td>
<td>Context–free grammar completed</td>
</tr>
<tr>
<td>Oct. 28</td>
<td>Scanner, parser completed</td>
</tr>
<tr>
<td>Nov. 12</td>
<td>Semant, AST, SAST, Codegen completed</td>
</tr>
<tr>
<td>Nov. 14</td>
<td>Hello World milestone completed</td>
</tr>
<tr>
<td>Nov. 30</td>
<td>Matrix–related features completed</td>
</tr>
<tr>
<td>Dec. 4</td>
<td>Function–related features completed</td>
</tr>
<tr>
<td>Dec. 4</td>
<td>Statements– and expressions–related features completed</td>
</tr>
<tr>
<td>Dec 20</td>
<td>Addressed errors and warnings, sample code completed</td>
</tr>
<tr>
<td>Dec 22</td>
<td>Final clean up: remove unused code, formatting</td>
</tr>
</tbody>
</table>

**Code Frequency**

![Code Frequency Graph](image)

**Project Log**
4.8 Software Development Environment

Technologies used:
- GitHub-Hosted Git Repo: version control
- Ocaml (version 4.x): the programming language for implementing the Marble compiler
  - Ocamllyacc and Ocamllex: compiling scanner and parser frontend
- C (version 13.x): building helper methods for matrix operations in Marble
- LLVM (version 11.x): infrastructure for the Marble compiler
- Shell scripts and Makefile: automation for testing

4.9 Programming Style Guide

We generally followed the following guidelines while programming the Marble compiler:
- Formatting: use `ocamlformat` to format Ocaml code with the "conventional" Ocaml style guide
- Indentation: use only spaces to ensure consistency over different machines
- Naming: use snake_case as opposed to camelCase for variable and function names
- File Name: files names for passing tests start with `test-` and files names for failing tests start with `fail-

5. Architecture Design

5.1 Architecture Diagram

This is the overall system architecture. The major components are the scanner, parser, semantic checker, code generator. Each step will generate outputs that can be processed by the following step. The details of each component will be discussed in sections below. From the users' point of view, the system runs end-to-end, takes the input source file, and generates an LLVM IR that can be executed by the LLVM interpreter.
5.2 Scanner
The Marble source files will first go through the scanner. The scanner program of Marble uses `ocamllex` lexical analyzer. It will detect pre-defined keywords and regular expression patterns. It will also discard white spaces, newlines, and comments. After scanning through the source file, the scanner will generate a list of tokens, which can be used by the parser. The scanner is implemented in the `scanner.mll`.

5.3 Parser
The parser of Marble uses `ocamlyacc` to generate the Abstract Syntax Tree (AST) from input tokens. The terminal symbols are declared using `%token`. It also defines the entry point of the grammar with `%start`. Associativity and precedences are also defined in the parser. More importantly, the parser defines a set of rules, that will match the tokens with semantic actions. Semantic actions are arbitrary OCaml expressions, that are evaluated to produce the semantic attribute attached to the defined nonterminal. The parser consists of `parser.mly`, `ast.ml`.

5.4 Semantic Checker
The AST produced by the parser is not annotated with types, and it needs to be semantically checked. The semantic checker will detect duplicated use of variable or function names, and invalid usage of the null type variable declaration. It will also reject invalid types for a given operation. For example, in Marble, logical operators must be performed on two boolean type variables, and arithmetic operators must be performed on variables of the same type. It also makes sure that the value is assigned to the variables with the correct type. The generated Semantically-checked AST (SAST) will be passed to the code generator. The semantic checker is implemented in `semant.ml` and `sast.ml`.
5.5 Code Generator

The code generator takes the AST and generates LLVM intermediate representation (IR). It maps the Marble data types to the corresponding LLVM types and maps Marble operators to LLVM operators. For example, the Marble matrixes are represented by a float pointer pointing to a consecutive memory address, and we access a matrix element by calculating the offset based on row index and column index. In Marble, we have two scopes, the global scope, and the local scope, for each scope, the variable names are stored in a hash table. Finally, we link the LLVM IR with our c library, which includes our implementation of the matrix operations, to generate an executable.

6. Test Plan

6.1 Source Language Programs

6.1.1 Source

Program 1:
This is a simple program where we define two variables a and b with matrix type. Then we conduct matrix addition, subtraction, and multiplication among them. The +, −, * shown in the code below is the syntactical sugar with their actual functionality is calling the built-in function matrix additions, matrix subtractions, and matrix multiplications defined in our library. With the printing function, we can check if the result is consistent with the expected output.
Program 2:
This is a simple program where we would like to access a matrix element and update its value. Our language support that user can access a matrix element with index (row, col) by mat[row, col] and update a matrix element with a simple assign statement. The `printf` function would print out the whole matrix where we can check if the matrix element has been updated.

```java
int function main() {
    // mat +/- mat
    matrix a = [1.1, 2.2; 3.3, 4.4];
    matrix b = [5.5, 6.6; 7.7, 8.8];
    matrix c = a + b;
    printf(c);
    matrix d = a - b;
    printf(d);
    // int * mat mat * int
    int d = 3;
    matrix e = d * a;
    printf(e);
    matrix f = a * d;
    printf(f);
    // float * mat mat * float
    float g = 1.1;
    matrix h = g * a;
    printf(h);
    matrix i = b * g;
    printf(i);
    // mat * mat
    matrix j = a * b;
    printf(j);
    matrix k = b * a;
    printf(k);
    return 0;
}
```

6.1.2 Target
Target for program 1:
```java
; ModuleID = 'Marble'
source_filename = "Marble"

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

declare i32 @printf(i8*, ...)
declare i32 @printfm(double*)
declare i32* @addm(i32*, i32*)
declare double* @addmf(double*, double*)
declare i32* @subm(i32*, i32*)
declare double* @submf(double*, double*)
declare i32* @scalarm(double, i32*)
declare double* @scalarmf(double, double*)
declare i32* @multiplication(i32*, i32*)
declare double* @multiplicationf(double*, double*)
define i32 @main() {
  entry:
  %matrix = alloca [6 x double], align 8
  %ptr = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 0
  store double 2.000000e+00, double* %ptr, align 8
  %ptr1 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 1
  store double 2.000000e+00, double* %ptr1, align 8
  %ptr2 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 2
  store double 1.100000e+00, double* %ptr2, align 8
  %ptr3 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 3
  store double 2.200000e+00, double* %ptr3, align 8
  %ptr4 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 4
  store double 3.300000e+00, double* %ptr4, align 8
  %ptr5 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 5
  store double 4.400000e+00, double* %ptr5, align 8
```

%a = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 0
%a7 = alloca double*, align 8
store double* %a, double** %a7, align 8
%matrix8 = alloca [6 x double], align 8
%ptr9 = getelementptr inbounds [6 x double], [6 x double]* %matrix8, i32 0, i32 0
store double 2.000000e+00, double* %ptr9, align 8
%ptr10 = getelementptr inbounds [6 x double], [6 x double]* %matrix8, i32 0, i32 1
store double 2.000000e+00, double* %ptr10, align 8
%ptr11 = getelementptr inbounds [6 x double], [6 x double]* %matrix8, i32 0, i32 2
store double 5.500000e+00, double* %ptr11, align 8
%ptr12 = getelementptr inbounds [6 x double], [6 x double]* %matrix8, i32 0, i32 3
store double 6.600000e+00, double* %ptr12, align 8
%ptr13 = getelementptr inbounds [6 x double], [6 x double]* %matrix8, i32 0, i32 4
store double 7.700000e+00, double* %ptr13, align 8
%ptr14 = getelementptr inbounds [6 x double], [6 x double]* %matrix8, i32 0, i32 5
store double 8.800000e+00, double* %ptr14, align 8
%b = getelementptr inbounds [6 x double], [6 x double]* %matrix8, i32 0, i32 0
%b16 = alloca double*, align 8
store double* %b, double** %b16, align 8
%a17 = load double*, double** %a7, align 8
%b18 = load double*, double** %b16, align 8
%c = call double* @addmf(double* %a17, double* %b18)
%c19 = alloca double*, align 8
store double* %c, double** %c19, align 8
%c20 = load double*, double** %c19, align 8
%printfm = call i32 @printfm(double* %c20)
%a21 = load double*, double** %a7, align 8
%b22 = load double*, double** %b16, align 8
%d = call double* @submf(double* %a21, double* %b22)
%d23 = alloca double*, align 8
store double* %d, double** %d23, align 8
%d24 = load double*, double** %d23, align 8
%printfmf25 = call i32 @printfm(double* %d24)
%d26 = alloca i32, align 4
store i32 3, i32* %d26, align 4
%d27 = load i32, i32* %d26, align 4
%d28 = load double*, double** %a7, align 8
%scalar = sitofp i32 %d27 to double
%e = call double* @scalarmf(double %scalar, double* %a28)
%e29 = alloca double*, align 8
store double* %e, double** %e29, align 8
%e30 = load double*, double** %e29, align 8
%printmf31 = call i32 @printf(double* %e30)
%a32 = load double*, double** %a7, align 8
%d33 = load i32, i32* %d26, align 4
%scalar34 = sitofp i32 %d33 to double
%f = call double* @scalarmf(double %scalar34, double* %a32)
%f35 = alloca double*, align 8
store double* %f, double** %f35, align 8
%f36 = load double*, double** %f35, align 8
%printmf37 = call i32 @printf(double* %f36)
%g = alloca double, align 8
store double 1.100000e+00, double* %g, align 8
%g38 = load double, double** %g, align 8
%a39 = load double*, double** %a7, align 8
%h = call double* @scalarmf(double %g38, double* %a39)
%h40 = alloca double*, align 8
store double* %h, double** %h40, align 8
%h41 = load double*, double** %h40, align 8
%printmf42 = call i32 @printf(double* %h41)
%b43 = load double*, double** %b16, align 8
%g44 = load double, double** %g, align 8
%i = call double* @scalarmf(double %g44, double* %b43)
%i45 = alloca double*, align 8
store double* %i, double** %i45, align 8
%i46 = load double*, double** %i45, align 8
%printmf47 = call i32 @printf(double* %i46)
%a48 = load double*, double** %a7, align 8
%b49 = load double*, double** %b16, align 8
%j = call double* @multiplicationf(double* %a48, double* %b49)
%j50 = alloca double*, align 8
store double* %j, double** %j50, align 8
%j51 = load double*, double** %j50, align 8
%printmf52 = call i32 @printf(double* %j51)
%b53 = load double*, double** %b16, align 8
%a54 = load double*, double** %a7, align 8
%k = call double* @multiplicationf(double* %b53, double* %a54)
%k55 = alloca double*, align 8
store double* %k, double** %k55, align 8
%k56 = load double*, double** %k55, align 8
%printmf57 = call i32 @printf(double* %k56)
ret i32 0
; ModuleID = 'Marble'
source_filename = "Marble"

@fmt = private unnamed_addr constant [4 x i8]cio\0A\00, align 1
@fmt.1 = private unnamed_addr constant [4 x i8]c\0g\0A\00, align 1

declare i32 @printf(i8*, ...)
declare i32 @printfm(double*)
declare i32* @addm(i32*, i32*)
declare double* @addmf(double*, double*)
declare i32* @subm(i32*, i32*)
declare double* @submf(double*, double*)
declare i32* @scalarm(double, i32*)
declare double* @scalarmf(double, double*)
declare i32* @multiplication(i32*, i32*)
declare double* @multiplicationf(double*, double*)

define i32 @main() {
entry:
%matrix = alloca [6 x double], align 8
%ptr = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 0
store double 2.000000e+00, double* %ptr, align 8
%ptr1 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 1
store double 2.000000e+00, double* %ptr1, align 8
%ptr2 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 2
store double 1.100000e+00, double* %ptr2, align 8
%ptr3 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 3
store double 2.200000e+00, double* %ptr3, align 8
%ptr4 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 4
store double 3.300000e+00, double* %ptr4, align 8
%ptr5 = getelementptr inbounds [6 x double], [6 x double]* %matrix, i32 0, i32 5
store double 4.400000e+00, double* %ptr5, align 8
We start writing all test cases for the "Hello World" iteration where we test primitive type variable declaration, assignments, and related basic arithmetic operations. Then we expanded the test into 1) all types of variables including matrix and float, 2) all types of Binary Op including mod, comparison, and negation associated with all available data type, 3) all statements including if else, while, 4) all function–related issue including its formals, return, and scope, 5) lastly all matrix–related operations such as matrix multiplication, matrix element update, and dimension check.

6.2.1 Passing Tests (27)
```java
// test-boolOp.mb
int function main(){
    printf(true&&true);
    printf(true&&false);
    printf(true||false);
    printf(true||true);
    printf(false||true);
    printf(false||false);
    printf(!true);
    printf(!false);
    return 0;
}
// test-boolOp.out
1 0 1 1 1 0

***************************************************************************
//test-comments.mb
/*
Each program must include one `main` function.
*/
int function main(){
    // print 111
    printf(111);
    return 0;
}
//test-comments.out
111

***************************************************************************
//test-floatBinop.mb
int function main(){
    printf(1.1+2.2);
    printf(1.1*2.2);
    printf(1.1-2.2);
    printf(1.1/2.2);
    printf(4.1%3.0);
```
```cpp
    printb(4.0<2.0);
    printb(4.0>2.0);
    printb(3.0>=3.0);
    printb(5.0<=3.0);
    printb(3.0==3.0);
    printb(3.0!=4.0);
    return 0;
  }

  //test-floatBinop.out
  3.3
  2.42
  -1.1
  0.5
  1.1
  0
  1

  //test-floatUnop.mb
  int function main(){
    printf(-2.1);
    return 0;
  }

  //test-floatUnop.out
  -2.1

  //test-funcFormal.mb
  int function custom_print(int a){
    print(1);
    return 1;
  }

  int function main(){
    custom_print(1);
    return 0;
  }

  //test-funcFormal.out
  1

  //test-funcReturn.mb
  int function get1(){
    return 1;
  }
```

```c
float function getPi(){
  return 3.14;
}

bool function getTrue(){
  return true;
}

matrix function getMatrix(){
  matrix a = [1.1, 2.2; 3.3, 4.4];
  return a;
}

int function main(){
  print(get1());
  printf(getPi());
  printb(getTrue());
  printfm(getMatrix());
  return 0;
}

int function main(){
  print(1+2);
  print(1*2);
  print(2-1);
  print(4/2);
  print(4%3);
  printb(4<2);
  printb(4>2);
  printb(3>=3);
  printb(5<=3);
  printb(3==3);
  printb(3!=4);
  return 0;
}
```

//test-intUnop.mb
int function main(){
    print(-2);
    return 0;
}

//test-intUnop.out
-2

//test-matrix_access.mb
int function main() {
    matrix a = [1.1, 2.2; 3.3, 4.4];
    float b = a[1,1];
    printf(b);
    return 0;
}

//test-matrix_access.out
4.4

//test-matrix_dim.mb
int function main() {
    matrix a = [1.1, 2.2; 3.3, 4.4; 5.5, 6.6];
    int r = rows(a);
    print(r);
    int c = cols(a);
    print(c);
}

//test-matrix_dim.out
3
2

//test-matrix_ops.mb
int function main() {
    // mat +/- mat
matrix a = [1.1, 2.2; 3.3, 4.4];
matrix b = [5.5, 6.6; 7.7, 8.8];
matrix c = a + b;
printf(c);
matrix d = a - b;
printf(d);
// int * mat mat * int
int d = 3;
matrix e = d * a;
printf(e);
matrix f = a * d;
printf(f);
// float * mat mat * float
float g = 1.1;
matrix h = g * a;
printf(h);
matrix i = b * g;
printf(i);
// mat * mat
matrix j = a * b;
printf(j);
matrix k = b * a;
printf(k);
return 0;
}
//test-matrix_ops.out
6.600000 8.800000
11.000000 13.200000
-4.400000 -4.400000
-4.400000 -4.400000
3.300000 6.600000
9.900000 13.200000
3.300000 6.600000
9.900000 13.200000
1.210000 2.420000
3.630000 4.840000
6.050000 7.260000
8.470000 9.680000
22.990000 26.620000
52.030000 60.500000
27.830000 41.140000
37.510000 55.660000
*************************************************************************
//test-matrix_update.mb
int function main() {
    matrix a = [1.1, 2.2; 3.3, 4.4];
a[0,1] = 5.5;
printf(a);
return 0;
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>} //test-matrix_update.out</td>
</tr>
<tr>
<td>246</td>
<td>1.100000 5.500000</td>
</tr>
<tr>
<td>247</td>
<td>3.300000 4.400000</td>
</tr>
<tr>
<td>248</td>
<td>*************************************************************************</td>
</tr>
<tr>
<td>249</td>
<td>//test-print.mb</td>
</tr>
<tr>
<td>250</td>
<td>int function main(){</td>
</tr>
<tr>
<td>251</td>
<td>print(111);</td>
</tr>
<tr>
<td>252</td>
<td>return 0;</td>
</tr>
<tr>
<td>253</td>
<td>} //test-print.out</td>
</tr>
<tr>
<td>254</td>
<td>111</td>
</tr>
<tr>
<td>255</td>
<td>*************************************************************************</td>
</tr>
<tr>
<td>256</td>
<td>//test-printb.mb</td>
</tr>
<tr>
<td>257</td>
<td>int function main(){</td>
</tr>
<tr>
<td>258</td>
<td>bool a = false;</td>
</tr>
<tr>
<td>259</td>
<td>printb(a);</td>
</tr>
<tr>
<td>260</td>
<td>printf(true);</td>
</tr>
<tr>
<td>261</td>
<td>return 0;</td>
</tr>
<tr>
<td>262</td>
<td>} //test-printb.out</td>
</tr>
<tr>
<td>263</td>
<td>0</td>
</tr>
<tr>
<td>264</td>
<td>1</td>
</tr>
<tr>
<td>265</td>
<td>*************************************************************************</td>
</tr>
<tr>
<td>266</td>
<td>//test-printf.mb</td>
</tr>
<tr>
<td>267</td>
<td>int function main() {</td>
</tr>
<tr>
<td>268</td>
<td>float a = 1.1;</td>
</tr>
<tr>
<td>269</td>
<td>printf(a);</td>
</tr>
<tr>
<td>270</td>
<td>return 0;</td>
</tr>
<tr>
<td>271</td>
<td>} //test-printf.out</td>
</tr>
<tr>
<td>272</td>
<td>1.1</td>
</tr>
<tr>
<td>273</td>
<td>*************************************************************************</td>
</tr>
<tr>
<td>274</td>
<td>//test-printm.mb</td>
</tr>
<tr>
<td>275</td>
<td>int function main(){</td>
</tr>
<tr>
<td>276</td>
<td>matrix a = [1.0, 2.0; 3.0, 4.0];</td>
</tr>
<tr>
<td>277</td>
<td>printf(a);</td>
</tr>
<tr>
<td>278</td>
<td>return 0;</td>
</tr>
<tr>
<td>279</td>
<td>matrix b;</td>
</tr>
<tr>
<td>280</td>
<td>b = [1.0, 2.0; 3.0, 4.0; 5.0, 6.0];</td>
</tr>
<tr>
<td>281</td>
<td>printf(b);</td>
</tr>
<tr>
<td>282</td>
<td>return 0;</td>
</tr>
<tr>
<td>283</td>
<td>} //test-printm.out</td>
</tr>
<tr>
<td>284</td>
<td>1.000000 2.000000</td>
</tr>
<tr>
<td>285</td>
<td>3.000000 4.000000</td>
</tr>
<tr>
<td>286</td>
<td>1.000000 2.000000</td>
</tr>
</tbody>
</table>
int a;

int function foo(){
    int a = 1;
    print(a);
    return 0;
}

int function main(){
    int a = 2;
    foo();
    print(a);
    return 0;
}

//test-scope.out
1
2

//test-stmtAssign.mb
int x;

int function main(){
    x = 1;
    x = x + 1;
    print(x);
    x += 2;
    print(x);
    x -= 2;
    print(x);
    return 0;
}

//test-stmtAssign.out
2
4
2

//test-stmtFor.mb
int function main(){
    int i = 0;
    for(int x = 0;x < 10;x=x+1){
        i = i + 1;
    }
    int a = 3;
    int function foo(){
        int a = 4;
        print(a);
        return 0;
    }
    int function main(){
        int a = 5;
        foo();
        print(a);
        return 0;
    }

//test-scope.out
1
2

//test-stmtAssign.out
2
4
2

//test-stmtFor.out
2
4
2
```c
338 }  
339     print(i);
340     return 0;
341 }  
342 //test-stmtFor.out  
343 10  
344 /**************************************************************************/
345 //test-stmtIfElse.mb  
346 int function main() {
347     int a = 1;
348     if(a > 0){
349         a = -1;
350     }
351     else{
352         a = 1;
353     }
354     print(a);
355     if(a > 0){
356         a = -1;
357     }
358     else{
359         a = 1;
360     }
361     print(a);
362     if(a >= 1){
363         a = 2;
364     }
365     print(a);
366     return 0;
367 }
368 //test-stmtIfElse.out
369 -1
370 1
371 2
372 /**************************************************************************/
373 //test-stmtVDeAssign.mb
374 int function main() {
375     int x = 1;
376     print(x);
377     return 0;
378 }
379 //test-stmtVDeAssign.out
380 1
```
385  *****************************************************************************
386  //test-stmtVDeclare.mb
387  int function main(){
388     int x;
389     x = 2;
390     print(x);
391     return 0;
392  }
393  //test-stmtVDeclare.out
394  2
395  *****************************************************************************
396  //test-stmtWhile.mb
397  int function main(){
398      int i = 1;
399      int x = 1;
400      while(x < 10){
401          x = x + i;
402      }
403      print(x);
404      return 0;
405  }
406  //test-stmtWhile.out
407  10
408 409  *****************************************************************************
410  //test-determinant.mb
411  float function determinant(matrix a){
412      int dim = rows(a);
413      float r_scalar = 1.0;
414
415      // Row ops on matrix a to get in upper triangle form
416      for(int dia = 0; dia < dim; dia=dia+1){
417          for(int r = dia + 1; r < dim; r=r+1){
418              r_scalar = a[r,dia] / a[dia,dia];
419              for(int c = 0; c < dim; c=c+1){
420                  a[r,c] = a[r,c] - r_scalar * a[dia,c];
421              }
422          }
423      }
424
425      // Once matrix a is in upper triangle form
426      // Multiply entries on the diagonal
427      float product = 1.0;
428      for(int i = 0; i < dim; i = i+1){
429          product = product * a[i,i];
430      }
431      return product;
```c
int function main(){
    matrix a = [1.0, 2.0; 3.0, 4.0];
    float det_a = determinant(a);
    printf(det_a); // -2

    matrix b = [2.0, 0.0, 2.0; 1.0, 1.0, 2.0; 2.0, 1.0, 8.0];
    float det_b = determinant(b);
    printf(det_b); // 10

    return 0;
}

//test-determinant.out
-2
10
*************************************************************************
// test-matrix_zeros.mb
int function main() {
    matrix a = zeros(3, 2);
    printf(a);
    return 0;
}

// test-matrix_zeros.out
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
*************************************************************************
// test-fibonacci.mb
int function fib(int n){
    if(n == 1){
        return 1;
    }
    if(n == 2){
        return 2;
    }
    return fib(n-1) + fib(n-2);
}
```

6.2.2 Failing Tests (19)
```java
//fail-expr_binopType1.mb
int function main(){
    int x = 1;
    float y = 2.0;
    x = x + y;
    print(x);
    return 0;
}

//fail-expr_binopType1.err
Fatal error: exception Failure("illegal binary operator int + float in x + y")
*************************************************************************

//fail-expr_binopType2.mb
int function main(){
    int x = 1;
    float y = 2.0;
    print(x<y);
    return 0;
}

//fail-expr_binopType2.err
Fatal error: exception Failure("illegal binary operator int < float in x < y")
*************************************************************************

//fail-expr_binopType3.mb
int function main(){
    float x = 1.0;
    float y = 2.0;
    print(x&&y);
    return 0;
}

//fail-expr_binopType3.err
Fatal error: exception Failure("illegal binary operator float && float in x && y")
*************************************************************************

//fail-expr_unary1.mb
int function main(){
    int x = 1;
    x = !x;
    print(x);
    return 0;
}

//fail-expr_unary1.err
Fatal error: exception Failure("illegal unary operator !int in !x")
*************************************************************************
```
//fail-expr_unary2.mb
int function main()
{
    bool x = true;
    x = ~x;
    printf("\%d\n", x);
    return 0;
}

//fail-expr_unary2.err
Fatal error: exception Failure("illegal unary operator -Bool in ~x")
*************************************************************************

//fail-funcFormal.mb
int function custom_print(int a){
    printf("\%d\n", a);
    return 1;
}

int function main()
{
    custom_print();
    return 0;
}

//fail-funcFormal.err
Fatal error: exception Failure("expecting 1 arguments in custom_print();")
*************************************************************************

//fail-funcReturn.mb
int function main()
{
    return true;
}

//fail-funcReturn.err
Fatal error: exception Failure("return gives Bool expected int in true")
*************************************************************************

//fail-funcUndefined.mb
int function main()
{
    custom_print();
    return 0;
}

//fail-funcUndefined.err
Fatal error: exception Failure("unrecognized function custom_print")
*************************************************************************

// fail-matrix_dim.mb
int function main()
{
    matrix a = [1.0, 2.0; 3.0, 4.0, 5.0];
    printmf(a);
    return 0;
}

//fail-matrix_dim.err
Fatal error: exception **Failure**("all rows of matrices must have the same number of elemens")

```c
//fail-matrix_type.mb
int function main(){
    matrix a = [1.0,2.0;3,4];
    printf(a);
    return 0;
}
//fail-matrix_type.err
Fatal error: exception **Failure**("Types in matrix do not match.")
```

```c
//fail-stmt_assign.mb
int x;
int function main(){
    x = 1.1;
    printf(x);
    return 0;
}
//fail-stmt_assign.err
Fatal error: exception **Failure**("Illegal assignment!")
```

```c
//fail-stmt_for.mb
int i = 0;
int function main(){
    for(int x = 0;x + 10;x=x+1){
        i = i + 1;
    }
    printf(i);
    return 0;
}
//fail-stmt_for.err
Fatal error: exception **Failure**("Expect to have a Bool type here.")
```

```c
//fail-stmt_ifelse.mb
int function main() {
    int a = 1;
    if(a + 0){
        a = -1;
    }
    else{
        a = 1;
    }
    printf(a);
    return 0;
}
```

```c
87 Fatal error: exception **Failure**("all rows of matrices must have the same number of elemens")
88 *************************************************************************
89 //fail-matrix_type.mb
90 int function main(){
91     matrix a = [1.0,2.0;3,4];
92     printf(a);
93     return 0;
94 }
95 //fail-matrix_type.err
96 Fatal error: exception **Failure**("Types in matrix do not match.")
97 *************************************************************************
98 //fail-stmt_assign.mb
99 int x;
100 int function main(){
101     x = 1.1;
102     printf(x);
103     return 0;
104 }
105 //fail-stmt_assign.err
106 Fatal error: exception **Failure**("Illegal assignment!")
107 *************************************************************************
108 //fail-stmt_for.mb
109 int i = 0;
110 int function main(){
111     for(int x = 0;x + 10;x=x+1){
112         i = i + 1;
113     }
114     printf(i);
115     return 0;
116 }
117 //fail-stmt_for.err
118 Fatal error: exception **Failure**("Expect to have a Bool type here.")
119 *************************************************************************
120 //fail-stmt_ifelse.mb
121 int function main() {
122     int a = 1;
123     if(a + 0){
124         a = -1;
125     }
126     else{
127         a = 1;
128     }
129     printf(a);
130     return 0;
131 }
```
//fail-stmt_ifelse.err
Fatal error: exception Failure("Expect to have a Bool type here.")
*************************************************************************

//fail-stmt_ifelse2.mb
int function main() {
    int a = 1;
    if(a + 0){
        a = -1;
    }
    print(a);
    return 0;
}

//fail-stmt_ifelse2.err
Fatal error: exception Failure("Expect to have a Bool type here.")
*************************************************************************

//fail-stmt_v_de_assign.mb
int function main() {
    int x = 1.1;
    print(x);
    return 0;
}

//fail-stmt_v_de_assign.err
Fatal error: exception Failure("Type not correct")
*************************************************************************

//fail-stmt_while.mb
int function main() {
    int i = 1;
    int x = 1;
    while(x + 10){
        x = x + i;
    }
    print(x);
    return 0;
}

//fail-stmt_while.err
Fatal error: exception Failure("Expect to have a Bool type here.")
*************************************************************************

//fail-string.mb
int function main() {
    print("Hello World");
}

//fail-string.err
Fatal error: exception Failure("illegal character ")
*************************************************************************

//fail-scope.mb
int function meth() {
    int a = 0;
    return 0;
}
6.3 Test Automation

To increase the efficiency of running all the test suites, we used the test script `testall.sh` which provides a series of instructions for the system to run all the tests (include both passing and failing) with a single command. The output of a passing test would be compared with the expected we designed in the corresponding .out file (the name is consistent for .mb and .out file) while the output of a failing test would be compared to the expected error message we designed in the .err file as well. In both cases, if the content matches, the test is passed; otherwise, it would report failures and show differences in the log file.

6.4 Roles and Responsibilities

The division of test cases is consistent with the division of functionality where Huaxuan writes all test cases regarding the matrix–related operations (operations, dimension, and matrix update) and Infrastructure (printing function), Qiwen writes all test cases regarding Statements (if else, while), Yixin writes all test cases regarding Expression (Binop, unary), and Xindi writes all test cases regarding Functions (formals, return, scoping). Huaxuan set up the testing script such as `testall.sh` where all group members can use single command for testing all test cases.

7. Lessons Learned

7.1 Huaxuan Gao

The most important thing that I would like to mention here is the cooperation between team members. The Marble team is very enthusiastic about what we are doing. The collaboration is great among us. For example, when we started to think about what kind of features we should have, Yixin and Qiwen brings in their Java experience while Xindi and I were more familiar with Python, we have some interesting debates on the pros and cons of different languages. These thinkings leads to the idea of Marble. Our team is very hard working as well, I remember a time

```c
int function main(){
    // Invalid since "a" is not declared in meth2's scope
    int b = a + 2;
    return 0;
}
```

```c
int function main() {
    matrix a = zeros(3, 0);
}
```
when Qiwen submits a pull request at 4 am to fix a bug, and we held 2 meetings every week regularly.

My advice for future teams would be, keep trying until you see the turning point. Because the most exciting moment for me during the whole semester is when I put everything together and print out an integer 0, which means our system finally runs end to end. After that, features are implemented at the incredible speed.

7.2 Qiwen Luo

My biggest takeaway was that it is extremely necessary to hold a regular meeting every week or every other week when working as a team. During our regular meetings, we brainstormed on Marble's features, shared the current progress of each member, solved problems and determined tasks to be completed before the next meeting. Among them, specifying what to be completed before next meeting is the most important. It not only makes each meeting more efficient but also breaks a big project into some small tasks, which makes us feel that the task is simpler and makes the progress monitoring to be easier.

As for advice for future teams, try to determine the context–free grammar for the language as soon as possible. Instead of only discussing the language broadly, the grammar can give us a clearer picture of the language and it makes the feasibility check easier.

7.3 Yixin Pan

Besides the technical aspect I learned over the project such as Ocaml programming and Context–Free Grammar design, what I learned most is about the significance of project management and the team collaboration. To begin with, when doing the topic brainstorm, throwing out as many ideas as possible; those ideas would facilitate the discussion where you can quickly discover the core features you would like to include in the language. Then, during the language design, it would be easier for you to choose the specific features (e.g. static scoping v.s. dynamic scoping) based on the core features. In such circumstances, during the implementation, it would be more efficient to get main feature working while setting aside some nice–to–have features into later iterations which are harder to complete than the expectation. More importantly, setting up a regular implementation cycle is the key to the success. Everyone in the team should stick to the plan they proposed so that the actual implementation is always ahead of schedule rather than behind it.

7.4 Xindi Xu
Identifying a set of features to build is definitely the hardest part but the most important part of any project. Our team often ran into situations where we need to give up fancier features due to time constraints. In such situations, discussing alternative designs with the team and the project advisor early in the process allows us to stay on track. Code reviewing and pair programming also allowed everyone to get on the same page on the language design and implementation of different pieces of the codebase. Having a consistent meeting schedule with the team and the project adviser helped keep us on schedule. As for the more technical part, I benefit a lot from learning OCaml and compiler design by reading the codebases of MicroC and examples from past projects.

My advice for future teams: start designing the compiler with a bare minimum set of features and gradually expand later. Be conservative when adding new features and try to have everyone’s agreement before proceeding.

8. Appendix

8.1 Project Log
commit f42d482c37f6a4768f8b779fa3bcf2a98ba8a2cd
Merge: 89469db 5f96e1a
Author: HuaxuanGAO <33063662+HuaxuanGAO@users.noreply.github.com>
Date: Wed Dec 22 21:11:52 2021 -0500

Merge pull request #21 from PLT-Marble/fix/remove-unused-code

remove unused code

commit 5f96e1aafbacfdec9732faae29548c009a7206e31
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 21:07:03 2021 -0500

remove unused code

commit 7b06f28eaf45e0f365da2fe5aee1673d4db06cf1
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 20:43:38 2021 -0500

remove unused code

commit 89469db1ba77d762ef617edd62ace6d34057fa7
Merge: bb96f8b 724cbdd
Author: TL-QL <40536493+TL-QL@users.noreply.github.com>
Date: Wed Dec 22 20:40:49 2021 -0500

Merge pull request #20 from PLT-Marble/feat/malloc-for-matrix

fix

commit 724cbdd052137448fbc29c96c164e468909309e6
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 20:38:33 2021 -0500

fix

commit bb96f8b4b0bde2715635d2e3a4c57b817051781f
Merge: c933782 8868c57
Author: HuaxuanGAO <33063662+HuaxuanGAO@users.noreply.github.com>
Date: Wed Dec 22 19:18:21 2021 -0500

Merge pull request #19 from PLT-Marble/fix/syntax-fix

fix

commit 8868c57fe43920aaa24f354f03983d7cfdf9d14e
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 19:11:44 2021 -0500
fix

commits c93378204d85af35aac47f466a024ae880c2c658
Merge: 3f2594f bff40f7
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 12:19:53 2021 -0500

Merge pull request #18 from PLT-Marble/feat/matrix-generator-zeros

feat: matrix generator: zeros()

commits bff40f7bb09da45811db1775751184c6a6f880d9
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 12:07:11 2021 -0500

rows and cols can't be 0

commits 3be72c223dd76766d5a16887f0735776fdbcfe8e4
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 11:24:18 2021 -0500

remove more unused code

commits b66158cccbfc7e0123af135b7ea77be73b0143060
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 11:21:53 2021 -0500

remove useless code

commits 4bc00a815b1a63391e80cf449366886fcf24e1f5
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 11:17:36 2021 -0500

add zeros()

commits 3f2594f6ef27436fa884b9d1efb3714b3c52c5
Merge: a020e32 8cc4e28
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 22 09:41:48 2021 -0500

Merge pull request #17 from PLT-Marble/wrapup

fixed for loop with multi stmts in loop-body

commits 8cc4e2837a8814a3b5a4908618306abb4ef3413e
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Wed Dec 22 04:06:01 2021 -0500

fixed for loop with multi stmts in loop-body
commit a020e328323a67a4c4cf1fecd71da5cda4fb6578
  Merge: d5c3770 a1c86c2
  Author: Xindi Xu <xindixu0@gmail.com>
  Date:   Tue Dec 21 22:39:02 2021 -0500
  
  Merge pull request #15 from PLT-Marble/recursive-fibonacci
  fibonacci test passed

commit d5c37706047bdd132fa4e61747fa52df7143e55a
  Merge: b9c9c18 47e86fd
  Author: Xindi Xu <xindixu0@gmail.com>
  Date:   Tue Dec 21 22:28:06 2021 -0500
  
  Merge pull request #16 from PLT-Marble/feat/demo-code
  feat/add demo code-determinant

commit 47e86fd147e7bc0a1788279161b1fe691872e1f3
  Author: Xindi Xu <xindixu0@gmail.com>
  Date:   Tue Dec 21 22:15:17 2021 -0500
  
  update variable name

commit 609a0e885827cd78daa3c6aa53ec3123b564f394
  Author: Xindi Xu <xindixu0@gmail.com>
  Date:   Tue Dec 21 22:09:55 2021 -0500
  
  add demo code

commit a1c86c2e68032e771af6f26dccc3c37a9b35dad
  Author: Huaxuan Gao <gogoghx@163.com>
  Date:   Mon Dec 20 15:13:47 2021 -0500
  
  fibonacci test passed

commit b9c9c180f98d97948f797a7bd207a89f11ffa701
  Merge: 7006f8d c51058f
  Author: TL-QL <40536493+TL-QL@users.noreply.github.com>
  Date:   Mon Dec 20 00:59:15 2021 -0500
  
  Merge pull request #14 from PLT-Marble/wrapup
  Resolved all warnings and parser conflicts

commit c51058f94edc17ec73687a4f97e9fa88b0d98eeb
  Author: TooLazy-QL <qiwenluo98@gmail.com>
  Date:   Mon Dec 20 00:47:18 2021 -0500

50
Resolved all warnings and parser conflicts

commit 7006f8d768ecb65b6c6d1ffb4a0d4cb0cfe1062
Merge: 2893ee0 cc703cc
Author: HuaxuanGAO <33063662+HuaxuanGAO@users.noreply.github.com>
Date: Sat Dec 18 15:35:14 2021 -0500

Merge pull request #13 from PLT-Marble/command-line

Command line

commit cc703ccf4cfa50c3ecc461fda5076c3ee56260a4
Merge: eb9908f 2893ee0
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sat Dec 18 15:29:00 2021 -0500

Merge branch 'main' into command-line

commit 2893ee0e8888d0ac04ba8ed942cd20591406697a
Merge: b34a55c 6b17a89
Author: HuaxuanGAO <33063662+HuaxuanGAO@users.noreply.github.com>
Date: Sat Dec 18 15:28:34 2021 -0500

Merge pull request #12 from PLT-Marble/clean-code

remove redundant code and test case

commit eb9908f141af4914e102bcb8945bf67cc1e823b3
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sat Dec 18 15:27:31 2021 -0500

command line tool marble interpreter

commit 6b17a891bfaa5c71d5f50d8f848c1b635dec05
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sat Dec 18 10:27:52 2021 -0500

remove redundant code and test case

commit b34a55cc7a574e943e05fff17bfa116d361c084e
Merge: d5da2d9 3ca6390
Author: Xindi Xu <xindixu0@gmail.com>
Date: Thu Dec 16 20:48:23 2021 -0500

Merge pull request #11 from PLT-Marble/test/more-tests

Tests: function failure tests, comment

commit 3ca63900b199416c600eca7b9e7584a8af1cb94
Author: Xindi Xu <xindixu0@gmail.com>
Date: Thu Dec 16 20:36:39 2021 -0500
failed tests

commit a7bb021d036694ad64b3e9695737bfa4288e9e31
Author: Xindi Xu <xindixu0@gmail.com>
Date: Thu Dec 16 20:29:32 2021 -0500
add more tests

commit d5da2d9607c63c6c0a58900e3265097dca7b188b
Merge: acbca4c 54dd6b3
Author: Xindi Xu <xindixu0@gmail.com>
Date: Sat Dec 4 15:26:46 2021 -0500
Merge pull request #10 from PLT-Marble/stmtNexpr
stmt and expr done

commit 54dd6b380f747450b3b0c841846c5ff83f80dded
Author: Xindi Xu <xindixu0@gmail.com>
Date: Sat Dec 4 15:26:10 2021 -0500
update tests

commit 59896807874fffd7b1db3aca971b09d7c8313185f
Merge: 2885489 acbca4c
Author: Xindi Xu <xindixu0@gmail.com>
Date: Sat Dec 4 15:19:39 2021 -0500
Merge branch 'main' into stmtNexpr

commit acbca4ca0c850de1a2bde637ef06a94d813c41d4
Merge: 7631fb2 cde43e9
Author: HuaxuanGAO <33063662+HuaxuanGAO@users.noreply.github.com>
Date: Sat Dec 4 15:07:02 2021 -0500
Merge pull request #5 from PLT-Marble/feat/explicit-function-return-type

Feat: function return type, formal, and main

commit cde43e9e958412f6ba5bafbb8260b9eab6b46dcfa
Merge: d2f6b21 67f53dc
Author: Xindi Xu <xindixu0@gmail.com>
Date: Thu Dec 2 20:45:35 2021 -0500
Merge pull request #9 from PLT-Marble/feat/treat-main-function-as-normal-functions
feat: treat main function as normal functions

commit 67f53dcc7e353d88c505ead3960f6c24380ffc5c
Merge: 258aed8 0ea75c1
Author: Huaxuan Gao <gogoghx@163.com>
Date:    Thu Dec 2 20:38:05 2021 -0500

  Merge branch 'feat/treat-main-function-as-normal-functions' of ssh://github.com/PLT-Marble/Marble into feat/treat-main-function-as-normal-functions

commit 258aed81830ddc95d12ad829ea261edefcd44a49
Author: Huaxuan Gao <gogoghx@163.com>
Date:    Thu Dec 2 20:37:52 2021 -0500

  comment type cast in test case

commit 0ea75c1d44dd7a5437af79d7bdef0058329469c
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Thu Dec 2 20:35:34 2021 -0500

  fix: reset marble.ml

commit 2aea72085b539b1e912ae1475d86b48853f1c067
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Thu Dec 2 20:35:34 2021 -0500

  fix: typo

commit 15ad1608c67158eff919d2479fe7a62057e65900
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Thu Dec 2 20:24:52 2021 -0500

  fix: typo

commit 48ec5fcf0c2bbcc421dbb20261b701dfdcca20f88
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Thu Dec 2 20:22:33 2021 -0500

  feat: treat main function as normal functions

commit d2f6b21edc26b9ab8cc20be7f4ea487ffa45ef94
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Thu Dec 2 01:27:43 2021 -0500

  test: return values

commit 8f31518cf4983f1d95cc805869b85f02564341ce
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Thu Dec 2 00:13:44 2021 -0500
clean up: function_decls

commit 336f2202823acbb468fddce2d7edbb4ba9d1eb8f
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 1 23:48:24 2021 -0500

  fix: add return types for predefined functions

commit 28ccce4e8bab8eaa906153b28f9491a466e79869f
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 1 18:21:11 2021 -0500

  revert .gitignore

commit 280548998990164a732e2cf15d652dd0bc35b6
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Wed Dec 1 00:04:49 2021 -0500

  stmt and expr done

commit f6040da7e76c703d9a77d24dd4573cf57bfccce9
Author: Xindi Xu <xindixu0@gmail.com>
Date: Wed Dec 1 00:04:25 2021 -0500

  fix after merge

commit 843205c5f4e88251de300ed0ad5c75bdff47ee00
Author: Xindi Xu <xindixu0@gmail.com>
Date: Tue Nov 30 23:56:00 2021 -0500

  update after merging

commit d4a8d1ad1ca521fe4b73957238990b286d75b71f
Author: Xindi Xu <xindixu0@gmail.com>
Date: Tue Nov 30 19:56:57 2021 -0500

  update gitignore

commit 9188212918e74c0c91610178952eea894500bb5d
Merge: 89e5786f761fb2
Author: Xindi Xu <xindixu0@gmail.com>
Date: Tue Nov 30 19:56:22 2021 -0500

  Merge branch 'main' into feat/explicit-function-return-type

commit 89e5786c828b6d79faa8c7f76c08362eff119096
Merge: 3775515 c47bce0
Author: Xindi Xu <xindixu0@gmail.com>
Date: Tue Nov 30 19:50:37 2021 -0500
Merge pull request #7 from PLT-Marble/feat/func-formal-final
Feat/func formal final

commit c47bce0d0729ed77eb707b1b9ad230f442c54fbe
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Tue Nov 30 19:24:26 2021 -0500

feat: handle formal for functions

commit 377551500885091abcbb1673017c2fa87b7ed
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Tue Nov 30 18:21:25 2021 -0500

update test script

commit 7631fb24ff5580a80d03458ed948ea685f01b18c
Merge: f46342c 15349eb
Author: HuaxuanGAO <33063662+HuaxuanGAO@users.noreply.github.com>
Date:    Tue Nov 30 18:10:52 2021 -0500

Merge pull request #4 from PLT-Marble/matrix
Matrix

commit f46342cbda36b92c6fa13c159f473229619eb90e
Merge: a7cf14a 69e312e
Author: HuaxuanGAO <33063662+HuaxuanGAO@users.noreply.github.com>
Date:    Tue Nov 30 18:10:44 2021 -0500

Merge pull request #3 from PLT-Marble/float-and-bool-types
Float and bool types

commit c91154df6d9a5dff28b3060f3ecd585c29d5cde
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Tue Nov 30 01:29:48 2021 -0500

function return and formals

commit 0231b410acff532a53894a905faddf3bb8e14d5cb
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Tue Nov 30 01:28:01 2021 -0500

add and check functions

commit eefba159b94c4819e1a67ecfe443cf3bcd9baf17
Author: Xindi Xu <xindixu0@gmail.com>
Date:    Tue Nov 30 01:27:01 2021 -0500
format

commit fa91c7bb5e46d81a9478529d6021953734344242
Author: Xindi Xu <xindixu0@gmail.com>
Date: Mon Nov 29 23:12:24 2021 -0500

    function with explicit return type

commit 15349ebdf3d200990a788839d6e9ccee3c06ba7e6
Author: Huaxuan Gao <gogoghx@163.com>
Date: Mon Nov 29 16:00:20 2021 -0500

    rows cols for mat

commit 13209a22d45cb4ec45c678487f292a1d07930bc7
Author: Huaxuan Gao <gogoghx@163.com>
Date: Mon Nov 29 15:44:20 2021 -0500

    mat bin op done

commit 05e64b69a3926ef94778557b436e333948604f50
Author: Huaxuan Gao <gogoghx@163.com>
Date: Mon Nov 29 15:07:45 2021 -0500

    mat getter setter

commit 8f382633467d64f7e38f22b7e43c3cd7a6e8ed06
Author: Huaxuan Gao <gogoghx@163.com>
Date: Mon Nov 29 12:57:58 2021 -0500

    print matrix

commit 73e5cd5002f5f7e0ffcede8e725ba129ce0035c83
Merge: 5bf5714 69e312e
Author: Huaxuan Gao <gogoghx@163.com>
Date: Mon Nov 29 11:33:44 2021 -0500

    Merge branch 'float-and-bool-types' into matrix

commit 5bf57142305b86c061ee5231407a91c8c25c2067
Author: Huaxuan Gao <gogoghx@163.com>
Date: Mon Nov 29 11:23:37 2021 -0500

    test forbin push to main

commit 69e312ee0325f01d114d8f9aebbd76f048cebfffb
Merge: 4c14bf5 a7cf14a
Author: AlbertG <gogoghx@163.com>
Date: Mon Nov 29 11:14:28 2021 -0500
Merge branch 'main' into float-and-bool-types

commit 4c14bf5400854c2bfca138f4a5b7a48d48a4691d
Author: Huaxuan Gao <gogohx@163.com>
Date: Mon Nov 29 11:01:41 2021 -0500

printb

commit 384990fc400bd60172b6f8866c51e06c764ff835
Author: Huaxuan Gao <gogohx@163.com>
Date: Mon Nov 29 10:03:14 2021 -0500

float op done

commit f1da3ac675c36aa9e3ac25fff1711c7eb20c8347
Author: Huaxuan Gao <gogohx@163.com>
Date: Sun Nov 28 17:42:08 2021 -0500

add float type

commit 2d16f98cd755c85d4e88bb120e251353319fc09a
Author: Xindi Xu <xindixu0@gmail.com>
Date: Fri Nov 26 22:41:41 2021 -0500

format

commit a7cf14ae3bb3da15704893cb6d3354d7f169ddc8
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Fri Nov 26 00:29:07 2021 -0500

Added while, for, if-else(no elif) - semant&codegen

commit 1b06bf8f45ac386879d348a1de6433cdf4ad401c
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Fri Nov 26 00:28:37 2021 -0500

Added while, for, if-else(no elif)

commit 42f1ea6fe3354f7c17e1e7fe739b93931d221cc6
Author: Huaxuan Gao <gogohx@163.com>
Date: Mon Nov 22 10:56:07 2021 -0500

add Make file

commit 86101c96dc1535c465e184ad2fda61d1cfb2ad97
Merge: a9571fb 8154ccc
Author: Xindi Xu <xindixu0@gmail.com>
Date: Sat Nov 20 16:01:10 2021 -0500
Merge pull request #2 from PLT-Marble/helloworld

Helloworld

commit 8154cccc8b9e7cfb455fd894278c913c68cb805
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sun Nov 14 15:10:47 2021 -0500

update readme with instructions to run

commit 4774d996d7e14150f00b2bf403c6769b4a9df4ab
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sun Nov 14 10:36:29 2021 -0500

update makefile

commit 9d5f0ff663f9f782743738b9f3b3c5f75feff4ab
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Sun Nov 14 02:57:22 2021 -0500

Completed helloworld - stmt&expr in sement&codegen with tests

commit f231d732568c93f48874fb542b8b6a8ee65f79b8
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sat Nov 13 17:42:09 2021 -0500

added test script

commit f51f427612c92e96b05e88c1c53b11c05bb24119
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sat Nov 13 17:12:48 2021 -0500

print(1) success

commit a9571fb25284754c4b0ca6f3720db03c6b66847
Author: Xindi Xu <xindixu0@gmail.com>
Date: Sat Nov 13 00:41:27 2021 -0500

handle vars and funcs in codegen

commit 489d2ea86769280c7864d46f11a3fe77abf6f321
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Fri Nov 12 23:16:01 2021 -0500

Added _tags

commit 04be1cdeb8aaba4849cbe4d95d5d0b8ada01007fe
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Fri Nov 12 23:14:53 2021 -0500
Comment out functionalities not included in this iter

commit 5914695363640a7b4392f8c4687cf2c23f1e9141
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Fri Nov 12 21:25:04 2021 -0500

Updated semant=stmt

commit 4de66c32018c94b0dc5113c193e9fe3983c4fa50
Author: Xindi Xu <xindixu0@gmail.com>
Date: Fri Nov 12 17:29:23 2021 -0500

update after merging

commit 719eb020d36a5510c21bd823de5a6e093d264daf
Author: Huaxuan Gao <gogoghx@163.com>
Date: Fri Nov 12 17:23:12 2021 -0500

marble.ml

commit 89a04bfa67790e3eb9c195a7837c2c8dc05794c0
Merge: ca01390 c03d547
Author: Xindi Xu <xindixu0@gmail.com>
Date: Fri Nov 12 17:19:02 2021 -0500

Merge pull request #1 from PLT-Marble/pretty-printing

Pretty printing

commit c03d547e0080425e6394d1abba12e92dde6fafed
Merge: 5b39dac ca01390
Author: Xindi Xu <xindixu0@gmail.com>
Date: Fri Nov 12 17:17:04 2021 -0500

Merge branch 'main' into pretty-printing

commit 5b39dac0cf77ba62d8d41c77a0aed9dca370dff3
Merge: e7246b4 b522a17
Author: Huaxuan Gao <gogoghx@163.com>
Date: Fri Nov 12 16:55:14 2021 -0500

merge

commit ca0139015642347807a416a2b3b93ba6d7687872
Author: Yixin Pan <ypan37@jhu.edu>
Date: Fri Nov 12 16:52:42 2021 -0500

expr in codegen

commit e7246b42afce23555e3bb2a7aed74fe6707742f5
Created semant.ml and codegen.ml
commit cbdbdfec2396ec79109ce9c9db6063b50c049da0
Merge: f9eeaf9 f3bb6fe
Author: Xindi Xu <xindixu0@gmail.com>
Date: Tue Nov 9 14:33:39 2021 -0500
Merge branch 'main' of https://github.com/PLT-Marble/Marble
commit f9eeaf99db5b8a62873472a1548e0415d251a3f3
Author: Xindi Xu <xindixu0@gmail.com>
Date: Tue Nov 9 14:33:37 2021 -0500
meeting notes with TA
commit f3bb6fe3f01af1be1b75dfe7ad7a5163f1a81459
Author: Yixin Pan <ypan37@jhu.edu>
Date: Mon Nov 8 23:56:21 2021 -0500
update on sast for expr
commit fd4b357ae3150ef22257d7f1609500c95aef666f
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Mon Nov 8 21:31:15 2021 -0500
Updated stmt&elifstmts&assignstmt for ast and sast
commit 9405274dd48e121bb6a3f4c17eb745922d9b1823
Author: Yixin Pan <ypan37@jhu.edu>
Date: Mon Nov 8 20:52:25 2021 -0500
quick fix for MLit
commit b99d0f916489d60ba3e0a89e0f9d066b9267525d
Author: Yixin Pan <ypan37@jhu.edu>
Date: Mon Nov 8 16:52:19 2021 -0500
update on expr for ast and sast
commit 6c5eaf9c5b1f0bb55e4828b0f165b85c0552282
Author: Yixin Pan <ypan37@jhu.edu>
Date: Mon Nov 8 16:43:53 2021 -0500
quick update on parser
commit 5d6241fc0c74600d5ecb493c6217fe446ad0f269
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sat Nov 6 20:29:57 2021 -0400
print matrix

commit 16bbfbc76b4f6fcf90fc4274344d1d627376e1a
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sat Nov 6 17:56:31 2021 -0400

    add prefix to print

commit 6d200eff2d72d9e45c56c547ec88ddf973b50744
Author: Huaxuan Gao <gogoghx@163.com>
Date: Sat Nov 6 17:11:55 2021 -0400

    printable

commit 0dd98dfd30dbee16f0a017ec54835307c2c3f3
Author: Xindi Xu <xindixu0@gmail.com>
Date: Sat Nov 6 16:01:35 2021 -0400

    update ast and add sast

commit 0ef760cedc54fc92b3f341f74b146a5d0d46e47
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Fri Nov 5 15:22:08 2021 -0400

    Changed the order of stmt append to match with stmts in fdecl

commit 6073268b7094a605f486a152075e891d2e
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Fri Nov 5 11:45:49 2021 -0400

    clean file

commit 83dc1e2921beb64a58a487934d775e181c606be7
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Fri Nov 5 11:43:56 2021 -0400

    Fixed NOT, AND, OR precedence and associativity

commit 1a5f059c5e85e8d06d016f7049a5e3c
Author: Xindi Xu <xindixu0@gmail.com>
Date: Fri Nov 5 11:06:23 2021 -0400

    TA comments Nov. 5

commit a47cdd277d997e6ce2faeba9bde04f7d9291ec0d
Author: Yixin Pan <ypan37@jhu.edu>
Date: Sat Oct 30 20:56:18 2021 -0400

    clean parser file output
commit be855c1a7cc0f93bf55afc97ba9e9f6d4d5021a7
Author: Yixin Pan <ypan37@jhu.edu>
Date:  Sat Oct 30 20:54:50 2021 -0400

  fix logic operator

commit 64b259d129ee1934f6e711cac22a433e86dd573c
Merge: 36fa031 b445df9
Author: Huaxuan Gao <huaxuan@A-G.localdomain>
Date:  Sat Oct 30 20:40:51 2021 -0400

  Merge branch 'main' of ssh://github.com/PLT-Marble/Marble into main

commit 36fa031859c3437b478dc1baf26316408e439a73
Author: Huaxuan Gao <huaxuan@A-G.localdomain>
Date:  Sat Oct 30 20:39:06 2021 -0400

  matrix op keywords

commit b445df9b3417c4dc8e50bc78c2f3fc9af51136c9
Author: Xindi Xu <xindixu0@gmail.com>
Date:  Thu Oct 28 22:59:43 2021 -0400

  fix: mod

commit 4c7cf223e284b402318794eb477723e1a8f56d24
Author: Xindi Xu <xindixu0@gmail.com>
Date:  Thu Oct 28 22:30:44 2021 -0400

  fix: update program, decls rules

commit 83f996ac0d9217beb7f6ab663a7244be4ed73163
Merge: ab6236d 55fca9c
Author: Xindi Xu <xindixu0@gmail.com>
Date:  Thu Oct 28 22:03:20 2021 -0400

  Merge branch 'main' of https://github.com/PLT-Marble/Marble into main

commit ab6236d9842691fabe27bfe8fa484b4b97cb43fe
Author: Xindi Xu <xindixu0@gmail.com>
Date:  Thu Oct 28 21:16:22 2021 -0400

  fixme: add TA review comments

commit 55fca9c80a6d5cc4a14f0359402106af4ab1834
Merge: 34668a7 4bc6f4e
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:  Thu Oct 28 17:01:21 2021 -0400

  Merge branch 'main' of https://github.com/PLT-Marble/Marble into main
796  commit 34668a7e39bbcfca594a6a190748233efb6845c5d
797  Author: TooLazy-QL <qiwenluo98@gmail.com>
798  Date:     Thu Oct 28 17:00:52 2021 -0400
799
800     Added assignment
801
802  commit 4bc6f4e22ee20331663b03ea4b1b1c706c2d033c
803  Author: Huaxuan Gao <huaxuan@A-G.localdomain>
804  Date:     Thu Oct 28 11:45:47 2021 -0400
805
806     add TRUE/FALSE to keywords, remove BOOL
807
808  commit a910c799c754771cf42c2e9a4f0eb66924ff6216
809  Author: Huaxuan Gao <huaxuan@A-G.localdomain>
810  Date:     Thu Oct 28 11:27:36 2021 -0400
811
812     fix shift/reduce conflict
813
814  commit 7f1d49c4b4ee9e67368029b98eb2ebd539c25769
815  Author: Huaxuan Gao <huaxuan@A-G.localdomain>
816  Date:     Thu Oct 28 11:17:07 2021 -0400
817
818     fix symbol not found and some typo
819
820  commit 4b4de680eb38469d6772314d61c72523e156431f
821  Author: Yixin Pan <ypan37@jhu.edu>
822  Date:     Wed Oct 27 23:31:52 2021 -0400
823
824     quick fix for continue and break
825
826  commit b871f4a803e052d98e01f14ca8402df56d059cbf
827  Author: Yixin Pan <ypan37@jhu.edu>
828  Date:     Wed Oct 27 23:30:13 2021 -0400
829
830     parser for expr
831
832  commit a87667e512b0ecbdeeeae449b411843ab6db0b525
833  Merge: b4a0733 fef682c
834  Author: TooLazy-QL <qiwenluo98@gmail.com>
835  Date:     Wed Oct 27 19:47:05 2021 -0400
836
837     Merge branch 'main' of https://github.com/PLT-Marble/Marble into main
838
839  commit b4a0733f01c702318b638d3491d31eae09285e4
840  Author: TooLazy-QL <qiwenluo98@gmail.com>
841  Date:     Wed Oct 27 19:46:41 2021 -0400
842
843     Added stmts, stmt and elifstmts
commit fef682ce9bbbedce910f14e07e579113fca4e70dc
Author: Xindi Xu <xindixu0@gmail.com>
Date:   Wed Oct 27 19:31:58 2021 -0400

    fix: use epsilon instead of opt for formals

commit 7a73b1d76cc36f7a4b4f7f9be76cb5efbee76774
Author: Xindi Xu <xindixu0@gmail.com>
Date:   Wed Oct 27 19:28:10 2021 -0400

    feat: add CFG (program to type)

commit c5bbdcd24753494e1db2ab8c6228ee198999b16a
Author: Xindi Xu <xindixu0@gmail.com>
Date:   Wed Oct 27 19:27:41 2021 -0400

    fix: add formatter

commit 578df2460c80da6553f69ba72c7d62bfec9e4562
Merge: a395a7a d14b4c4
Author: Huaxuan Gao <huaxuan@A-G.localdomain>
Date:   Tue Oct 26 19:21:43 2021 -0400

Merge branch 'main' of ssh://github.com/PLT-Marble/Marble into main

commit a395a7aee2eeb23bb359e19ab84f80a807702ccc
Author: Huaxuan Gao <huaxuan@A-G.localdomain>
Date:   Tue Oct 26 19:21:41 2021 -0400

MLIT

commit d14b4c4a3599dfff20f0799d7a262a28e41cd37e1
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Tue Oct 26 14:58:21 2021 -0400

    Moved . to operator

commit 034149badc2841b72e5a8b5f0231b12696d8d71e
Merge: 9aa36e5 7b06435
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Tue Oct 26 14:57:10 2021 -0400

revert

commit 9aa36e5b2e55e309f7921a6492b6e2d3a3c316a
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Tue Oct 26 14:55:08 2021 -0400

    Moved . to operator
commit 7b06435b702e9b3a12e56b22337ebc09e32bde9e
Author: Huaxuan Gao <huaxuan@A-G.localdomain>
Date:   Tue Oct 26 13:19:57 2021 -0400

    parser initial setup

commit 60007419fe02004b049f64e5e8f0f096c7a546d7
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Tue Oct 26 12:16:41 2021 -0400

    Added constructor to keywords

commit ccdbeabb56299c6f8486c6f028710b3ec512c635
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Mon Oct 25 23:38:39 2021 -0400

    Added matrix literal to tokens

commit fbe2b19688650ffa34bf9be771cb22c57c4946b2
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Mon Oct 25 21:05:17 2021 -0400

    Deleted matrix from keywords and added class to keywords

commit dc78ff00f8674d95fcf32f91cc6dcc70876d153c
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Mon Oct 25 20:22:06 2021 -0400

    Add . as a seperator

commit 7842ad8af20d7ac3cb5206cfecc592c45917ad8
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Mon Oct 25 19:42:24 2021 -0400

    Added true/false to tokens

commit 6f690c091c2394bebc4bbab6cea09843ad512b0e
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Mon Oct 25 19:33:28 2021 -0400

    Added keywords for types

commit a97b17d02226a10368f1c8c53a0234c7e7b11e90
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date:   Mon Oct 25 19:09:12 2021 -0400

    Added += and -= as tokens

commit d15b2eeaf0f8fe499e213a2c711f737e8e2e49a44
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Mon Oct 25 18:59:22 2021 -0400

Changed float format to digits.digits

commit f325a3973b28afec4a4c514a75c75d0552b2df7
Author: TooLazy-QL <qiwenluo98@gmail.com>
Date: Mon Oct 25 18:53:38 2021 -0400

Deleted keyword try, catch, throw and export

commit a18cbe0785bb192e16da50127ab737754e7a7572
Author: Huaxuan Gao <huaxuan@A-G.localdomain>
Date: Sat Oct 23 14:37:15 2021 -0400

test scanner

commit cdfd2d658fa58f173826ef45e74ab5675e202e77
Author: Huaxuan Gao <huaxuan@A-G.localdomain>
Date: Sat Oct 23 10:23:23 2021 -0400

init scanner

commit 1caaf34ff5c14c8be91f20634d8b35c0748f042
Author: HuaxuanGAO <33063662+HuaxuanGAO@users.noreply.github.com>
Date: Sat Oct 23 07:47:26 2021 -0400

Initial commit
8.2 Code Listing

8.2.1 scanner.mll
{ open Parser
(* Reference: https://ocaml.org/manual/lexyacc.html *)
(* 2.7 keywords *)
let keyword_table = Hashtbl.create 53
let _ =
  List.iter (fun (kwd, tok) -> Hashtbl.add keyword_table kwd tok)
  [
    "return", RETURN;
    "function", FUNCTION;
    "int", INT;
    "float", FLOAT;
    "bool", BOOL;
    "matrix", MATRIX;
    "null", NULL;
    "while", WHILE;
    "for", FOR;
    "if", IF;
    "else", ELSE;
  ]
}

let digit = ['0'-'9']
let digits = digit+
let float = digits '.' digits
let quote = ['\'' \'']

rule tokenize = parse
  [' ' '	' '' '
'] { tokenize lexbuf }
(* 2.2 comments *)
| "//" { comment lexbuf }
| "/*" { comments lexbuf }
(* 2.1 types *)
| digits as lit { ILIT(int_of_string lit) }
| float as lit { FLIT(float_of_string lit) }
| "true" { BLIT(true) }
| "false" { BLIT(false) }
(* 2.6 operators *)
| '=' { ASSIGN }
| "+=" { PLUSASSIGN }
| "-=" { MINUSASSIGN }
| '+' { PLUS }
| '-' { MINUS }
| "*" { TIMES }
| '/\' { DIVIDE }
| '%' { MOD }
| "==" { EQ }
| "!=" { NEO }
| '<' { LT }
8.2.2 parser.mly
let parse_error s =
    begin
      try
        let start_pos = Parsing.symbol_start_pos ()
          and end_pos = Parsing.symbol_end_pos () in
        Printf.printf "File \"\%s\", line %d, characters %d-%d: \n"
        start_pos.pos_fname
        start_pos.pos_lnum
        (start_pos.pos_cnum - start_pos.pos_bol)
        (end_pos.pos_cnum - start_pos.pos_bol)
        with Invalid_argument(_)->() end;
        Printf.printf "Syntax error: \n" s;
        raise Parsing.Parse_error
    end

%{
  open Ast
  open Lexing

  let parse_error s =
    begin
      try
        let start_pos = Parsing.symbol_start_pos ()
          and end_pos = Parsing.symbol_end_pos () in
        Printf.printf "File \"\%s\", line %d, characters %d-%d: \n"
        start_pos.pos_fname
        start_pos.pos_lnum
        (start_pos.pos_cnum - start_pos.pos_bol)
        (end_pos.pos_cnum - start_pos.pos_bol)
        with Invalid_argument(_)->() end;
        Printf.printf "Syntax error: \n" s;
        raise Parsing.Parse_error
    end

%}

%token PLUS MINUS TIMES DIVIDE MOD
%token EQ NEQ LT LEQ GT GEQ
%token ASSIGN PLUSASSIGN MINUSASSIGN
%token LPAREN RPAREN LBRACK RBRACK LBRACE RBRACE COMMA SEMI
%token IF ELSE
%token WHILE FOR
%token RETURN FUNCTION
%token NULL
%token INT FLOAT BOOL MATRIX

%token <int> ILIT
%token <float> FLIT
%token <bool> BLIT
%token <string> ID
%token EOF

%left IF ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left TIMES DIVIDE MOD
%left PLUS MINUS
%left LBRACK RBRACK
%right NOT
%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

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

vdecl: dtype ID SEMI { $1, $2 }

fdecl: dtype FUNCTION ID LPAREN formals RPAREN LBRACE stmts RBRACE { 
  return = $1;
  fname = $3;
  formals = List.rev $5;
  stmts = List.rev $8;
 }

formals:
  nothing */ { [] } 
  |(dtype ID { [(1, 2)] }) 
  | formals COMMA dType ID { (3, 4) :: $1 }

dtype:
  INT { Int }
  | FLOAT { Float }
  | BOOL { Bool }
  | MATRIX { Matrix }

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

stmt:
  expr SEMI { Expr ($1) }
  | RETURN expr SEMI { Return ($2) }
  | dType ID SEMI { VDeclare($1, $2) }
  | assignstmt SEMI { AssignStmt($1) }
  | WHILE LPAREN expr RPAREN LBRACE stmts RBRACE {While($3, List.rev $6)}
  | FOR LPAREN assignstmt SEMI expr SEMI assignstmt RPAREN LBRACE stmts RBRACE {For($3, $5, $7, List.rev $10)}
  | IF LPAREN expr RPAREN LBRACE stmts RBRACE {If($3, List.rev $6)}
  | IF LPAREN expr RPAREN LBRACE stmts RBRACE ELSE LBRACE stmts RBRACE {IfElse($3, List.rev $6, List.rev $10)}
assignstmt:

dtype ID ASSIGN expr { VDeAssign($1, $2, $4) }

| ID PLUSASSIGN expr { Assign($1, Binop(Id($1), Add, $3)) } 
| ID MINUSASSIGN expr { Assign($1, Binop(Id($1), Sub, $3)) } 
| ID ASSIGN expr { Assign($1, $3) } 
| expr LBRACK expr COMMA expr RBRACK ASSIGN expr { MAssign($1, $3, $5, $8) } 

expr:

| ILIT { ILit($1) } 
| FLIT { FLit($1) } 
| BLIT { BLit($1) } 
| matrix { MLit($1) } 
| expr LBRACK expr COMMA expr RBRACK { Access($1, $3, $5) } 
| ID { Id($1) } 
| 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) } 
| ID LPAREN inputs RPAREN { Func($1, $3) } 
| MINUS expr %prec NOT { Unary(Neg, $2) } 
| NOT expr { Unary(Not, $2) } 
| expr AND expr { Binop($1, And, $3) } 
| expr OR expr { Binop($1, Or, $3) } 
| expr MOD expr { Binop($1, Mod, $3) } 
| expr EQ expr { Binop($1, Eq, $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) } 

inputs:

/* nothing */ { [] } 
| expr { [$1] } 
| expr COMMA inputs { $1 :: $3 } 

matrix:

| LBRACK matrix_row_list RBRACK { $2 } 

matrix_row_list:

| matrix_row { [$1] } 
| matrix_row SEMI matrix_row_list { $1 :: $3 } 

matrix_row:

| expr { [$1] } 
| expr COMMA matrix_row { $1 :: $3 }
8.2.3 ast.ml
type operator =
  | Add
  | Sub
  | Mul
  | Div
  | Mod
  | And
  | Or
  | Eq
  | Neq
  | Less
  | Leq
  | Greater
  | Geq

type uop = Neg | Not

type expr =
  | Binop of expr * operator * expr
  | ILit of int
  | FLit of float
  | BLit of bool
  | MLit of expr list list
  | Id of string
  | Func of string * expr list
  | Access of expr * expr * expr
  | Unary of uop * expr

type dtype = Int | Float | Bool | Matrix | Null

(*type elifstmt = Elif of expr * stmt list*)

(type assignstmt =
  | VDeAssign of dtype * string * expr
  | Assign of string * expr
  | MAssign of expr * expr * expr * expr

(type stmt =
  | Expr of expr
  | Return of expr
  | VDeclare of dtype * string
  | AssignStmt of assignstmt
  | If of expr * stmt list
  | IfElse of expr * stmt list * stmt list
  | For of assignstmt * expr * assignstmt * stmt list
  | While of expr * stmt list

(type bind = dtype * string
type fdecl = {
    return : dtype;
    fname : string;
    formals : bind list;
    stmts : stmt list;
}

type program = bind list * fdecl list

(* Pretty-printing functions from microc *)
let string_of_typ = function
    | Int -> "int"
    | Null -> "null"
    | Float -> "float"
    | Bool -> "Bool"
    | Matrix -> "matrix"

let string_of_op = function
    | Add -> "+
    | Sub -> "-
    | Mul -> "*
    | Div -> "/"\n    | Mod -> "%"
    | Eq -> "=="
    | Neq -> "!="
    | Less -> "<"
    | Leq -> "<="
    | Greater -> ">
    | Geq -> ">="
    | And -> "&&"
    | Or -> "||"

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

let rec string_of_expr = function
    | ILit l -> string_of_int l
    | FLit l -> string_of_float l
    | BLit l -> string_of_bool l
    | MLit l ->
        let string_of_row l = String.concat "" (List.map string_of_expr l) in
        String.concat "" (List.map string_of_row l)
    | Id s -> s
    | Binop (e1, o, e2) ->
        string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
    | Unary (o, e) -> string_of_uop o ^ string_of_expr e
    | Func (id, inputs) ->
        id ^ "(" ^ String.concat ", " (List.map string_of_expr inputs) ^ ");\n"
let string_of_expr e1 ^ " ^ string_of_expr e2 ^ " ^ string_of_expr e3

let string_of_assignstmt = function
  | VDeAssign (t, id, expr) ->
    "VDeAssign: " ^ string_of_typ t ^ id ^ string_of_expr expr ^ ";\n"
  | Assign (v, e) -> "Assign: " ^ v ^ " = " ^ string_of_expr e ^ ";\n"
  | MAssign (id, r, c, v) ->
    "MAssign: " ^ string_of_expr id ^ "[" ^ string_of_expr r ^ ", "
    ^ string_of_expr c ^ "] = " ^ string_of_expr v ^ ";\n"

let rec string_of_stmt = function
  | Expr expr -> string_of_expr expr ^ ";\n"
  | Return expr -> "return: " ^ string_of_expr expr ^ ";\n"
  | Assignstmt assignstmt -> string_of_assignstmt assignstmt
  | VDeclare (t, id) -> "VDeclare: " ^ string_of_typ t ^ " ^ id ^ ";\n"
  | While (e, ss) ->
    "While(" ^ string_of_expr e ^ ") { "
    ^ String.concat "\n" (List.map string_of_stmt ss)
    ^ " }\n"
  | For (as1, e, as2, ss) ->
    "For( " ^ string_of_assignstmt as1 ^ "; " ^ string_of_expr e ^ "; "
    ^ string_of_assignstmt as2 ^ ") { "
    ^ String.concat "\n" (List.map string_of_stmt ss)
    ^ " }\n"
  | If (e, ss) ->
    "If( " ^ string_of_expr e ^ ") {"
    ^ String.concat "\n" (List.map string_of_stmt ss)
    ^ " }\n"
  | IfElse (e, ss1, ss2) ->
    "If( " ^ string_of_expr e ^ ") {
    ^ String.concat "\n" (List.map string_of_stmt ss1)
    ^ " } Else{ 
    ^ String.concat "\n" (List.map string_of_stmt ss2)
    ^ " }\n"

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

let string_of_fdecl fdecl =
  "fdecl: " ^ fdecl.fname ^ "("
  ^ String.concat ", " (List.map snd fdecl.formals)
  ^ ")\n"
  ^ String.concat "" (List.map string_of_stmt fdecl.stmts)
  ^ "\n"

let string_of_program (vars, funcns) =
  String.concat "" (List.map string_of_vdecl vars)
  ^ "\n"
8.2.4 sast.ml

```ml
145   ^ String.concat "\n" (List.map string_of_fdecl funcs)
146```
open Ast

type sexpr = dtype * sx

and sx =
  | SBinop of sexpr * operator * sexpr
  | SId of string
  | SILit of int
  | SFLit of float
  | SBLit of bool
  | SMLit of sexpr list list
  | SFunc of string * sexpr list
  | SAccess of sexpr * sexpr * sexpr
  | SUnary of uop * sexpr

type sassignstmt =
  | SVDeAssign of dtye * string * sexpr
  | SAssign of string * sexpr
  | SMAssign of sexpr * sexpr * sexpr * sexpr

type sstmt =
  | SExpr of sexpr
  | SReturn of sexpr
  | SVDeclare of dtyp * string
  | SAssignStmt of sassignstmt
  | SIf of sexpr * sstmt list
  | SIfElse of sexpr * sstmt list * sstmt list
  | SFor of sassignstmt * sexpr * sassignstmt * sstmt list
  | SWhile of sexpr * sstmt list

type sbind = dtype * string

type sfdecl = {
  sreturn : dtype;
  sfname : string;
  sformals : sbind list;
  sstmts : sstmt list;
}

type sprogram = sbind list * sfdecl list

(* Pretty-printing functions from microc *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^
  (match e with
   | SILit l -> string_of_int l
   | SFLit l -> string_of_float l
let string_of_row l = String.concat "" (List.map string_of_sexpr l) in
_string_of_row l

let string_of_sassignstmt = function
| SAssign (v, e) -> "Assign: " ^ v ^ " = " ^ string_of_sexpr e ^ ";\n"
| SDeAssign (t, id, sexpr) ->
  "VDeAssign: " ^ string_of_typ t ^ id ^ string_of_sexpr sexpr ^ ";\n"
| SMAssign (id, r, c, v) ->
  "MAssign: " ^ string_of_sexpr id ^ "[" ^ string_of_sexpr r ^ "]", " ^ string_of_sexpr c ^ "]" = " ^ string_of_sexpr v ^ ";\n"

let rec string_of_sstmt = function
| SExpr sexpr -> string_of_sexpr sexpr ^ ";\n"
| SReturn sexpr -> "return: " ^ string_of_sexpr sexpr ^ ";\n"
| SAssignStmt sassignstmt -> string_of_sassignstmt sassignstmt
| SDeclare (t, id) -> "VDeclare: " ^ string_of_typ t ^ " " ^ id ^ ";\n"
| SWhile (e, ss) ->
  "While(" ^ string_of_sexpr e ^ ") { "
  ^ String.concat "\n" (List.map string_of_sstmt ss)
  ^ " }\n"
| SFor (as1, e, as2, ss) ->
  "For( " ^ string_of_sassignstmt as1 ^ "; " ^ string_of_sexpr e ^ ";
  " ^ string_of_sassignstmt as2 ^ ") { "
  ^ String.concat "\n" (List.map string_of_sstmt ss)
  ^ " }\n"
| SIf (e, ss) ->
  "If( " ^ string_of_sexpr e ^ ") {
  ^ String.concat "\n" (List.map string_of_sstmt ss)
  ^ " }\n"
| SIfElse (e, ss1, ss2) ->
  "If( " ^ string_of_sexpr e ^ ") {
  ^ String.concat "\n" (List.map string_of_sstmt ss1)
  ^ " } Else{ "
  ^ String.concat "\n" (List.map string_of_sstmt ss2)
let string_of_svdecl (t, id) = "vdecl: " ^ string_of_typ t ^ " id ^ ";\n"

let string_of_sfdecl fdecl =  
  "fdecl: " ^ fdecl.sfname ^ "("  
  ^ String.concat ", " (List.map snd fdecl.sformals)  
  ^ ")\n{"  
  ^ String.concat "" (List.map string_of_sstmt fdecl.sstmts)  
  ^ "}\n"

let string_of_sprogram (svars, sfuncs) =  
  String.concat "" (List.map string_of_svdecl svars)  
  ^ "\n"  
  ^ String.concat "\n" (List.map string_of_sfdecl sfuncs)

8.2.5 semant.ml
open Ast
open Sast
module StringMap = Map.Make (String)

let check (globals, functions) =
  let check_binds kind binds =
    List.iter
      (function
       | Null, b -> raise (Failure ("illegal null " ^ kind ^ " " ^ b))
       | _ -> ()
     )
    binds;
  let rec dups = function
    | [] -> ()
    | (_, n1) :: (_, n2) :: _ when n1 = n2 ->
      raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
    | _ :: t -> dups t
  in
  dups (List.sort (fun (_, a) (_, b) -> compare a b) binds)
in

(*** Check global variables ***)
check_binds "global" globals;

(*** Check functions ***)
let built_in_decls =
  let add_bind map (name, ftypes, ret) =
    StringMap.add name
    {
      return = ret;
      fname = name;
      (* Handles an list of formals types*)
      formals = List.map (fun t -> (t, "PLACEHOLDER")) ftypes;
      stmts = [];
    }
  map
  in
  List.fold_left add_bind StringMap.empty
  ["print", [ Int ], Null];
  "printb", [ Bool ], Null];
  "printf", [ Float ], Null];
  "printfm", [ Matrix ], Null];
  "rows", [ Matrix ], Int]);
  "cols", [ Matrix ], Int]);
  "zeros", [ Int; Int ], Matrix]);
in
let add_func map fd =
  let built_in_err = "function " ^ fd.fname ^ " may not be defined"
  and dup_err = "duplicate function " ^ fd.fname
  and make_err er = raise (Failure er) 
  and n = fd.fname (* Name of the function *) in 
  match fd with
  (* No duplicate functions or redefinitions of built-ins *)
  | _ when StringMap.mem n built_in_decls -> make_err built_in_err
  | _ when StringMap.mem n map -> make_err dup_err
  | _ -> StringMap.add n fd map
  in

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

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

  let _ = find_func "main" in

  let check_function func =
    check_binds "formal" func.formals;

  let check_assign lvaluet rvaluet err =
    if lvaluet = rvaluet then lvaluet else raise (Failure err)
  in

  let symbols =
    List.fold_left
      (fun m (ty, name) -> StringMap.add name ty m)
    StringMap.empty (globals @ func.formals)
  in

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

  let rec check_expr e env =
    match e with
    | Id n -> (type_of_identifier n env, SId n)
    | ILit l -> (Int, SILit l)
    | FLit l -> (Float, SFLit l)
    | BLit l -> (Bool, SBLit l)
    | MLit l ->
      let find_inner_type l =
        match l with

let t, _ = check_expr hd env in
  t
| _ -> Null

let find_type mat =
  match mat with (* tl *)
  | hd :: _ -> find_inner_type hd | _ -> Null

let my_type = find_type l in
let rec matrix_expr l =
  match l with
  | hd :: tl ->
    let ty, e = check_expr hd env in
    if ty != my_type then
      raise (Failure "Types in matrix do not match.");
    (ty, e) :: matrix_expr tl
  | _ -> []

  (Matrix, SMLit (List.map matrix_expr l))
| Binop (e1, op, e2) ->
 1068
  let t1, e1' = check_expr e1 env and t2, e2' = check_expr e2 env
in
  (* All binary operators require operands of the same type *)
| same = t1 = t2 in
  (* Determine expression type based on operator and operand types *)
  let ty =
    match op with
    | (Add | Sub | Mul | Div | Mod) when same && t1 = Int -> Int
    | (Add | Sub | Mul | Div | Mod) when same && t1 = Float -> Float
    | (Add | Sub) when same && t1 = Matrix -> Matrix
    | (Eq | Neq) when same -> Bool
    | (Less | Leq | Greater | Geq) when same && (t1 = Int || t1 = Float)
    -> Bool
    | (And | Or) when same && t1 = Bool -> Bool
    | Mul when same && t1 = Matrix -> Matrix
    | Mul when t1 = Matrix && (t2 = Float || t2 = Int) -> Matrix
    | Mul when t2 = Matrix && (t1 = Float || t1 = Int) -> Matrix
    | _ ->
      raise (Failure
        ("illegal binary operator" ^ string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^ string_of_typ t2 ^ " in " ^ string_of_expr e))
  in
  (ty, SBinop ((t1, e1'), op, (t2, e2')))

| hd :: _ ->
  let t, _ = check_expr hd env in
  t
| _ -> Null

Unary (op, e1) as e ->
  let tp, c = check_expr e1 env in
  let ty =
    match op with
    | Neg when tp = Int || tp = Float -> tp
    | Not when tp = Bool -> Bool
    | _ ->
      raise
        (Failure
          ("illegal unary operator " ^ string_of_uop op
           ^ string_of_typ tp ^ " in " ^ string_of_expr e))
      in
      (ty, SUnary (op, (tp, c)) |
        Func (id, inputs) as func ->
          let fd = find_func id in
          let param_length = List.length fd.formals in
          if List.length inputs != param_length then
            raise
              (Failure
                ("expecting " ^ string_of_int param_length ^ " arguments
                in "
                ^ string_of_expr func))
          else
            let check_call (ft, _) e =
              let et, e' = check_expr e env in
              let err =
                "illegal argument found " ^ string_of_typ et ^ " expected
                "
                ^ string_of_typ ft ^ " in " ^ string_of_expr e
              in
              (check_assign ft et err, e')
            in
            let args' = List.map2 check_call fd.formals inputs in
            (fd.return, SFunc (id, args')) |
          Access (m, r, c) ->
            let r_t, _ = check_expr r env in
            let c_t, _ = check_expr c env in
            if r_t != Int || c_t != Int then
              raise (Failure "index must be of type int");
            let _, _ = check_expr m env in
            (Float, SAccess (check_expr m env, check_expr r env, check_expr
c env))
            in
            (* Check stmts - Return a semantically-checked statement i.e.
containing sexprs *)
          let rec check_stmt env stmt =
            match stmt with
Expr e -> (SExpr (check_expr e env), env)
Return e ->
  let t, e' = check_expr e env in
  if t = func.return then (SReturn (t, e'), env)
  else
    raise
      (Failure
        ("return gives " ^ string_of_typ t ^ " expected "
         ^ string_of_typ func.return ^ " in " ^ string_of_expr e))
VDeclare (t, s) -> (SVDeclare (t, s), StringMap.add s t env)
AssignStmt astmt -> (match astmt with
  | VDeAssign (t, s, e) ->
    let t', _ = check_expr e env in
    let decl_type = check_assign t t' "Type not correct" in
    (SAssignStmt (SVDeAssign (t, s, check_expr e env)),
      StringMap.add s decl_type env )
  | Assign (s, e) ->
    let left_typ = type_of_identifier s env
    and right_typ, e' = check_expr e env in
    let error = "Illegal assignment!" in
    ignore (check_assign left_typ right_typ error);
    (SAssignStmt (SAssign (s, (right_typ, e'))), env)
  | MAssign (m, r, c, e) ->
    let r_t, _ = check_expr r env in
    let c_t, _ = check_expr c env in
    let e_t, _ = check_expr e env in
    if r_t != Int || c_t != Int then
      raise (Failure "index must be of type int")
    if e_t != Float then raise (Failure "value must be of type
      float")
    (SAssignStmt
      (SMAssign
        (check_expr m env,
         check_expr r env,
         check_expr c env,
         check_expr e env )),
        env ))
  | While (e, stmts) ->
    let typ, _ = check_expr e env in
    if typ != Bool then raise (Failure "Expect to have a Bool type
      here.");
    (SWhile (check_expr e env, check_stmts env stmts), env)
  | For (astmt, e2, astmt2, stmts) ->
    let sastmt, env2 = check_assignstmt env astmt in
    let sastmt2, _ = check_assignstmt env2 astmt2 in
    let typ, _ = check_expr e2 env2 in
    if typ != Bool then raise (Failure "Expect to have a Bool type
      here.");
(SFor (sastmt, check_expr e2 env2, sastmt2, check_stmts env2 stmts),
    env )
        | If (e, stmts) ->
        let typ, _ = check_expr e env in
        if typ != Bool then raise (Failure "Expect to have a Bool type here.");
        (SIf (check_expr e env, check_stmts env stmts), env)
        | IfElse (e, stmts1, stmts2) ->
        let typ, _ = check_expr e env in
        if typ != Bool then raise (Failure "Expect to have a Bool type here.");
        (SIfElse (check_expr e env, check_stmts env stmts1, check_stmts env stmts2),
            env )
        and check_assignstmt env astmt =
        match astmt with
        | VDeAssign (t, s, e) ->
        let t', _ = check_expr e env in
        let decl_type = check_assign t t' "Type not correct" in
        (SVDeAssign (t, s, check_expr e env), StringMap.add s decl_type env)
        | Assign (s, e) ->
        let left_typ = type_of_identifier s env
        and right_typ, e' = check_expr e env in
        let error = "Illegal assignment!" in
        ignore (check_assign left_typ right_typ error);
        (SAssign (s, (right_typ, e')), env)
        | _ -> raise (Failure "Illegal assignment in for-loop header!")
        and check_stmts env stmts =
        match stmts with
        | [ (Return _ as s) ] ->
        let st, _ = check_stmt env s in
        [ st ]
        | Return _ :: _ -> raise (Failure "Unreachable statements after return")
        | s :: ss ->
        let st, env2 = check_stmt env s in
        st :: check_stmts env2 ss
        | [] -> []
    in
    {
    sreturn = func.return;
    sfname = func.fname;
    sformals = func.formals;
    sstmts = check_stmts symbols func.stmts;
    }
    in
8.2.6 codegen.ml

281 (globals, List.map check_function functions)
(* Code generation: translate takes a semantically checked AST and produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial
http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:
http://llvm.moe/
http://llvm.moe/ocaml/
*)

module L = Llvm
module A = Ast
open Sast
module StringMap = Map.Make (String)

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
  let context = L.global_context () in
  (* Create the LLVM compilation module into which we will generate code *)
  let the_module = L.create_module context "Marble" in
  (* Get types from the context *)
  let i32_t = L.i32_type context
  and i8_t = L.i8_type context
  and i1_t = L.i1_type context
  and float_t = L.double_type context
  and void_t = L.void_type context in
  (* Return the LLVM type for a Marble type *)
  let ltype_of_typ = function
    | A.Int -> i32_t
    | A.Float -> float_t
    | A.Bool -> i1_t
    | A.Matrix -> L.pointer_type float_t
    | A.Null -> void_t
  in
  (* Create a map of global variables after creating each *)
  let global_vars : L.llvalue StringMap.t =
    let global_var m (t, n) =
      let init =
        match t with
          | A.Int -> L.const_int (ltype_of_typ t) 0

(* built-in function *)
let printf_t : L.lltype =
  L.var_arg_function_type i32_t [ | L.pointer_type i8_t |]
in
let printf_func : L.llvalue =
  L.declare_function "printf" printf_t the_module
in
let printfm_t : L.lltype =
  L.function_type i32_t [ | L.pointer_type float_t |]
in
let printfm_func : L.llvalue =
  L.declare_function "printfm" printfm_t the_module
in
(* matrix addition *)
let addmf_t : L.lltype =
  L.function_type (L.pointer_type float_t)
  [ | L.pointer_type float_t; L.pointer_type float_t |]
in
let addmf_func : L.llvalue = L.declare_function "addmf" addmf_t
  the_module in

(* subtraction *)
let submf_t : L.lltype =
  L.function_type (L.pointer_type float_t)
  [ | L.pointer_type float_t; L.pointer_type float_t |]
in
let submf_func : L.llvalue = L.declare_function "submf" submf_t
  the_module in

(* scalar multiplication *)
let scalarmf_t : L.lltype =
  L.function_type (L.pointer_type float_t)
  [ | float_t; L.pointer_type float_t |]
in
let scalarmf_func : L.llvalue =
  L.declare_function "scalarmf" scalarmf_t the_module
in
(* matrix multiplication *)
let multiplicationmf_t : L.lltype =
  L.function_type (L.pointer_type float_t)
let multiplicationmf_func : L.llvmvalue =
  L.declare_function "multiplicationmf" multiplicationmf_t the_module

let get_matrix_rows matrix builder =
  let typ = L.string_of_lltype (L.type_of_matrix) in
  let ret =
    match typ with
    | "double*" ->
      let rows = L.build_load matrix "rows" builder in
      L.build_fptosi rows i32_t "rowsint" builder
    | _ -> L.build_load matrix "rows" builder
  in
  ret

let get_matrix_cols matrix builder =
  let ptr =
    L.build_in_bounds_gep matrix [| L.const_int i32_t 1 |] "ptr" builder
  in
  let typ = L.string_of_lltype (L.type_of_ptr) in
  let ret =
    match typ with
    | "double*" ->
      let cols = L.build_load ptr "cols" builder in
      L.build_fptosi cols i32_t "colsint" builder
    | _ -> L.build_load ptr "cols" builder
  in
  ret

let function_decls : (L.llvmvalue * sfdecl) StringMap.t =
  let function_decl m fdecl =
    let name = fdecl.sfname in
    let formal_types =
      Array.of_list (List.map (fun (t, _) -> ltype_of_typ t) fdecl.sformals)
    in
    let ftype = L.function_type (ltype_of_typ fdecl.sreturn) formal_types in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m
  in
  List.fold_left function_decl StringMap.empty functions
let build_function_body fdecl =
    let the_function, _ = StringMap.find fdecl.sfname function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in
    let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
    and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder in

    let local_vars = Hashtbl.create 20 in
    let add_formal (t, n) p =
        L.set_value_name n p;
        let local = L.buildalloca (ltype_of_typ t) n builder in
        ignore (L.build_store p local builder);
        Hashtbl.add local_vars n local
    in

    (* add formals *)
    List.iter2 add_formal fdecl.sformals (Array.to_list (L.params the_function));

    (* Return the value for a variable or formal argument. Check local names first, then global names *)
    let lookup n =
        try Hashtbl.find local_vars n
        with Not_found -> ( try StringMap.find n global_vars
            with Not_found ->
            raise (Failure ("Runtime: undeclared identifier " ^ n)))
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder ((_, e) : sexpr) =
        match e with
        | SILit i -> L.const_int i32_t i
        | SFLit f -> L.const_float float_t f
        | SBLit b -> L.const_int i1_t (if b then 1 else 0)
        | SMLit l ->
            let find_inner_type l =
                match l with
                | hd :: _ ->
                    let t, _ = hd in
                    t
                | _ -> A.Int
                in
            let find_type mat =
                match mat with hd :: _ -> find_inner_type hd | _ -> A.Int
                in
            let my_type = find_type l in
            let make_matrix =
match my_type with
  | A.Float ->
    let count a = List.fold_left (fun x _ -> x + 1) 0 a in
    let rows = float_of_int (count l) in
    let cols = float_of_int (count (List.hd l)) in
    let rec valid_dims m =
      match m with
        | hd :: tl ->
          if count hd == count (List.hd l) then valid_dims tl
           else false
        | _ -> true
      in
    if not (valid_dims l) then
      raise
        (Failure
         "all rows of matrices must have the same number of \
          elemens")
    else
      (* allocate space 2 + rows * cols*)
      let matrix =
        L.build_malloc
        (L.array_type float_t
         (2 + (int_of_float rows * int_of_float cols)))
          "matrix" builder
        in

      let eval_row row =
        List.fold_left
          (fun eval_row x -> eval_row @ [ expr builder x ])
           [] row
        in
      let unfolded =
        List.fold_left (fun unflfd row -> unflfd @ eval_row row)
           [] l
        in
      let unfolded =
        [ L.const_float float_t rows; L.const_float float_t cols ]
          @ unfolded
        in
      let rec store idx lst =
        match lst with
          | hd :: tl ->
            let ptr =
              L.build_in_bounds_gep matrix
                [ [ L.const_int i32_t 0; L.const_int i32_t idx ]]
                  "ptr" builder
            in
ignore (L.build_store hd ptr builder);
store (idx + 1) tl
  | _ -> ()
in
store 0 unfolded;
L.build_in_bounds_gep matrix
  [ | L.const_int i32_t 0; L.const_int i32_t 0 |
  "matrix" builder
  | _ -> raise (Failure "invalid matrix type")
in
make_matrix
  | SId s -> L.build_load (lookup s) s builder
  | SBinop (((A.Matrix, _)) as m1), op, m2) ->
    let m1' = expr builder m1 and m2' = expr builder m2 in
    let ret =
      match op with
      | A.Add -> L.build_call addmf_func [ | m1'; m2' | ] "addmf"
builder
      | A.Sub -> L.build_call submf_func [ | m1'; m2' | ] "submf"
builder
      | A.Mul ->
        let t', _ = m2 in
        let ret_val' =
          match t' with
          | A.Int ->
            let scalar =
              L.build_sitofp m2' float_t "scalar" builder
          in
          L.build_call scalarmf_func [ | scalar; m1' | ] "scalarmf"
builder
          | A.Float ->
            L.build_call scalarmf_func [ | m2'; m1' | ] "scalarmf"
builder
          | _ ->
            L.build_call multiplicationmf_func [ | m1'; m2' | ] "matmf"
builder
          in
          ret_val'
        | _ -> raise (Failure "internal error: semant should have rejected")
in
ret
  | SBinop ((_ as m1), (_ as op), ((A.Matrix, _)) as m2)) ->
    let m1' = expr builder m1 and m2' = expr builder m2 in
    let ret =
      match op with
      | A.Mul ->
        let t, _ = m1 in
let ret_val =
match t with
| A.Int ->
  let scalar =
    L.build_sitofp m1' float_t "scalar" builder
  in
  L.build_call scalarmf_func [| scalar; m2' |]

"scalarmf"
| A.Float ->
  L.build_call scalarmf_func [| m1'; m2' |] "scalarm"
  builder
  | _ -> raise (Failure "should be caught elsewhere")
  in
  ret_val
  | _ -> raise (Failure "internal error: semant should have rejected")
  in
  ret
| SBinop (((A.Float, _) as e1), op, e2) ->
  let e1' = expr builder e1 and e2' = expr builder e2 in
  (match op with
   | A.Add -> L.build_fadd
   | A.Sub -> L.build_fsub
   | A.Mul -> L.build_fmul
   | A.Div -> L.build_fdiv
   | A.Mod -> L.build_frem
   | A.Eq -> L.build_fcmp L.Fcmp.Oeq
   | A.Neq -> L.build_fcmp L.Fcmp.One
   | A.Less -> L.build_fcmp L.Fcmp.Olt
   | A.Leq -> L.build_fcmp L.Fcmp.Ole
   | A.Greater -> L.build_fcmp L.Fcmp.Ogt
   | A.Geq -> L.build_fcmp L.Fcmp.Oge
   | A.And | A.Or ->
     raise
     (Failure
      "internal error: semant should have rejected and/or on float")
    e1' e2' "tmp" builder
| SBinop (e1, op, e2) ->
  let e1' = expr builder e1 and e2' = expr builder e2 in
  (match op with
   | A.Add -> L.build_add
   | A.Sub -> L.build_sub
   | A.Mul -> L.build_mul
   | A.Div -> L.build_sdiv
   | A.Mod -> L.build_srem
   | A.Eq -> L.build_icmp L.Icmp.Eq
   | A.Neq -> L.build_icmp L.Icmp.Ne
   | A.Less -> L.build_icmp L.Icmp.Slt
A. Leq -> L.build_icmp L.Icmp.Sle
A. Greater -> L.build_icmp L.Icmp.Sgt
A. Geq -> L.build_icmp L.Icmp.Sge
A. And -> L.build_and
A. Or -> L.build_or

e1' e2' "tmp" builder

| SU unary (op, ((t, _) as e)) ->
| (* Unary and Negate *)
| let e' = expr builder e in
| (match op with
| A. Neg when t = A. Float -> L.build_fneg
| A. Neg -> L.build_neg
| A. Not -> L.build_not)
| e' "tmp" builder

| SA access (((_, _) as m), r, c) ->
| (* get desired pointer location *)
| let matrix = expr builder m
| and row_idx = expr builder r
| and col_idx = expr builder c in
| let cols = get_matrix_cols matrix builder in
| (* row = row_idx * cols *)
| let row = L.build_mul row_idx cols "row" builder in
| (* row_col = (row_idx * cols) + col_idx *)
| let row_col = L.build_add row col_idx "row_col" builder
| and offset = L.const_int i32_t 2 in
| (* idx = 2 + (row_idx * cols) + col_idx *)
| let idx = L.build_add offset row_col "idx" builder in
| let ptr = L.build_in_bounds_gep matrix [ | idx | ] "ptr" builder
| L.build_load ptr "element" builder
| (* Function call *)
| SF func ("print", [ e ]) | SF func ("printf", [ e ]) ->
| L.build_call printf_func
| | [ | int_format_str; expr builder e | ]
| "printf" builder
| SF func ("print", [ e ]) ->
| L.build_call printf_func
| | [ | float_format_str; expr builder e | ]
| "printf" builder
| SF func ("printf", [ e ]) ->
| L.build_call printf_func
| | [ | expr builder e | ] "printfm" builder
| SF func ("rows", [ e ]) ->
| L.build_call printfm_func [ | expr builder e | ] "printfm" builder
| SF func ("cols", [ e ]) ->
| L.build_call printfm_func [ | expr builder e | ] "printfm" builder
| SF func ("rows", [ e ]) ->
| get_matrix_rows matrix builder
| SF func ("cols", [ e ]) ->
| get_matrix_cols matrix builder
| SF func ("zeros", [ (_), rows); (_, cols) ]) -> (match (rows, cols) with
| SF func ("zeros", [ (_), rows); (_, cols) ]) -> (match (rows, cols) with
| SF func ("zeros", [ (_), rows); (_, cols) ]) -> (match (rows, cols) with
|
if \( r == 0 \) || \( c == 0 \) then
    raise (Failure "rows and cols can't be zero")
else
    let matrix =
        L.build_malloc
        (L.array_type float_t (2 + (r * c)))
        "matrix" builder
    in
    let eval_row row =
        List.fold_left
        (fun eval_row _ -> eval_row @ [ L.const_float float_t 0.0 ])
        [] row
    in
    let unfolded =
        List.fold_left (fun unfld row -> unfld @ eval_row row)
        [] []
    in
    let unfolded =
        [ L.const_float float_t (float_of_int r);
          L.const_float float_t (float_of_int c);
        ]
        @ unfolded
    in
    let rec store idx lst =
        match lst with
        | hd :: tl ->
            let ptr =
                L.build_in_bounds_gep matrix
                ([] L.const_int i32_t 0; L.const_int i32_t idx [])
                "ptr" builder
            in
            ignore (L.build_store hd ptr builder);
            store (idx + 1) tl
            | _ -> ()
        in
    store 0 unfolded;
    L.build_in_bounds_gep matrix
    ([] L.const_int i32_t 0; L.const_int i32_t 0 [])
    "matrix" builder
    | _ -> raise (Failure "Rows and cols of a matrix must be integers.")
    | SFunc (f, args) ->
        let fdef, fdecl = StringMap.find f function_decls in
        let llargs = List.rev (List.map (expr builder) (List.rev args))
    in
    let result =
match fdecl.sreturn with A.Null -> "null" | _ -> f ^ "_result"
    in
    L.build_call fdef (Array.of_list llargs) result builder
  in

  (* LLVM insists each basic block end with exactly one "terminator"
  instruction that transfers control. This function runs "instr
  builder" if the current block does not already have a terminator. Used,
  e.g., to handle the "fall off the end of the function" case. *)
  let add_terminal builder instr =
    match L.block_terminator (L.insertion_block builder) with
    | Some _ -> ()
    | None -> ignore (instr builder)
  in
  let rec stmt builder = function
  | SExpr e ->
    ignore (expr builder e);
    builder
  | SReturn e ->
    ignore
    (match fdecl.sreturn with
    (* Special "return nothing" instr *)
    | A.Null -> L.build_ret_void builder
    | A.Matrix ->
      let e' = expr builder e in
      L.build_ret
      (L.build_bitcast e' (ltype_of_typ A.Matrix) "tmp"
       builder)
    | _ -> L.build_ret (expr builder e) builder);
    builder
  | SVDeclare (t, s) ->
    let local_var = L.build_alloca (ltype_of_typ t) s builder in
    Hashtbl.add local_vars s local_var;
    builder
  | SAssignStmt sastmt -> (match sastmt with
  | SVDeAssign (t, s, se) ->
    let e' = expr builder se in
    L.set_value_name s e';
    let local_var = L.build_alloca (ltype_of_typ t) s builder in
    ignore (L.build_store e' local_var builder);
    Hashtbl.add local_vars s local_var;
    builder
  | SAssign (s, se) ->
    let e' = expr builder se in
    ignore (L.build_store e' (lookup s) builder);
    builder
  | SMAssign (m, r, c, e) ->
(* get desired pointer location *)
let matrix = expr builder m
and row_idx = expr builder r
and col_idx = expr builder c in
let cols = get_matrix_cols matrix builder in
(* row = row_idx * cols *)
let row = L.build_mul row_idx cols "row" builder in
(* row_col = (row_idx * cols) + col_idx *)
let row_col = L.build_add row row_idx cols "row_col" builder in
let offset = L.const_int i32_t 2 in
(* idx = 2 + (row_idx * cols) + col_idx *)
let idx = L.build_add offset row_col offset "idx" builder in
let ptr = L.build_in_bounds_gep matrix [| idx |] "ptr" builder in

(* update value at that location *)
let e' = expr builder e in
let m_typ = L.string_of_lltype (L.type_of matrix)
and e_typ = L.string_of_lltype (L.type_of e') in
let e_fixed = match (m_typ, e_typ) with
| "double*", "i32" ->
  L.build_uitofp e float_t "float_e" builder
| _ -> e'
in
ignore (L.build_store e_fixed ptr builder);
builder in
| SWhile (se, sstmts) ->
  let se_bb = L.append_block context "while" the_function in
  ignore (L.build_br se_bb builder);
  let body_bb = L.append_block context "while_body" the_function
  in
  add_terminal
  (List.fold_left stmt (L.builder_at_end context body_bb) sstmts)
  (L.build_br se_bb);
  let pred_builder = L.builder_at_end context se_bb in
  let bool_val = expr builder se in
  let merge_bb = L.append_block context "merge" the_function in
  ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
  L.builder_at_end context merge_bb
| SFor (sastmt, se2, sastmt2, sstmts) ->
  List.fold_left stmt builder
  [ SAssignStmt sastmt; SWhile (se2, sstmts @ [ SAssignStmt sastmt2 ]));
  ]
| SIf (se, sstmts) ->
  let bool_val = expr builder se in
  let merge_bb = L.append_block context "merge" the_function in
  let b_br_merge = L.build_br merge_bb in
let then_bb = L.append_block context "then" the_function in
add_terminal
(List.fold_left stmt (L.builder_at_end context then_bb) sstmts)
  b_br_merge;
let else_bb = L.append_block context "else" the_function in
add_terminal
(List.fold_left stmt (L.builder_at_end context else_bb) [])
  b_br_merge;
ignore (L.build_cond_br bool_val then_bb else_bb builder);
L.builder_at_end context merge_bb
| SIIfElse (se, sstmts1, sstmts2) ->
  let bool_val = expr builder se in
  let merge_bb = L.append_block context "merge" the_function in
  let b_br_merge = L.build_br merge_bb in
  let then_bb = L.append_block context "then" the_function in
  add_terminal
  (List.fold_left stmt (L.builder_at_end context then_bb) sstmts1)
  b_br_merge;
  let else_bb = L.append_block context "else" the_function in
  add_terminal
  (List.fold_left stmt (L.builder_at_end context else_bb) sstmts2)
  b_br_merge;
  ignore (L.build_cond_br bool_val then_bb else_bb builder);
L.builder_at_end context merge_bb
(* Build the code for each statement in the function *)
let builder = List.fold_left stmt builder fdecl.sstmts in

(* Add a return if the last block falls off the end *)
add_terminal builder
  (match fdecl.sreturn with
   | A.Null -> L.build_ret_void
   | A.Float -> L.build_ret (L.const_float float_t 0.0)
   | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in
List.iter build_function_body functions;
the_module
(* 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: ./microc.native [-a|-s|-l|-c] [file.mc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

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

8.2.8 matrix_helper.c
/*
 *  matrix_helper.c
 *  reference:
 *  https://github.com/emilydringel/MATRIX_MANIA/blob/main/c_functions/matrix_functions.c
 */

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

void printmf(double* element_one) /* prints the matrix<int> */
{
    int rows = (int) element_one[0];
    int cols = (int) element_one[1];
    int size = rows * cols;
    char matrix[size+100]; /* maybe should be better about size?? */
    strcpy(matrix, "");
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
            int location = 2 + (i*cols) + j;
            char buffer[1000];
            sprintf(buffer, "%f", element_one[location]);
            strcat(matrix, buffer);
            if(j!=cols-1){
                strcat(matrix, " ");
            }
        }
        if(i!=rows-1){
            strcat(matrix, "\n");
        }
    }
    printf("%s\n", matrix);
}

double* addmf(double* m1, double* m2)
{
    if(m1[0]!=m2[0] || m1[1]!=m2[1]){  
        printf("RUNTIME ERROR: matrices being added do not have the same dimensions.\n");
        exit(1);
    }
    int rows = (int) m1[0];
    int cols = (int) m1[1];
    int size = 2 + rows * cols;
    double *empty = malloc(size * sizeof(double));
    empty[0] = rows;
    empty[1] = cols;
for (int i = 2; i < size; i++) {
    empty[i] = m1[i] + m2[i];
} return empty;
}

double* submf(double* m1, double* m2) {
    if(m1[0]!=m2[0] || m1[1]!=m2[1]){
        printf("RUNTIME ERROR: matrices being subtracted do not have the same dimensions.\n");
        exit(1);
    }
    int rows = (int) m1[0];
    int cols = (int) m1[1];
    int size = 2 + rows * cols;
    double *empty = malloc(size * sizeof(double));
    empty[0] = rows;
    empty[1] = cols;
    for (int i = 2; i < size; i++) {
        empty[i] = m1[i] - m2[i];
    } return empty;
}

double* scalarmf(double x, double* m){
    int rows = (int) m[0];
    int cols = (int) m[1];
    int size = 2 + rows * cols;
    double *empty = malloc(size * sizeof(double));
    for (int i = 0; i < size; i++) {
        empty[i] = m[i];
        if(i>=2){
            empty[i] *= x;
        }
    } return empty;
}

double* multiplicationmf(double* m1, double* m2){
    if(m1[1]!=m2[0]){  
        printf("RUNTIME ERROR: matrices being multiplied do not have complementary dimensions.\n");
        exit(1);
    }
    int rows_one = (int) m1[0];
    int rows_two = (int) m2[0];
    int cols_one = (int) m1[1];
    int cols_two = (int) m2[1];
    double *empty = malloc(rows_one * cols_two * sizeof(int));
empty[0] = (double) rows_one;
empty[1] = (double) cols_two;
for(int row=0; row<rows_one; row++){
    for(int col=0; col<cols_two; col++){
        empty[2+(cols_two*row)+col] = 0;
        for(int val=0; val<cols_one; val++){
            double x = m1[2+cols_one*row+val]*m2[2+cols_two*val+col];
            empty[2+cols_two*row+col] += x;
        }
    }
}
return empty;