1. Introduction:

1.1 Motivation

Matrices are a useful tool to represent finite sets of data across a wide range of subject matter, including the hard sciences, mathematics, engineering, and computer science. Moreover, basic matrix operations, including matrix multiplication, summation, subtraction, inversion, and finding the transpose, eigenvalues, and eigenvectors of matrices, have been unusually difficult to perform without the use of external packages in languages like Python and R. The purpose of our language, The MatriCs, is to not only simplify these computations, but also to reduce the running time of matrix operations. Using C-like syntax, our language is tailored for programmers familiar with C, but not necessarily familiar with popular “data science” packages, like SciPy (Python), NumPy (Python), and Matrix (R).

1.2 Background

MatriCs is a strongly typed language that combines a C-like syntax with a whole host of special operators. These operators enable the user to perform fundamental computations involving linear algebra, including but not limited to calculating a matrix’s transpose and inverse, carrying out matrix multiplication, and extracting submatrices through slicing. At the very core of our language is the special data type: matrix. MatriCs will compile to LLVM.
2. Language Tutorial:

2.1 Setting Up the Environment

In order to use MatriCs in the appropriate environment please follow the steps provided in the official Github page of MatriCs. The link is given below:

https://github.com/mannykoum/MatriCs/tree/dev/microc-llvm

Please note that different operating systems have different have separate instructions. Upon successful completion of the environment setup, the configuration files can be found here:

https://github.com/mannykoum/MatriCs/tree/dev

The configuration files should be cloned from the official Github site to ensure that the language runs without any problems.

2.2 Building and Using the Language

2.2.1 Building

Upon complete and successful installation of the MatriCs environment, the language is ready for building. Before using the language, the configuration files have to be built. In order to do this, simply run the following command:

> ./build.sh

After successful build, the compiler will output a “Build successful” message.

2.2.2 Compiling

MatriCs compiler supports three different kinds of compilation flags:
   -a : Prints the AST
   -l : Generates LLVM but doesn’t check it
   -c : Generates LLVM and compiles the given file
If the file is compiled successfully, the following command should generate the LLVM output of the input file:

`.neo.native -c hello.neo > hello.ll`

2.3 Basics of MatriCs

MatriCs has the following data types that must be declared at the beginning of body of the function:

- Integers
- Booleans
- Strings
- Voids
- N dimensional Matrices

An example of a variable declaration is given below:

```c
int main (){  
  bool x; //declaration of variable x with data type boolean 
  x = true; //setting x is equal to true - note that this done on a seperate 
  printb(x); //printing x to the standard output 
  return 0; 
}
```

2.3.1 Global Variables

Global variables are allowed in MatriCs and therefore any global variable declared outside of the scope of any function will be accessed by any of the functions in the program. For instance, in the following example, x can be reached from any of the functions:

```c
int x;
```
```c
int main(){
    int y;
    y = 0;
    if(y == 1){
        x = 1;
    }
    else{
        x = 2;
    }
    return 0;
}
```

2.3.2 Operators

MatriCs supports the following binary operators:

- Arithmetic (+, -, *, /, %)
- Relational (==, !=, <, <=, >, >=)
- Logical (!, &&, ||)

… and the following unary operators:

- Negation (!)
- Increment/Decrement (++, --)

For a detailed set of examples about the implementation of operators please check the test codes attached.

2.3.3 Vectors and Matrices

MatriCs provides its user the ease of manipulating vectors and matrices of any dimension. While most of the operators are designed for two dimensional matrices or below, the user is given the freedom to declare, define and print matrices of any dimension.

An example matrix and vector declaration is given below. Note that the matrices and vectors support only the integer or float data types. Other data types are not supported.
```c
#include <stdlib.neo>

int main() {
    int[2,2,2] a;
    a = [[[1,2], [3,4]],
         [[5,6], [7,8]]];
    print3d(a);
}
```

The function given above declares and defines a three dimensional 2x2x2 matrix and calls the standard library function `print3d` to print the matrix in a formatted way.

### 2.3.4 Standard Library

The MatriCs standard library contains many matrix manipulation functions to give its user ease of working with matrices. Standard library functions include transpose, add, subtract, identity for two dimensional matrices and print function for up to three dimensional matrices. The user must include the header “#include `<stdlib.neo>`” to make use of the matrix functions.

### 2.3.5 Control Flow

Control flow is exactly the same as the C syntax and includes if/else/else if statements, while and for loops as well as return. The following programs display the simple statements and loops:

```c
//while loop
int main() {
    int i;
    i = 1;
    while (i < 5) {
        print_int(i);
    }
}
```
2.3.6 Defining Functions

MatriCs allows its users to define and use their own functions. Function declaration and definition are same as the C syntax. Functions can return void, string, int and float data types. An example function and its calling in the main is given below:

```c
int main() {
    int[4] a;
    a = [1, 2, 3, 4];
    if (a[1] == 2) {
        printb(true);
    } else {
        printb(false);
    }
    return 0;
}
```
void print3d(int[2,2,2] c) {
    int i;
    int j;
    int k;
    for(i = 0; i < 2; i++) {
        print("[");
        for(j = 0; j < 2; j++) {
            print("[");
            for(k = 0; k < 2; k++) {
                if(k != 1) {
                    print_int(c[i,j,k]);
                    print(", ");
                } else {
                    print_int(c[i,j,k]);
                }
            }
            if(j != 1) {
                print("]", "");
            } else {
                print("]");
            }
        }
        if(j != 1) {
            print("]");
        } else {
            print("]");
        }
    }
    print("]
");
}

int main() {
    int[2,2,2] a;
    a = [[[1,2], [3,4]],
         [[5,6], [7,8]]];
    print3d(a);
}
3. Language Reference Manual:

3.1 Lexical Elements

3.1.1 Tokens
Our language can be broken down into six categories of tokens: identifiers, keywords, literals, operators, punctuation, and comments. Whitespace is used to separate tokens but otherwise will be ignored. Indentation should be used for stylistic purposes but is not necessary for the proper functioning of MatriCs programs.

3.1.2 Identifiers
Identifiers are strings used for naming different elements, such as variables and
functions. Identifiers must begin with a letter, but can contain digits and underscores as well.

These rules are described by the definitions involving regular expressions below:

identifier := (letter) (letter | digit | underscore)*
digit := '0' - '9'
letter := uppercase_letter | lowercase_letter
uppercase_letter := 'A' - 'Z'
lowercase_letter := 'a' - 'z'

3.1.3 Keywords

The following literals cannot be used as identifiers. They are also case sensitive.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>Similar to C conditional</td>
</tr>
<tr>
<td>else if</td>
<td>Similar to C conditional</td>
</tr>
<tr>
<td>else</td>
<td>Similar to C conditional</td>
</tr>
<tr>
<td>for</td>
<td>Similar to C for loop</td>
</tr>
<tr>
<td>while</td>
<td>Similar to C while loop</td>
</tr>
<tr>
<td>return</td>
<td>Return from function</td>
</tr>
<tr>
<td>null</td>
<td>No data</td>
</tr>
<tr>
<td>void</td>
<td>Returns nothing</td>
</tr>
<tr>
<td>import</td>
<td>Import external libraries for extended functions</td>
</tr>
</tbody>
</table>

3.1.4 Punctuation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>;</td>
<td>Used to end a statement, as well as to define each row of a matrix.</td>
</tr>
<tr>
<td>{}</td>
<td>Curly brackets are used to enclose functions, while and for loops, and if statements. In other words, they are used to delineate the scope of blocks of code in the program. They are also used to define the special data type, matrix.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>( )</td>
<td>Use to specify and pass arguments for a function and the precedence of operators. Also used to enclose conditions in for and while loops and if statements.</td>
</tr>
<tr>
<td>[]</td>
<td>Use to specify dimension of the matrix data-type when it is declared. Also used to access elements in the array at a given position.</td>
</tr>
<tr>
<td>,</td>
<td>Used to separate function arguments and to separate different in a specific row in a given matrix.</td>
</tr>
<tr>
<td>&quot; &quot;</td>
<td>Used to declare a variable of string data type</td>
</tr>
<tr>
<td>//</td>
<td>Inline comment</td>
</tr>
<tr>
<td>/* */</td>
<td>Block comment</td>
</tr>
</tbody>
</table>

### 3.2 Operators and Expressions

#### 3.2.1 Standard Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition, Subtraction, Multiplication, Division, Modulo*</td>
<td>+, -, *, /, %</td>
<td>int a = 5 + 8</td>
</tr>
<tr>
<td>Additive, Subtractive, Multiplicative, Divisive, Modular Assignment*</td>
<td>+=, -=, *=, /=, %=</td>
<td>int a = 4; a += 2;</td>
</tr>
<tr>
<td>Assignment</td>
<td>=</td>
<td>int a = 7</td>
</tr>
<tr>
<td>Equality check</td>
<td>==</td>
<td>7 == 7</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;</td>
<td>6 &gt; 5</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;</td>
<td>5 &lt; 3</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>=&gt;</td>
<td>5 =&gt; 4</td>
</tr>
</tbody>
</table>

// Returns 1
| Less than or equal to | <= | 5 <= 5  
//Returns 1 |
|-----------------------|----|---------|
| Not equal             | != | 5 != 3  
// Returns 1 |
| Logical Not           |   | int a = 1;  
int b = 0;  
!(a && b) //Returns 1 |
| Logical AND           | &&| //with the values from the example above  
a && b //Returns 0 |
| Logical OR            | ||| //with the values from the example above  
a || b //Returns 1 |

*Note that these operations are automatically casted into float when an int and a float are used in an operation.

3.2.2 Matrix Operators

Note that all of these operations are implemented in the standard library, stdlib.neo.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
</table>
| Scalar Multiplication,       | Element-wise multiplication/division or scalar multiplication/division/power | .* , ./ , ^ | mat int C = A.*B;  
mat int C = A./B;  
mat int C = A.*2;  
mat int C = A./2;  
mat int C = A.*2; |
| Scalar Division,             |                                                                             |        |                                                                        |
| Scalar Power                  |                                                                             |        |                                                                        |
| Matrix Multiplication,       | Matrix multiplication/division. Operation is not commutative. If at least one input is  | *, / | mat int C = A*B;  
mat int C = A/B;  
mat int C = A*3;  
mat int C = A/3; |
| Matrix Division              | scalar, then A*B is equivalent to A.*B and is commutative.                   |        |                                                                        |
| Addition, Subtraction        | Addition/subtraction, scalar or element-wise                                | +, -   | mat int C = A+B;  
mat int C = A+2;  
mat int C = A+B;  
mat int C = A-2; |
| Transpose                    | Returns the                                                                | '      | mat int B = A'; |

*Note that these operations are automatically casted into float when an int and a float are used in an operation.*
### 3.2.3 Expressions

Expressions are made of at least one operand and zero or more operators. Innermost expressions are evaluated first and the priority of an expression is determined by parentheses. The direction of evaluation is from left to right.

### 3.3 Literals

#### 3.3.1 String Literals

String literals are a sequence of zero or more letters, spaces, digits, other ASCII characters numbers 32 to 126, excluding the double quote. These strings should be enclosed in double quotes. For example: “Hello, world”.

#### 3.3.2 Integer Literals

Integer literals are one or more number digits, in succession with no whitespace or punctuation character in between them. For example: 42, 666, 2.

#### 3.3.3 Floating-Point Literals

Floating-point literals are decimal numbers. A decimal number is a fraction whose denominator is a power of ten and whose numerator is expressed by figures placed to the right of a decimal point. The integer part is expressed by figures placed to the left of a decimal point. Similarly to integer literals whitespace is not allowed to separate digits or digits and the decimal point `.`. For example: 4.2, 5.0, 222.666

#### 3.3.4 Matrix Literals

Matrix literals can be integer or floating-point numbers. Matrices can only be composed of entirely integers or floating-point numbers.

### 3.4 Data Types

#### 3.4.1 Primitive Data Types
MatriCs provides primitive data types that are common to many high level languages. The full list of primitive data types is given below:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Syntax</th>
<th>Range of Values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>Used to define integer types</td>
<td>int</td>
<td>-2,147,483,648 to 2,147,483,648</td>
<td>int a = 5;</td>
</tr>
<tr>
<td>float</td>
<td>32-bit floating number</td>
<td>float</td>
<td>±1.79769313486231570E+308</td>
<td>float a = 5.0;</td>
</tr>
<tr>
<td>bool</td>
<td>Used to define boolean types (true and false)</td>
<td>bool</td>
<td>True, False</td>
<td>bool flag = true;</td>
</tr>
<tr>
<td>string</td>
<td>Used to define strings of characters</td>
<td>string</td>
<td>Any string of ASCII characters enclosed, excluding double quotes.</td>
<td>string str = “abcd1234”</td>
</tr>
</tbody>
</table>

3.4.2: Non-Primitive Data Types

3.4.2.1 Matrices

Note that MatriCs doesn’t have a data type called arrays and therefore the matrix data type can be used to declare/define an array of any dimensions (i.e. the matrix data type can be used to declare/define one dimensional array).

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix</td>
<td>Defines a matrix</td>
<td>typeofmatrix[row][column] nameofmatrix; nameofmatrix = [elm1,elm2, .. ; elmn,elm(n+1)];</td>
<td>Int[3,3] matr_x; matr_x = [ [1,2,3; 4,5,6] [7,8,9]];</td>
</tr>
</tbody>
</table>

3.4.2.2: Vectors
Vectors are basically 1 dimensional matrices and are declared and defined as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector</td>
<td>Defines a vector</td>
<td>typeofvector[size] nameofvector; nameofvector = [elm1, elm2, …];</td>
<td>int[5] v; v = [1, 2, 3, 4, 5];</td>
</tr>
</tbody>
</table>

### 3.4.2.3 Declaring Matrices

A matrix is declared by first typing the data type to be stored in the matrix, the name of the matrix, and the dimensions of the matrix enclosed by square brackets. An example is given below:

```
int matr_x [2,5]; //returns a 2 by 5 uninitialized matrix holding values of type int
int matr_x = [[0, 0, 0] 0, 0, 0]; //defines a 2 by 3 and initializes its values to zero
```

### 3.4.2.4 Accessing and Setting Array Elements

A matrix element can be accessed by simply typing the name of the matrix and the dimensions of the element desired to be accessed inside square brackets.

```
int elmt;
elmt = matr_x[2,5];
//elmt is equal to 6 - i.e. the element in the second row and fifth column of “matr_x”
```

Array elements can be set either in the declaration stage or later on. Simultaneous declaration and definition is demonstrated in the example below:

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

Array elements can also be set after declaration. An example is given below:

```
matr_x[2][5] = 99
```

### 3.5 Statements
3.5.1 The if Statement

Format:

```java
if(condition){
    action1;
}else if(condition){
    action2;
}else{
    action3
}
```

Description:
The if statement consists of a block of code that only executes if the condition enclosed within the parentheses is true. If not, the block of code is ignored and the program jumps to the next line. Optional additions include else if, which contains another condition to be checked, and else, which executes in the case that none of the conditions above are met. These additions are not required, but if an else statement is used, then it should be the last element of the statement.

3.5.2 The while Loop

Format:

```java
while(condition){
    action;
}
```

Description:
The while loop consists of a block of code that repeatedly executes as long as the condition enclosed within the parentheses is true. If the condition is not true when the code is first encountered, then the program jumps over the block entirely. Furthermore, if the condition is true and remains true indefinitely, then the code gets caught in an infinite loop and the program never continues beyond the block.

3.5.3 The for Loop

Format:

```java
for(variable initialization; condition; increment step){
    action;
}
```

Description:
The for loop is a generalization of the while loop. Within the parentheses there are three distinct parts, separated by semicolons. The first part, variable initialization, runs once
when the for loop is first encountered. The second part is a condition checked on every iteration to determine whether the block of code inside the loop should be executed. If it is executed, the third part then increments (or decrements) and the condition is re-evaluated. The initialization, condition, and increment can be any expressions.

3.6 Functions

3.6.1 Function Definitions

Function definition consists of the type of value returned by the function followed by the name of the function and the set of parameters and types enclosed in parentheses. The scope of the function will be defined by the opening and the closing curly braces, one placed after the function declaration and the other placed after the block of code defining the function is finished. An example of a function is given below:

```c
int determinant(mat int m){
    int det;
    int sign;
    det = 0;
    sign = 1;

    if(rows(m) == 1) { // base case, dimensional array
        return m[0,0];
    }
}
```

3.6.2 Calling Functions

A function can be called by its name followed by its parameters. Note that there is no need to specify the type of the parameter when calling the function. An example is given below:

```c
sum (2, 4) //returns the sum of 2 and 4
```

3.7 Program Structure and Scope

3.7.1 Program Structure

MatriCs program are found on a single source file. The major components of a MatriCs program are matrix declarations, matrix specifications, and function declarations in this specific order.

3.7.2 Scope
Declarations made within a while/for loop, if statement, or any function are available only by reference within this specific block of code. Ones that are made outside of while/for loops, if statement, or any function are available by reference throughout the rest of the code.

3.8 Built-in Functions

3.8.1 The print Function

void print(string str): prints string str on the console, similarly print_int(int a), printb(bool b), and print_float(float a) prints their data types to the console

3.9 Standard Library

The standard library is not a part of MatriCs, but an environment that supports standard MatriCs will provide the function declarations and type and macro definitions of this library. The standard library is invoked by calling “include stdlib.neo”. Currently, the MatriCs standard library contains the functions listed in part 3.2.2.

Appendix A Sample Code

Methods may be used recursively:

```c
int determinant(int m){
    int det;
    int sign;
    int rows;
    int cols;
    int smaller_m[(rows(m))-1][(cols(m))-1];
    det = 0;
    sign = 1;

    if(rows(m) == 1) {   // base case, dimensional array
        return m[0][0];
    }

    // lib functions to get # of rows and columns in m
    // (actually the values should be the same anyway)
    rows = rows(m);
    cols = cols(m);
```
// finds determinant using row-by-row expansion
for(int i = 0; i < rows; i++){

    // keep decomposing the matrix by 1 dimension

    for(int a = 1; a < rows; a++){
        for(int b = 0; b < cols; b++){

            if(b < i){
                smaller_m[a-1][b] = m[a][b];
            }
            elif(b > i){
                smaller_m[a-1][b-1] = m[a][b];
            }
        }
    }

    if (i%2 == 0){    // sign changes based on i
        sign = 1;
    }
    else{
        sign = -1;
    }
    det += sign*m[0][i]*determinant(smaller_m); // rec call for det
}
return det;
}

int transpose(int m) {
    int rows;
    int cols;
    int m_transpose[cols(m)][rows(m)];

    rows = rows(m);
    cols = cols(m);

    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++) {
            m_transpose[j][i] = m[i][j];
        }
    }
}
return m_transpose;
}

int sum_matrix(int m) {
    int sum;
    int cols;
    int rows;
    sum = 0;

    cols = cols(m);
    rows = rows(m);

    for(int i = 0; i < rows; rows++) {
        for(int j = 0; j < cols; cols++) {
            // Will go through all elements
            sum += m;
        }
    }
    return sum;
}

Appendix B Context-Free Grammar

program → ε | program vdecl | program fdecl

fdecl → id ( formals ) { vdecls stmts }

formals → id | formals , id

vdecls → vdecl | vdecls vdecl

vdecl → type id; | type id = expr; | matrix id | matrix id = matexpr;

stmts → ε | stmts stmt

stmt → expr ; | return expr ; | { stmts } | if ( expr ) stmt else_if | if ( expr ) stmt else_if else stmt | while ( expr ) stmt | for ( expr ; expr ; expr ) stmt | foreach( vdecl : id )

expr → lit | id | id ( actuals ) | ( expr ) | expr + expr | expr - expr | expr * expr | expr / expr | expr == expr | expr != expr | expr < expr | expr <= expr | expr > expr | expr >= expr | expr = expr | expr .* expr | expr ./ expr | expr .^ expr
4 Project Plan

4.1 Project Planning Process:

The MatriCs team spent the first two months of the project duration planning the features of the function. After the initial planning stage, the functionalities were defined and depending on these the scanner and the parser were updated. After the initial scanner and parser update stage, codegen and semant were updated accordingly. After the implementation of each feature a detailed set of test cases were written and tested for any bug or edge cases. The MatriCs team worked on each feature one by one; and while one half of the team was working on implementing the feature, the other team was working on developing relevant test code and coming up with relevant built in functions relative to the feature being implemented. After the development of the feature was done and tested, the feature was once again discussed by the team. The team would discuss the applications and the usability of the feature and would update the relevant configuration files accordingly.

4.2 Style Guide Used By The Team

4.2.1 Variable and Function Declarations

- Unless the variable being declared is a constant do not use capital letters to identify variables
- Always declare variables on top of the code block
- Do not start a variable or a function name with a number; always start with a lowercase letter
• Parameter names in the function definition or declaration should be different than the variable names in the function body itself
• One variable per declaration
• Do not declare unused variables

4.2.2 Braces and Spacing

• Braces are used even when optional in for, while and if/else statements
• For each block indentation use 2 spaces to indicate the scope of the statement being made
• Allow for maximum of 100 characters per line and if this limit is passed make sure that you do a line break

4.3 Project Timeline

January-February:
• Initial group meetings begin
• Brainstorming about the kind of language to be implemented
• Drafting the proposal
• Discussions on the proposal and about the TA comments
• Drafting the LRM
• Team meets at least once a week and discusses for at least an hour

March:
• Begin discussing the LRM and its applicability to code with our mentor TA
• Changing the syntax and LRM accordingly
• Download the MicroC compiler and begin studying the configuration files
• Implement Hello World
• Implement Scanner and Parser as well as other primitive data types
• Team meets at least twice a week and mostly does coding

April - May:
• Implement special data types
• Implement built in functions
• Discuss and implement function structure
• Start developing test suits on a regular basis
• Start brainstorming and developing the demo code for the presentation
• Start writing the project report and make the project presentation
4.4 Roles and Responsibilities of Each Team Member

In addition to the roles defined at the beginning of the semester, each team member took active role in the implementation of features. The team tried to work together as much as possible to help each other overcome challenging implementations, debugging and effective testing.

4.4 Software Development Environment

In order to develop the language LLVM 3.0.8 and its library functions were used. Additionally to make best use of version control and to ensure that each member of the team has instant access to the most recent version of the code, Github was used.

4.5 Project Log

The following graphs reflecting our progress throughout the project are extracted from our project repo:
5. Architectural Design

5.1 Block Diagram of the Major Components of Our Translator

![System Architecture Diagram](image)

5.2 Interfaces Between the Components

The following modules and interfaces are used in MatriCs:

- **preprocessor.ml**: Checks if the input file has any references to the standard library.
- **scanner.ml**: Takes in the scanned input file from preprocessor.ml and tokenizes the input file.
• parser.mly: Takes in the tokenized input and checks if its syntax is correct. If the syntax is completely correct, it generates an abstract syntax tree of the input code.

• semant.ml: Takes in the AST from parser and checks if its syntactically correct or not. If the AST passes all of the checks in the semant, semant.mly generates a semantically checked syntax tree (SAST).

• codegen.ml: Takes in the SAST generated and converts its leaves into LLVM IR.

• neo.ml : MatriCs compiler that calls all of the modules and operates. In terms of operation it gives the used a few options by giving three different compilation flags as outlined in the language tutorial section.

• ast.ml: Represents the abstract syntax tree of MatriCs’s language representation.

• sast.ml: Representation of the semantically checked abstract syntax tree of MatriCs’s language representation.

• stdlib.neo : Contains the standard library of a group of functions and it is called by the preprocessor when the program includes this library.

**Note that since the team worked on every feature as a whole, each component was implemented by everyone.**
6 Test Plan

6.1 Example Program and Its LLVM output

Example programs and their outputs can be found in the tests directory of the source code.

6.2 Test Suites

The MicroC test script, namely, testall.sh was used to run all of the tests. This test script takes in a .neo file and the designated output of the input code in the form of a .out file. It compares the actual output of the code with the contents in the .out file and outputs the llvm output, the actual output and the differences between the output and the designated output file in case the tests fail. Note that these files are outputted only when the code passes all of the modules but outputs some result other than the designated result.

6.3 Choosing Tests

In order to cover as much of the code as possible in terms of testing, a test suite was written after each feature implementation. Therefore every feature including functions, variable declarations, definitions, and control flows were tested in a separate test suite. To check the compatibility of the test code for every case some of the test suites were designed to cover more than a single feature.

6.4 Test Automation

While the team didn’t use any automation tool externally, the testing suites were tested using testall.sh, which was a very automated process for the scope of this project.
7 Lessons Learned

Talal - always make the best use of OCaml - although it might seem to be a challenging language its pattern matching functions come in very handy especially when implementing complicated data types

Manos - never leave error checking to the last minute, always consider each fail case after or during the implementation of a language. The error that wasn't checked before might show up later in another language

Duru - Test each feature with more than one test suite and consider each and every case to make sure that everything works

Flo - START EARLY

Nikhil - Always work in a team and consult for help - you don't have much time in this project

```c
TESTS:
int main()
{
    int i;
    bool b;

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

```c
int main() {
```
int[2,2] a;
int[2,2] b;
int[4] a_flat;
int[4] b_flat;
int[4] d_flat;
int[2,2] mat;

int i;
int j;
int k;
int l;
int m;
int z;
int c;
int c1;
int c2;

a = [[1,2], [3,4]];
b = [[5,6], [7,8]];
z = 6;
z++;
z--;
z++;

c = 0;
for(i = 0; i < 2; i = i + 1) {
    for (j = 0; j < 2; j = j + 1) {
        a_flat[c] = a[i, j];
b_flat[c] = b[i, j];
c = c + 1;
    }
}
c1 = 0;
for(k = 0; k < 4; k = k + 1) {
    d_flat[c1] = a_flat[c1] + b_flat[c1];
c1 = c1 + 1;
}
// print_int(d_flat[0]);
// print_int(d_flat[1]);
// print_int(d_flat[2]);
// print_int(d_flat[3]);
c2 = 0;
for(l = 0; l < 2; l = l + 1) {
    for(m = 0; m < 2; m = m + 1) {
        mat[l,m] = d_flat[c2];
        c2 = c2 + 1;
    }
}

print_int(mat[0,0]);
print_int(mat[0,1]);
print_int(mat[1,0]);
print_int(mat[1,1]);
print_int(z);
return 0;

int main() {
    // This is a comment
    /* This is *
    * another *
    * comment */
    return 0;
}

#include <stdlib.neo>

int main() {
    int[2,2,2] a;
    a = [[[1,2], [3,4]],
         [[5,6], [7,8]]];
    print3d(a);
}

int main (){
    int x;
    x = 5;
    return 0;
}

int main() {
float a;
float b;
a = 1.4;
b = 1.0;
b = a + b;
    print_float(b);
    return 0;
}

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

    a = 5.0;
b = 5;
c = a + b;
    print_float(c);
    return 0;
}

int main (){

        string x;
x = "hello\n";

        print(x);
        return 0;
}

int main (){

        string x;
x = "hello\n";

        print(x);
        return 0;
}

int main() {
    int i;
    int j;
// int c;
// int x;
// int y;

int [2, 2] mat;
// c = 0;
for(i = 0; i < 2; i = i + 1) {
    for (j = 0; j < 2; j = j + 1) {
        if(i == j) {
            // print_int(1);
            // mat[i,j] = 1;
            // print(mat[i,j]);
            // x = 1;
            mat[i, j] = 1;
            print_int(mat[i,j]);
            // c = c+1;
        } else {
            // print_int(0);
            // mat[i, j] = 0;
            // print(mat[i,j]);
            // y = 0;
            mat[i, j] = 0;
            print_int(mat[i,j]);
            // c = c + 1;
        }
    }
}
return 0;

#include <stdlib.neo>

float[2,2] inverseMat2by2(int[2,2] a) {
    float[2,2] b;
    float inv_mult;

    inv_mult = 1.0/((a[0,0]*a[1,1]) - (a[0,1]*a[1,0]));

    b[0,0] = inv_mult * a[1,1];
    b[0,1] = (inv_mult * a[0,1]) * -1;
    b[1,0] = (inv_mult * a[1,0]) * -1;
    b[1,1] = inv_mult * a[0,0];
}
int main() {
    int[2,2] a;
    a = [[4,7],[2,6]];

    print2d_f(b);
}

int main() {
    int[2,2,2,2] a;
    int i;
    int j;
    int k;
    int l;

    a = [[[1,2], [3,3]],
         [[7,2], [9,1]]],
         [[[1,2], [3,3]],
         [[7,2], [9,1]]];

    for (i = 0; i < 2; i = i+1) {
        for (j = 0; j < 2; j = j+1) {
            for (k = 0; k < 2; k = k+1) {
                for (l = 0; l < 2; l = l+1) {
                    print_int(a[i,j,k,l]);
                }
            }
        }
    }

    return 0;
}

void print2d(int[2,2] c) {
    int i;
    int j;
    for(i = 0; i < 2; i++) {
        print("[");
        for(j = 0; j < 2; j++) {
            if(j != 1) {
                print_int(c[i,j]);
                print(", ");
            } else {
                print_int(c[i,j]);
            }
        }
    }
}

int main() {
    int[2,2,2] a;
    int i;
    int j;
    int k;
    int l;

    a = [[4,7],[2,6]];

    print2d_f(b);
}
int main() {
    int[2,2] a;
    a = [[1,2],[3,4]];
    print2d(a);
}

void print3d(int[2,2,2] c) {
    int i;
    int j;
    int k;
    for(i = 0; i < 2; i++) {
        print("[");
        for(j = 0; j < 2; j++) {
            print("[");
            for(k = 0; k < 2; k++) {
                if(k != 1) {
                    print_int(c[i,j,k]);
                    print(",");
                } else {
                    print_int(c[i,j,k]);
                }
            }
            if(j != 1) {
                print("]");
            } else {
                print("]");
            }
        }
        if(i != 1) {
            print("]");
        } else {
            print(""");
        }
    }
    print("]
\n");
}

int main() {
    int[2,2,2] a;
    a = [[[1,2], [3,4]],
         [[5,6], [7,8]]];
    print3d(a);
}
int main (){
    bool x;
    x = true;
    printb(x);
    return 0;
}

int main() {
    int[3] x;
    int y;
    int z;
    int a;
    x = [10, 20, 30];
    y = x[0] + 10;
    z = x[1] + 0;
    a = x[2] - 10;
    print_int(y);
    print_int(z);
    print_int(a);
}

int main() {
    int[3] x;
    int y;
    int z;
    int a;
    x = [10, 20, 30];
    y = x[0] / 10;
    z = x[1] * 0;
    a = x[2] - 10;
    print_int(y);
    print_int(z);
    print_int(a);
}
```c
int main() {
    bool x;
    x = false;
    printb(x);
    return 0;
}

int main (){
    int x;
    x = 5;
    print_int(x);
    return 0;
}

int main() {
    string x;
    x = "my string";
    print(x);
    return 0;
}

int main() {
    bool x;
    x = true;
    printb(x);
    return 0;
}

int main() {
    int[4] a;
    int i;
    int c;
    int counter;

    a = [1, 2, 3, 4];
    c = 4;
    counter = 0;
```
print("[");
while(c > 0) {
    if(counter == 0) {
        print_int(a[counter]);
        counter = counter + 1;
        c = c - 1;
    }
    else {
        print(",");
        print_int(a[counter]);
        counter = counter + 1;
        c = c - 1;
        // print(",");
    }
}
print("]");

int main() {
    int[5] v;
    v = [1, 2, 3, 4, 5];

    print_int(v[0]);
    print_int(v[1]);
    print_int(v[2]);
    print_int(v[3]);
    print_int(v[4]);
}

int main() {
    int[4] a;
    a = [1, 2, 3, 4];

    if (a[1] == 2) {
        printb(true);
    } else {
        printb(false);
    }
}

int main() {
    int[4] b;
b = [1, 2, 3, 4];

if (b[1] == 3) {
    printb(true);
} else {
    printb(false);
}

int main() {
    int i;
    i = 1;
    while (i < 5) {
        print_int(i);
        i = i + 1;
    }
    return 0;
}

int main() {
    //
    //
    int[2,2] a;
    int i;
    int j;
    int[4] b;
    int c;
    int[4] b2;
    int c2;
    int[2,2] mat;
    int k;
    int l;
    int m;
    int n;
    int c3;

    a = [[1, 2], [3, 4]];
    c = 0;

    for(i = 0; i < 2; i = i + 1) {
        for(j = 0; j < 2; j = j + 1) {
            //
            // print_int(a[i,j]);
            b[c] = a[i,j];
        }
    }
}
c = c + 1;
}
}
c2 = 0;
for(k = 0; k < 2; k = k + 1) {
  for(l = 0; l < 2; l = l + 1) {
    b2[c2] = b[k + l*2];
    c2 = c2 + 1;
  }
}
// print_int(b2[0]);
// print_int(b2[1]);
// print_int(b2[2]);
// print_int(b2[3]);
c3 = 0;
for(m = 0; m < 2; m = m + 1) {
  for(n = 0; n < 2; n = n + 1) {
    mat[m,n] = b2[c3];
    c3 = c3 + 1;
  }
}
print_int(mat[0,0]);
print_int(mat[0,1]);
print_int(mat[1,0]);
print_int(mat[1,1]);

#include <stdlib.neo>

int main (){
  int[5] x;
  int a;
  int u;
  u = 9;
  x = [1,2,3,4,5];
a = x[2] + 3;
  print_int(a);
  print_int(u);
  return 0;
type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq | Greater | Geq | And | Or

type uop = Neg | Not

type uopment = Increment | Decrement

(* HERE ADD NUM TYPE FOR FLOATS AND INTS*)
type typ = Int | Float | Bool | MyString | Void | Vector of typ * int list

type bind = typ * string

type expr =
    Literal of int | Fliteral of float | BoolLit of bool | MyStringLit of string | Vector_lit of expr list | Id of string | Vdecl of bind | Binop of expr * op * expr
type stmt =
   Block of stmt list
| Expr of expr
| Return of expr
| If of expr * stmt * stmt
| For of expr * expr * expr * stmt
| While of expr * stmt

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

(type program = bind list * func_decl list)

(* Pretty-printing functions *)
let rec string_of_typ = function
  Int -> "int"
| Float -> "float"
| Bool -> "bool"
| MyString -> "string"
| Void -> "void"
| Vector(t, l) -> (string_of_typ t)^"[^"^
     (List.fold_left (fun str n -> str ^", ^"^string_of_int n) "" 1")"]"

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

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

let string_of_uopment = function
Increment -> "++"
| Decrement -> "--"

let rec string_of_expr = function
  | Literal(l) -> string_of_int l
  | Fliteral(f) -> string_of_float f
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | MyStringLit(s) -> s
  | Vector_lit(el) -> "[" (* ^ String.concat " , "
                  (List.map string_of_expr el) ^ "]" *)
  | Id(s) -> s
  | Vdecl(t, s) -> string_of_typ t^ " ^s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " ^ string_of_op o ^ " ^
      string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Incrementer(e, uop) -> string_of_expr e ^
      string_of_uopment uop
  | Assign(v, e) -> string_of_expr v ^ " = ^
      string_of_expr e
  | Call(f, el) ->
      f ^ "(" (* ^ String.concat " , "
                  (List.map
                      string_of_expr el) ^ ")")"
  | Vector_access(v, i) ->
      v ^ "][" ^
      (List.fold_left (fun str n -> str ^"",
                      "^string_of_expr n) " i) ^"])"
  | Dimlist(v) ->
      v ^ ".dims"
  | Noexpr -> ""

(*ADD PATTERN FOR VECTOR_LIT AND VECTOR_ACCESS*)

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

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

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

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

/* Ocamlyacc parser for MicroC */

{%
  open Ast
%
%
	%token SEMI COLON LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA DOT ROWS COLS
	%token PLUS MINUS TIMES DIVIDE MOD ASSIGN NOT DIMS
	%token PLUSPLUS MINMIN
	%token PLUS MINUS TIMES DIVIDE ASSIGN NOT DIMS
	%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
	%token RETURN IF ELSE FOR WHILE INT BOOL FLOAT STRTYPE VOID VECTOR MATRIX
	%token <string> STRING
	%token <int> LITERAL
	%token <float> FLITERAL
	%token <string> ID
	%token EOF

  %nonassoc NOELSE
  %nonassoc ELSE
  %nonassoc PLUSPLUS MINMIN
  %right ASSIGN
  %left OR
  %left AND
  %left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT NEG

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

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

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

formals_opt:
  /* nothing */ { [] }
formal_list:  
  typ ID { ([$1,$2]) }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
  INT { Int }
  | FLOAT { Float }
  | BOOL { Bool }
  | STRTYPE { MyString }
  | VOID { Void }
  | typ LBRACKET dim_list RBRACKET { Vector($1, $3) }

dim_list:
  /* nothing */ []
  | dim { List.rev $1 }

dim:
  LITERAL { [$1] }
  | dim COMMA LITERAL { $3 :: $1 }

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

vdecl:
  typ ID SEMI { ($1, $2) }
stmt_list:
    /* nothing */ { [] }
    | stmt_list stmt { $2 :: $1 }

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

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

expr:
    literal { $1 }
    | expr PLUS expr { Binop($1, Add, $3) }
    | expr MINUS expr { Binop($1, Sub, $3) }
    | expr TIMES expr { Binop($1, Mult, $3) }
    | expr DIVIDE expr { Binop($1, Div, $3) }
    | expr MOD expr { Binop($1, Mod, $3) }
    | expr EQ expr { Binop($1, Equal, $3) }
    | expr NEQ expr { Binop($1, Neq, $3) }
expr LT  expr { Binop($1, Less, $3) }
expr LEQ expr { Binop($1, Leq, $3) }
expr GT  expr { Binop($1, Greater, $3) }
expr GEQ expr { Binop($1, Geq, $3) }
expr AND expr { Binop($1, And, $3) }
expr OR  expr { Binop($1, Or, $3) }
MINUS expr %prec NEG { Unop(Neg, $2) }
NOT expr             { Unop(Not, $2) }
expr PLUSPLUS       { Incremener($1, Increment) }
expr MINMIN         { Incremener($1, Decrement) }
expr ASSIGN expr     { Assign($1, $3) }

ID LPAREN actuals_opt RPAREN { Call($1, $3) }
LPAREN expr RPAREN             { $2 }

ID LBRACKET explst RBRACKET { Vector_access($1, List.rev $3) }
vector                      { Vector_lit($1) }

ID DOT DIMS { Dimlist($1) }

literal:
LITERAL   { Literal($1) }
FLITERAL   { Fliteral($1) }
TRUE       { BoolLit(true) }
FALSE      { BoolLit(false) }
ID         { Id($1) }
STRING     { MyStringLit($1) }

vector:
vector_list:
  vect_element { [$1] }
  | vector_list COMMA vect_element { $3 :: $1 }

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

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

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

CodeGen:

(* 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
module S = Sast

module StringMap = Map.Make(String)

let translate (globals, functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "MatriCs"
  and i32_t = L.i32_type context
  and f64_t = L.double_type context (* prob with floats *)
  (* and i8_t = L.i8_type context *)
  and i1_t = L.i1_type context
  and str_t = L.pointer_type (L.i8_type (context))
  and array_t = L.array_type
  and void_t = L.void_type context in

  let rec ltype_of_typ = function
A.\ Int \to \text{i32}_t \\
| A.\ Float \to \text{f64}_t \\
| A.\ Bool \to \text{i1}_t \\
| A.\ MyString \to \text{str}_t \\
| A.\ Void \to \text{void}_t \\
| A.\ Vector(typ, szl) \to \text{match szl with} \\
| [] \to \text{let ltype_of_typ typ} \\
| [x] \to \text{array}_t (\text{ltype_of_typ (A.\ Vector(typ, [])))} x \\
| hd::tl \to \text{array}_t (\text{ltype_of_typ (A.\ Vector(typ, tl))}) \text{ hd} \\
in

(* Declare each global variable; remember its value in a map *)
\text{let global_vars =}
\text{let global_var m (t, n) =}
\text{let init = L.\ const_int (ltype_of_typ t) 0 in StringMap.\ add n (L.\ define_global n init the_module) m in}
\text{List.\ fold_left global_var StringMap.\ empty globals in}

(* Declare printf(), which the print built-in function will call *)
\text{let printf_t = L.\ var_arg_function_type i32_t [\ str_t |] in}
\text{let printf_func = L.\ declare_function "printf" printf_t the_module in}

(* Define each function (arguments and return type) so we can call it *)
\text{let function_decls =}
\text{let function_decl m fdecl =}
\text{let name = fdecl.\ S.\ sfname and formal_types =}
\text{Array.\ of_list (List.\ map (fun (t, _) -> \text{ltype_of_typ t}) fdecl.\ S.\ sformals) in}
\text{let ftype = L.\ function_type (ltype_of_typ fdecl.\ S.\ styp) in}
formal_types m in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
List.fold_left function_decl StringMap.empty functions m in

(* Fill in the body of the given function *)

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

    let int_format_str = L.build_global_stringptr "%d" "fmt"
    builder in
    let float_format_str = L.build_global_stringptr "%f" "fmt"
    builder in

    (* Construct the function's "locals": formal arguments and locally
       declared variables. Allocate each on the stack, initialize their
       value, if appropriate, and remember their values in the
       "locals" map *)
    let local_vars =
        let add_formal m (t, n) p = L.set_value_name n p;
        let local = L.buildalloca (ltype_of_typ t) n builder in
        ignore (L.build_store p local builder);
        StringMap.add n local m in

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

    let formals = List.fold_left2 add_formal StringMap.empty fdecl.S.sformals
(Array.to_list (L.params the_function)) in
List.fold_left add_local_formals fdecl.S.slocals in

(* Return the value for a variable or formal argument *)
let lookup n = try StringMap.find n local_vars
  with Not_found -> StringMap.find n global_vars
in

(* Construct code for an expression; return its value *)
let rec expr builder ex =
  let build_vect vname indices assign =
    if assign > 0 then
      L.build_gep (lookup vname) (Array.append [|L.const_int i32_t 0|] (Array.of_list (List.map (fun e -> expr builder e) indices))) vname builder
    else
      L.build_load (L.build_gep (lookup vname) (Array.append [|L.const_int i32_t 0|] (Array.of_list (List.map (fun e -> expr builder e) indices))) vname builder)
  in
  let build_vect_ref vname i =
    let rec make_empty_lst i1 l =
      if i1 > 0 then
        make_empty_lst (i1-1) (0::l)
      else
        l
    in
    let eml = make_empty_lst i [] in
    (L.build_in_bounds_gep (lookup vname) (Array.append [|L.const_int i32_t 0|] (Array.of_list (List.map (fun e -> expr builder (S.SLit(e))) eml))) vname builder)
  in
  (match ex with
S.Slit i -> L.const_int i32_t i
| S.SFlit f -> L.const_float f64_t f
| S.SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
| S.SMyStringLit str -> L.build_global_stringptr str "tmp" builder
| S.SNoexpr -> L.const_int i32_t 0
| S.SId(s, _) -> L.build_load (lookup s) s builder
| S.SVector_lit el, ty, diml -> (match diml with
  [x] -> L.const_array (ltype_of_typ ty) (Array.of_list (List.map (expr builder) el))
| hd::tl -> L.const_array (ltype_of_typ (A.Vector(ty, tl))) (Array.of_list (List.map (expr builder) el)))

(* match diml with
  [] -> L.const_array (ltype_of_typ ty)
  | [x] -> L.const_array array_t ((A.Vector(ty, []))) x
  | hd::tl -> *)

(*
  let lst = List.map (expr builder) el in
  let arr = Array.of_list lst in
  let makevect vty = function
    | SVector_lit(el, _, _) -> L.const_array (A.Vector()) arr
    | array_t (ltype_of_typ (A.Vector(ty, []))) x
      in makevect List.hd lst
    | L.const_array (ltype_of_typ ty) arr *
  in makevect List.hd lst
| S.SBinop (e1, op, e2, t1, t2, t) ->
  let el' = expr builder e1
  and e2' = expr builder e2 in
  let b = (match op with
    A.Add when t = A.Int -> L.build_add
    | A.Add when t = A.Float -> L.build_fadd
  end)
  L.build_fadd
  *)
let (e1'', e2'') = match t with
  Float ->
    ((match t1 with
       Int -> L.build_sitofp
       e1' (ltype_of_typ t) "tmp1" builder
       | _ -> e1'),
     (match t2 with
       Int -> L.build_sitofp
       e2' (ltype_of_typ t) "tmp2" builder
       | _ -> e2')))
  in b e1'' e2'' "tmp" builder
| S.SUnop(op, e, t) ->
let e' = expr builder e in

(match op with
  | A.Neg      -> L.build_neg
  | A.Not      -> L.build_not) e' "tmp" builder
  | S.SAssign (s, e, _) -> let e' = expr builder e in
                      (match s with
                       | Sid(var, _) -> ignore (L.build_store e' (lookup var) builder); e'
                       | SVector_access(vname, idx, ty) -> let gep = build_vect vname idx 1 in
                     ignore (L.build_store e' gep builder); e'
                       | _ -> raise(Failure("should not reach here")))
                       | S.SCall ("print_int", [e], _) | S.SCall ("printf", [e], _)
                          -> L.build_call printf_func [| int_format_str ; (expr builder e) |]
                             "printf" builder
                       | S.SCall ("printf", [e], _) -> L.build_call printf_func [| float_format_str ; (expr builder e) |]
                             "printf" builder
                       | S.SCall ("print", [e], _) -> L.build_call printf_func [| (expr builder e) |] "printf" builder
                       | S.SCall (f, act, _) ->
                            let (fdef, fdecl) = StringMap.find f function_decls in
                            let actuals = List.rev (List.map (expr builder) (List.rev act)) in
                            let result = (match fdecl.S.styp with A.Void -> ""
                                          | _ -> f ^ "_result") in
                            L.build_call fdef (Array.of_list actuals) result builder
                      | S.SVector_access (vname, idx, typ) ->
                           build_vect vname idx 0
| S.Dimlist (v, dl) -> L.const_array i32_t (Array.of_list (List.map (fun i -> L.const_int i32_t i) dl))

(*) Invoke "f builder" if the current block doesn't already have a terminal (e.g., a branch). *)

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

(* Build the code for the given statement; return the builder for the statement's successor *)

let rec stmt builder = function
  S.SBlock sl -> List.fold_left stmt builder sl
  | S.SExpr e -> ignore (expr builder e); builder
  | S.SReturn e -> ignore (match fdecl.S.styp with
    A.Void -> L.build_ret_void builder
    | _ -> L.build_ret (expr builder e) builder); builder
  | S.SIf (predicate, then_stmt, else_stmt) ->
    let bool_val = expr builder predicate in
    let merge_bb = L.append_block context "merge" the_function in

    let then_bb = L.append_block context "then" the_function in
    add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
    (L.build_br merge_bb);

    let else_bb = L.append_block context "else" the_function in
add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
   (L.build_br merge_bb);

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

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

  let body_bb = L.append_block context "while_body"
  the_function in
    add_terminal (stmt (L.builder_at_end context body_bb) body)
    (L.build_br pred_bb);

  let pred_builder = L.builder_at_end context pred_bb in
  let bool_val = expr pred_builder predicate in

  let merge_bb = L.append_block context "merge"
  the_function in
    ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
    L.builder_at_end context merge_bb

| S.For (e1, e2, e3, body) -> stmt builder
    ( S.Block [S.Expr e1 ; S.While e2, S.Block [body ; S.Expr e3]] )
    in
(* Build the code for each statement in the function *)

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

(* Add a return if the last block falls off the end *)

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

List.iter build_function_body functions;

the_module

Scanner:
open Ast

(* Expressions *)
type sexpr =
  SLit of int
  | SFlit of float
  | SBoolLit of bool
  | SMyStringLit of string
  | SVector_lit of sexpr list * typ * int list
  | SId of string * typ
  | SBinop of sexpr * op * sexpr * typ * typ
  | SUnop of uop * sexpr * typ
  | SAssign of sexpr * sexpr * typ
  | SCall of string * sexpr list * typ
  | SVector_access of string * sexpr list * typ
  | SReference of string * typ
  | SDimlist of string * int list
  | SNoexpr

(* Statements *)
type sstmt =
  SBlock of stmt list
  | SExpr of sexpr
  | SReturn of sexpr
(* Function Declarations *)

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

(* All method declarations | Main entry method *)
type sprogram = bind list * sfunc_decl list

Semant:

(*Semantic checking for MatriCs compiler*)

open Ast
open Sast

module StringMap = Map.Make(String)

(* Semantic checking of a program. Returns void if successful, throws an exception if something is wrong.

Check each global variable, then check each function *)

let check (globals, functions) =
(* Raise an exception if the given list has a duplicate *)
let report_duplicate exceptf list =
  let rec helper = function
    | n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
    | _ :: t -> helper t
    | [] -> ()
  in helper (List.sort compare list)

(* Raise an exception if a given binding is to a void type *)
let check_not_void exceptf = function
  (Void, n) -> raise (Failure (exceptf n))
  | _ -> ()

(* Raise an exception of the given rvalue type cannot be assigned to
the given lvalue type *)
let check_assign lvaluett rvaluett err =
  (match lvaluett with
    Vector(ltyp, lsize) ->
    (match rvaluett with
      Vector(rtyp, rsize) -> if ltyp == rtyp then ltyp else raise err
      | _ -> if ltyp == rvaluett then ltyp else raise err)
      | _ -> if lvaluett == rvaluett then lvaluett else raise err)
in

(**** Checking Global Variables ****)
list.iter (check_not_void (fun n -> "illegal void global " ^ n))
globals;

report_duplicate (fun n -> "duplicate global " ^ n) (list.map snd globals);

(** Checking Functions **)

if list.mem "int_to_string" (list.map (fun fd -> fd.fname) functions)
then raise (Failure ("function int_to_string may not be defined"))
else ()
if list.mem "print" (list.map (fun fd -> fd.fname) functions)
then raise (Failure ("function print may not be defined")) else ()

report_duplicate (fun n -> "duplicate function " ^ n)
(list.map (fun fd -> fd.fname) functions);

if list.mem "print" (list.map (fun fd -> fd.fname) functions)
then raise (Failure ("function print may not be defined")) else ();

(* Function declaration for a named function *)
let built_in_decls_funcs = [
{ typ = Void; fname = "print_int"; formals = [(Int, "x")];
  locals = []; body = [] }
{ typ = Void; fname = "print_float"; formals = [(Float, "x")];
  locals = []; body = [] }];
{ typ = Void; fname = "printb"; formals = [(Bool, "x")];
locals = []; body = [] }
(*
{ typ = Void; fname = "dimlist"; formals = [(Vector(), "v")];
locals = []; body = [] }
*)
]

let built_in_decls_names = [ "print_int"; "print_float"; "printb"(*;
"dimlist"*)]

in

let built_in_decls = List.fold_right2 (StringMap.add)
built_in_decls_names
built_in_decls_funcs
(StringMap.singleton "print"
{ typ = Void; fname = "print"; formals
= [[(MyString, "x")]];
locals = []; body = [] })

in

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

in
let function_decl s = try StringMap.find s function_decls
    with Not_found -> raise (Failure ("unrecognized function " ^ s))

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

(* -- Top-level function -- *)
let check_function func =

List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^ 
    " in " ^ func.fname)) func.formals;

report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ 
    func.fname) (List.map snd func.formals);

List.iter (check_not_void (fun n -> "illegal void local " ^ n ^ 
    " in " ^ func.fname)) func.locals;

report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ 
    func.fname) (List.map snd func.locals);

(* Type of each variable (global, formal, or local *)
let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
StringMap.empty (globals @ func.formals @ func.locals )

(* Function to print all symbols *)
(* List.iter (fun (t, n) -> print_string ( string_of_typ t ^ " " ^ n)) func.formals; *)

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

let match_binop t1 t2 op e =
  (match op with
    Add | Sub | Mult | Mod | Div ->
      (match t1 with
        Int ->
          (match t2 with
            Int -> Int
            | Float -> Float
            | _ -> raise (Failure ("undefined operation " ^
                                  string_of_op op ^ "on type " ^
                                  string_of_typ t2)))
            | Float -> Float
            | _ -> raise (Failure ("undefined operation " ^
                                string_of_op op ^ "on type " ^
                                string_of_typ t1)))
        | Equal | Neq when t1 = t2 -> Bool
        | Less | Leq | Greater | Geq
          when t1 = Int & t2 = Int -> Bool
| Less | Leq | Greater | Geq |

when t₁ = Float && t₂ = Float -> Bool
| And | Or when t₁ = Bool && t₂ = Bool -> Bool
| _ -> raise (Failure ("illegal binary operator " ^
string_of_typ t₁ ^ " " ^ string_of_op op ^ " " ^
string_of_typ t₂ ^ " in " ^ string_of_expr e)))

(* Return the type of an expression or throw an exception *)

let rec expr = function
    Literal _ -＞ Int
| Fliteral _ -＞ Float
| BoolLit _ -＞ Bool
| Id s -＞ type_of_identifier s
| MyStringLit _ -＞ MyString
| Vector_lit elements -＞
    let hdel = List.hd elements in
    (match hdel with
    Vector_lit(els) -＞
        let rec check_vect_vects = function
            | [] -＞ raise(Failure("empty literal not allowed"))
            | [x] -＞ let Vector(typ, szl) = expr x in Vector(typ,
                ((List.length elements)::szl))
            | fst :: snd :: tail -＞ let Vector(t₁, s₁) = expr fst and
                Vector(t₂, s₂) = expr snd in
                if t₁ == t₂ then
                    let flag = List.iter2 (fun a b -> if a != b then
                        raise (Failure("nested vector literals are of different sizes "))
                    else () ) s₁ s₂
                in check_vect_vects (snd::tail)
            else raise (Failure("nested vector literals are of different types 
                "^string_of_typ t₁^" and "^string_of_typ t₂))
                in check_vect_vects elements
let rec check_vector_types i = function
| [] -> raise( Failure ("empty literal not allowed"))
| [el] -> Vector(expr el, [i+1]) (* HACKY FIX *)
| fst :: snd :: tail ->
  if (expr fst) == (expr snd) then
    check_vector_types (i+1) (snd::tail)
  else raise (Failure ("unmatched element types in vector literal " ^
    string_of_typ (expr fst) ^ ", " ^ string_of_typ (expr snd)))

  in check_vector_types 0 elements)

| Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2
  in
    (match op with
      Add | Sub | Mod | Mult | Div ->
        (match t1 with
          Int ->
            (match t2
              with
                Int ->
                  Int
                | Float -> Float
                  | _ -> raise (Failure ("undefined operation " ^
                      string_of_op op ^ "on type " ^ string_of_typ t2)))
          | Float -> Float
            | _ -> raise (Failure ("undefined operation " ^
                      string_of_op op ^ "on type " ^ string_of_typ t1)))
          | Equal | Neq when t1 = t2 -> Bool
            | Less | Leq | Greater | Geq
              when t1 = Int && t2 = Int
          -> Bool
| Less | Leq | Greater | Geq

when \( t1 = \text{Float} \&\& t2 = \text{Float} \rightarrow \text{Bool} \\
| And | Or when \( t1 = \text{Bool} \&\& t2 = \text{Bool} \rightarrow \text{Bool} \\
| _ \rightarrow \text{raise (Failure ("illegal binary operator "} ^
  
  string_of_typ t1 ^ " " ^ string_of_op op ^ " \\
  " ^
  
  string_of_typ t2 ^ " in " ^ string_of_expr e)))

| Unop(op, e) as ex -> let t = expr e in

(match op with
  
  Neg when \( t = \text{Int} \rightarrow \text{Int} \\
  | \quad \text{Neg when } t \rightarrow \text{Float} \\
  | \quad \text{Not when } t = \text{Bool} \rightarrow \text{Bool} \\
  | \quad _ \rightarrow \text{raise (Failure ("illegal unary operator "} ^
    
    string_of_uop op ^
    
    string_of_typ t ^ " in " ^
    
    string_of_expr ex))
  
  | Incrementer(e, uop) as ex -> let t = expr e in

  (match t with
    
    Int \rightarrow \text{Int} \\
  | \quad _ \rightarrow \text{raise (Failure("illegal type cannot be incremented "} ^
    
    string_of_typ t)))

| Noexpr \rightarrow \text{Void}

| Assign(var_ex, e) as ex ->

  (match var_ex with
    
    Id var -> let lt = type_of_identifier var
    
      and rt = expr e in

      check_assign lt rt (Failure ("illegal assignment "} ^
        
        string_of_typ lt ^
        
        " = " ^ string_of_typ rt ^ " in " ^
        
        string_of_expr ex))

| Vector_access(vnm, idx) \rightarrow \text{let } lt = expr var_ex
and rt = expr e in

check_assign lt rt (Failure ("illegal vector assignment " ^
string_of_typ lt ^
" = " ^ string_of_typ rt ^ " in " ^
string_of_expr ex)))
| _ -> raise (Failure ("illegal assignment " ^
string_of_expr var_ex ^ " = " ^ string_of_expr e ^
" in " ^ string_of_expr ex)))
| Call(fname, actuals) as call -> let fd = function_decl fname
in
  if List.length actuals != List.length fd.formals then
    raise (Failure ("expecting " ^ string_of_int (List.length fd.formals) ^ " arguments in " ^
string_of_expr call))
  else
    List.iter2 (fun (ft, _) e -> let et = expr e in
      ignore (check_assign ft et
         (Failure ("illegal actual argument found " ^
string_of_typ et ^
" expected " ^ string_of_typ ft ^ " in " ^
string_of_expr e)))))
    fd.formals actuals;
  fd.typ
| Vector_access(nm, ilst) -> let Vector(typ, sz) =
type_of_identifier nm
  (let rec check_i = function
    [idx] -> let idxtyp = expr idx in
      if idxtyp != Int then raise (Failure ("array " ^ nm
      ^ " index not an integer"))
    else typ
    | hd::tl -> let idxtyp = expr hd in
      if idxtyp != Int then raise (Failure ("array " ^ nm
      ^ " index not an integer"))
    else check_i tl in check_i ilst)
| Dimlist s -> let nm = type_of_identifier s in
(match nm with
    Vector(typ, szl) -> Vector(Int, [(List.length szl)])
  | _ -> raise(Failure("dims cannot be called on non-vector
type "^s)))

(* TODO: TRY IMPLEMENTING INDEX OUT OF BOUNDS
else let Vector(typ, sz) = type_of_identifier nm in
  if (sz - 1) < idx then raise (Failure ("array " ^ nm
      ^ " index out of bounds")) *)

let check_bool_expr e = if expr e != Bool
  then raise (Failure ("expected Boolean expression in " ^
    string_of_expr e))
  else ()

(* Verify a statement or throw an exception *)
let rec stmt = function
    Block sl -> let rec check_block = function
                [Return _ as s] -> stmt s
                | Return _ ::: _ -> raise (Failure "nothing may follow a
                  return")
                | Block sl :: ss -> check_block (sl @ ss)
                | s :: ss -> stmt s ; check_block ss
                | [] -> ()
            in check_block sl
  | Expr e -> ignore (expr e)
  | Return e -> let t = expr e in if t = func(typ then () else
                             raise (Failure ("return gives " ^ string_of_typ t ^ " expected
                             " ^
                             string_of_typ func.typ ^ " in " ^
                             string_of_expr e))

```plaintext
| If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
| For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
| ignore (expr e3); stmt st
| While(p, s) -> check_bool_expr p; stmt s
in
stmt (Block func.body);
in

List.iter check_function functions;

let ast_to_sast func =
  let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
    StringMap.empty (globals @ func.formals @ func.locals)
  in

  (* Function to print all symbols *)
  (* List.iter (fun (t, n) -> print_string (string_of_typ t ^ " ") ^
  n)) func.formals; *)

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

let sast_to_typ = function
  SId(_, t) -> t
| SLit(_) -> Int
| SFlit(_) -> Float
| SBoolLit(_) -> Bool
```
| SMMyStringLit(_, ) -> MyString
| SBinop(_, op, _, _, _, t) -> t
| SAssign(_, _, t) -> t
| SCall(_, _, t) -> t
| SUnop(_, _, t) -> t
| SVVector_access(_, _, t) -> t
| _ -> raise (Failure ("vector type not supported"))

(* Return the type of an expression or throw an exception *)

let rec sexpr = function
    Literal i -> SLit(i)
| Fliteral f -> SFlit(f)
| BoolLit b -> SBoolLit(b)
| Id s -> SId(s, type_of_identifier s)
| MyStringLit st -> SMMyStringLit(st)

| Vector_lit elements ->
    let rec check_vect dimlist elm =
        let el = List.hd elm in
        match el with
            Vector_lit(ellist) -> check_vect (List.length elm :: dimlist) ellist
            | x -> ((List.length elm :: dimlist), sexpr x)
        in let dml, xtyp = check_vect [] elements
            and selements = List.map sexpr elements in
        (* SVVector_lit(selements, (sast_to_typ xtyp), dml) *)
        let head = List.hd selements in
        (match head with
            _ -> SVVector_lit(selements, (sast_to_typ xtyp), dml)
            | SVVector_lit(_, childdtyp, childdml) -> SVVector_lit(selements, Vector(childdtyp, childdml), dml))
| Binop(e1, op, e2) as e -> let t1 = sexpr e1 and t2 = sexpr e2 in
| let typ1 = sast_to_typ t1 and typ2 = sast_to_typ t2 in
| let typ_of_bop =
| (match op with
| Add | Sub | Mult | Mod | Div ->
| (match typ1 with
| Int ->
| (match typ2 with
| Int
| -> Int
| Float -> Float
| | _
| -> raise (Failure ("undefined operation " ^
| string_of_op op ^ "on type
| " ^ string_of_typ typ2)))
| | Float -> Float
| | _ -> raise
| (Failure ("undefined operation " ^
| string_of_op op ^ "on type
| " ^ string_of_typ typ1)))
| | Equal | Neq when typ1 = typ2 -> Bool
| | Less | Leq | Greater | Geq
| | when typ1 = Int && typ2 =
| Int -> Bool
| | Less | Leq | Greater | Geq
| | when typ1 = Float && typ2 =
| Float -> Bool
| | And | Or when typ1 = Bool && typ2 =
| Bool -> Bool
| | _ -> raise (Failure ("illegal binary operator " ^
| string_of_typ typ1 ^ " " ^ string_of_op op ^
string_of_typ typ2)) in
SBinop(t1, op, t2, typ1, typ2, typ_of_bop)
| Unop(op, e) -> let t = sexpr e in
SUnop(op, t, sast_to_typ(t))
| Incrementer(e, uop) ->
(match uop with
  Increment -> sexpr (Assign(e, Binop(e, Add, Literal(1))))
  | Decrement -> sexpr (Assign(e, Binop(e, Sub, Literal(1))))
)(* let t = sexpr e in
  let type_of_e = sast_to_typ t in
  let type_of_exp = (match type_of_e with
    Int -> Int
  | _ -> raise (Failure ("Illegal type " ^ string_of_typ type_of_e ^ " cannot be incremented")))
) in
SIncremeneter(e, uop, type_of_exp)
*)
| Noexpr -> SNoexpr

| Assign(var_ex, e) as ex ->
(match var_ex with
  Id var -> let lt = sexpr var_ex
            and rt = sexpr e
            and ty = type_of_identifier var in
            SAssign(lt, rt, ty)
  | Vector_access(vnm, idx) ->
    let lt = sexpr var_ex
    and rt = sexpr e
    and vty = type_of_identifier vnm in
    (match vty with
Vector(ty, _) -> SAssign(lt, rt, ty)
| _  -> raise(Failure("illegal assignment " ^
  string_of_expr var_ex ^ " = " ^ string_of_expr e ^
  " in " ^ string_of_expr ex)) (* should not reach here after
  check *)
| _  -> raise(Failure("illegal assignment " ^
  string_of_expr var_ex ^ " = " ^ string_of_expr e ^
  " in " ^ string_of_expr ex))) (* should not reach here
  after check *)
  (* let lt = sexpr var_ex in
  let rt = sexpr e in
  let ty = type_of_identifier s
  in SAssign(lt, rt, ty) *)
| Call(fname, actuals) -> let fd = function_decl fname in
  let sactuals = List.map sexpr actuals in
  SCall(fname, sactuals, fd.typ) (* sast_to_typ(sexpr (List.hd
  sactuals)) *)
| Vector_access(vname, idx) -> let Vector(ty, _) =
  type_of_identifier vname in
  let sidx = List.map sexpr idx in
  SVector_access(vname, sidx, ty)
| Dimlist s -> let vect = type_of_identifier s in
  (match vect with
  Vector(t,dl) -> SDimlist(s, dl)
  | _  -> raise(Failure("dims cannot be called on non-vector
  type "^s))
  )

in

let rec sstmt = function
let convert_fdecl_to_sfdecl fdecl =
{
  sfname = fdecl.fname;
  styp = fdecl.typ;
  sformals = fdecl.formals;
  slocals = fdecl.locals;
  sbody = ast_to_sast fdecl;
}

let sfdecls = list.map convert_fdecl_to_sfdecl functions

(globals, sfdecls)

Std Lib:

//Transpose 2x2
int transpose2(int[2, 2] mat) {
  int i;
  int j;
int k;
int l;
int m;
int n;

int [4] vec1;
int [4] vec2;

int count1;
int count2;
int count3;

int [2, 2] tr;

count1 = 0;
for(i = 0; i < 2; i = i + 1) {
    for(j = 0; j < 2; j = j + 1) {
        vec1[count1] = mat[i, j];
        count1 = count1 + 1;
    }
}

count2 = 0;
for(k = 0; k < 2; k = k + 1) {
    for(l = 0; l < 2; l = l + 1) {
        vec2[count2] = vec1[k + l*2];
        count2 = count2 + 1;
    }
}
count3 = 0;
for(m = 0; m < 2; m = m + 1) {
    for(n = 0; n < 2; n = n + 1) {
        tr[m, n] = vec2[count3];
        count3 = count3 + 1;
    }
}
return 0;

//Transpose 3x3
int transpose3(int [3, 3] mat) {
    int i;
    int j;
    int k;
    int l;
    int m;
    int n;

    int [9] vec1;
    int [9] vec2;

    int count1;
    int count2;
    int count3;

    int [3, 3] tr;
count1 = 0;
for(i = 0; i < 3; i = i + 1) {
    for(j = 0; j < 3; j = j + 1) {
        vec1[count1] = mat[i, j];
        count1 = count1 + 1;
    }
}

count2 = 0;
for(k = 0; k < 3; k = k + 1) {
    for(l = 0; l < 3; l = l + 1) {
        vec2[count2] = vec1[k + l*3];
        count2 = count2 + 1;
    }
}

count3 = 0;
for(m = 0; m < 3; m = m + 1) {
    for(n = 0; n < 3; n = n + 1) {
        tr[m, n] = vec2[count3];
        count3 = count3 + 1;
    }
}
return 0;

// Transpose 4x4
int transpose4(int [4, 4] mat) {
    int i;
int j;
int k;
int l;
int m;
int n;

int [16] vec1;
int [16] vec2;

int count1;
int count2;
int count3;

int [4, 4] tr;

count1 = 0;
for(i = 0; i < 4; i = i + 1) {
    for(j = 0; j < 4; j = j + 1) {
        vec1[count1] = mat[i, j];
        count1 = count1 + 1;
    }
}

count2 = 0;
for(k = 0; k < 4; k = k + 1) {
    for(l = 0; l < 4; l = l + 1) {
        vec2[count2] = vec1[k + l*4];
        count2 = count2 + 1;
    }
}
count3 = 0;
for (m = 0; m < 4; m = m + 1) {
    for (n = 0; n < 4; n = n + 1) {
        tr[m, n] = vec2[count3];
        count3 = count3 + 1;
    }
}
return 0;

// Transpose 5x5
int transpose5(int [5, 5] mat) {
    int i;
    int j;
    int k;
    int l;
    int m;
    int n;
    int [25] vec1;
    int [25] vec2;
    int count1;
    int count2;
    int count3;
    int [5, 5] tr;
count1 = 0;
for(i = 0; i < 5; i = i + 1) {
    for(j = 0; j < 5; j = j + 1) {
        vec1[count1] = mat[i, j];
        count1 = count1 + 1;
    }
}

count2 = 0;
for(k = 0; k < 5; k = k + 1) {
    for(l = 0; l < 5; l = l + 1) {
        vec2[count2] = vec1[k + l*5];
        count2 = count2 + 1;
    }
}

count3 = 0;
for(m = 0; m < 5; m = m + 1) {
    for(n = 0; n < 5; n = n + 1) {
        tr[m, n] = vec2[count3];
        count3 = count3 + 1;
    }
}
return 0;
}

// Add 2-D Matrices
int add2(int [2, 2] mat1, int [2, 2] mat2) {
int[4] a_flat;
int[4] b_flat;
int[4] d_flat;
int[2, 2] add_mat;

int i;
int j;
int k;
int l;
int m;

int count1;
int count2;
int count3;

count1 = 0;
for(i = 0; i < 2; i = i + 1) {
    for(j = 0; j < 2; j = j + 1) {
        a_flat[count1] = mat1[i, j];
b_flat[count1] = mat2[i, j];
count1 = count1 + 1;
    }
}

count2 = 0;
for(k = 0; k < 4; k = k + 1) {
    d_flat[count2] = a_flat[count2] + b_flat[count2];
count2 = count2 + 1;
}
count3 = 0;
for(l = 0; l < 2; l = l + 1) {
    for(m = 0; m < 2; m = m + 1) {
        add_mat[l, m] = d_flat[count3];
        count3 = count3 + 1;
    }
}

return 0;

int add3(int [3, 3] mat1, int [3, 3] mat2) {
    int[9] a_flat;
    int[9] b_flat;
    int[9] d_flat;
    int[3, 3] add_mat;

    int i;
    int j;
    int k;
    int l;
    int m;

    int count1;
    int count2;
    int count3;
int add4(int [4, 4] mat1, int [4, 4] mat2) {
    int[16] a_flat;
    int[16] b_flat;
    int count1 = 0;
    for(i = 0; i < 3; i = i + 1) {
        for(j = 0; j < 3; j = j + 1) {
            a_flat[count1] = mat1[i, j];
            b_flat[count1] = mat2[i, j];
            count1 = count1 + 1;
        }
    }
    count2 = 0;
    for(k = 0; k < 9; k = k + 1) {
        d_flat[count2] = a_flat[count2] + b_flat[count2];
        count2 = count2 + 1;
    }
    count3 = 0;
    for(l = 0; l < 3; l = l + 1) {
        for(m = 0; m < 3; m = m + 1) {
            add_mat[l, m] = d_flat[count3];
            count3 = count3 + 1;
        }
    }
    return 0;
}
int[16] d_flat;
int[4, 4] add_mat;

int i;
int j;
int k;
int l;
int m;

int count1;
int count2;
int count3;

count1 = 0;
for (i = 0; i < 4; i = i + 1) {
    for (j = 0; j < 4; j = j + 1) {
        a_flat[count1] = mat1[i, j];
        b_flat[count1] = mat2[i, j];
        count1 = count1 + 1;
    }
}

count2 = 0;
for (k = 0; k < 16; k = k + 1) {
    d_flat[count2] = a_flat[count2] + b_flat[count2];
    count2 = count2 + 1;
}

count3 = 0;
for(l = 0; l < 4; l = l + 1) {
    for(m = 0; m < 4; m = m + 1) {
        add_mat[l, m] = d_flat[count3];
        count3 = count3 + 1;
    }
}

return 0;
}

int add5(int [5, 5] mat1, int [5, 5] mat2) {
    int[25] a_flat;
    int[25] b_flat;
    int[25] d_flat;
    int[5, 5] add_mat;
    int i;
    int j;
    int k;
    int l;
    int m;
    int count1;
    int count2;
    int count3;
    count1 = 0;
    for(i = 0; i < 5; i = i + 1) {
        for(j = 0; j < 5; j = j + 1) {
a_flat[count1] = mat1[i, j];
b_flat[count1] = mat2[i, j];
count1 = count1 + 1;
}
}

count2 = 0;
for(k = 0; k < 25; k = k + 1) {
    d_flat[count2] = a_flat[count2] + b_flat[count2];
count2 = count2 + 1;
}

count3 = 0;
for(l = 0; l < 5; l = l + 1) {
    for(m = 0; m < 5; m = m + 1) {
        add_mat[l, m] = d_flat[count3];
count3 = count3 + 1;
    }
}

return 0;
}

// Subtract 2-D Matrices
int subtract2(int [2, 2] mat1, int [2, 2] mat2) {
    int[4] a_flat;
    int[4] b_flat;
    int[4] d_flat;
    int[2, 2] sub_mat;
int i;
int j;
int k;
int l;
int m;

int count1;
int count2;
int count3;

count1 = 0;
for(i = 0; i < 2; i = i + 1) {
    for(j = 0; j < 2; j = j + 1) {
        a_flat[count1] = mat1[i, j];
        b_flat[count1] = mat2[i, j];
        count1 = count1 + 1;
    }
}

count2 = 0;
for(k = 0; k < 4; k = k + 1) {
    d_flat[count2] = a_flat[count2] - b_flat[count2];
    count2 = count2 + 1;
}

count3 = 0;
for(l = 0; l < 2; l = l + 1) {
    for(m = 0; m < 2; m = m + 1) {

int subtract3(int [3, 3] mat1, int [3, 3] mat2) {
    int[9] a_flat;
    int[9] b_flat;
    int[9] d_flat;
    int[3, 3] sub_mat;

    int i;
    int j;
    int k;
    int l;
    int m;

    int count1;
    int count2;
    int count3;

    count1 = 0;
    for(i = 0; i < 3; i = i + 1) {
        for(j = 0; j < 3; j = j + 1) {
            a_flat[count1] = mat1[i, j];
            b_flat[count1] = mat2[i, j];
            count1 = count1 + 1;
        }
    }

    sub_mat[0, 0] = a_flat[count1] - b_flat[count1];
    sub_mat[0, 1] = a_flat[count1] - b_flat[count1];
    sub_mat[0, 2] = a_flat[count1] - b_flat[count1];
    sub_mat[1, 0] = a_flat[count1] - b_flat[count1];
    sub_mat[1, 1] = a_flat[count1] - b_flat[count1];
    sub_mat[1, 2] = a_flat[count1] - b_flat[count1];
    sub_mat[2, 0] = a_flat[count1] - b_flat[count1];
    sub_mat[2, 1] = a_flat[count1] - b_flat[count1];
    sub_mat[2, 2] = a_flat[count1] - b_flat[count1];

    return 0;
}
count1 = count1 + 1;

}

}

count2 = 0;
for(k = 0; k < 9; k = k + 1) {
    d_flat[count2] = a_flat[count2] - b_flat[count2];
    count2 = count2 + 1;
}

}

count3 = 0;
for(l = 0; l < 3; l = l + 1) {
    for(m = 0; m < 3; m = m + 1) {
        sub_mat[l, m] = d_flat[count3];
        count3 = count3 + 1;
    }
}

}

return 0;

}

int subtract4(int [4, 4] mat1, int [4, 4] mat2) {
    int[16] a_flat;
    int[16] b_flat;
    int[16] d_flat;
    int[4, 4] sub_mat;

    int i;
    int j;
int k;
int l;
int m;

int count1;
int count2;
int count3;

count1 = 0;
for (i = 0; i < 4; i = i + 1) {
    for (j = 0; j < 4; j = j + 1) {
        a_flat[count1] = mat1[i, j];
        b_flat[count1] = mat2[i, j];
        count1 = count1 + 1;
    }
}

count2 = 0;
for (k = 0; k < 16; k = k + 1) {
    d_flat[count2] = a_flat[count2] - b_flat[count2];
    count2 = count2 + 1;
}

count3 = 0;
for (l = 0; l < 4; l = l + 1) {
    for (m = 0; m < 4; m = m + 1) {
        sub_mat[l, m] = d_flat[count3];
        count3 = count3 + 1;
    }
}
int subtract5(int [5, 5] mat1, int [5, 5] mat2) {
    int[25] a_flat;
    int[25] b_flat;
    int[25] d_flat;
    int[5, 5] sub_mat;

    int i;
    int j;
    int k;
    int l;
    int m;

    int count1;
    int count2;
    int count3;

    count1 = 0;
    for(i = 0; i < 5; i = i + 1) {
        for(j = 0; j < 5; j = j + 1) {
            a_flat[count1] = mat1[i, j];
            b_flat[count1] = mat2[i, j];
            count1 = count1 + 1;
        }
    }
}
count2 = 0;
for(k = 0; k < 25; k = k + 1) {
    d_flat[count2] = a_flat[count2] - b_flat[count2];
    count2 = count2 + 1;
}

count3 = 0;
for(l = 0; l < 5; l = l + 1) {
    for(m = 0; m < 5; m = m + 1) {
        sub_mat[l, m] = d_flat[count3];
        count3 = count3 + 1;
    }
}

return 0;

//Identity 2x2
int id2() {
    int i;
    int j;

    int [2, 2] mat;
    print("[");
    for(i = 0; i < 2; i = i + 1) {
        for(j = 0; j < 2; j = j + 1) {
            if(i == j) {
                mat[i, j] = 1;
            } else {
                mat[i, j] = 0;
            }
        }
    }
    print("]");
    return 0;
}
// Identity 3x3
int id3() {
    int i;
    int j;
    int [3, 3] mat;
    print("[");
    for (i = 0; i < 3; i = i + 1) {
        for (j = 0; j < 3; j = j + 1) {
            if (i == j) {
                mat[i, j] = 1;
                print_int(mat[i, j]);
            } else {
                mat[i, j] = 0;
                print_int(mat[i, j]);
            }
        }
    }
    print("]");
    return 0;
}
// Identity 4x4
int id4() {
    int i;
    int j;

    int [4, 4] mat;
    print("[");
    for(i = 0; i < 4; i = i + 1) {
        for (j = 0; j < 4; j = j + 1) {
            if(i == j) {
                mat[i, j] = 1;
                print_int(mat[i,j]);
            }
            else {
                mat[i, j] = 0;
                print_int(mat[i,j]);
            }
        }
    }
    print("]");
    return 0;
}

// Identity 5x5
int id5() {

}
int i;
int j;
int [5, 5] mat;
print("[");
for(i = 0; i < 5; i = i + 1) {
    for (j = 0; j < 5; j = j + 1) {
        if(i == j) {
            mat[i, j] = 1;
            print_int(mat[i,j]);
        } else {
            mat[i, j] = 0;
            print_int(mat[i,j]);
        }
    }
}
print("]");
return 0;
}

// Identity 6x6
int id6() {
    int i;
    int j;
    int [6, 6] mat;
    print("[");
    for(i = 0; i < 6; i = i + 1) {
        for (j = 0; j < 6; j = j + 1) {

if(i == j) {
    mat[i, j] = 1;
    print_int(mat[i,j]);
}
else {
    mat[i, j] = 0;
    print_int(mat[i,j]);
}
}
}
print("]");
return 0;
}

// Identity 7x7
int id7() {
    int i;
    int j;

    int [7, 7] mat;
    print("[");
    for(i = 0; i < 7; i = i + 1) {
        for (j = 0; j < 7; j = j + 1) {
            if(i == j) {
                mat[i, j] = 1;
                print_int(mat[i,j]);
            }
        }
    }
    else {
        mat[i, j] = 0;
        print_int(mat[i,j]);
    }
// Identity 8x8
int id8() {
  int i;
  int j;

  int [8, 8] mat;
  print("[");
  for(i = 0; i < 8; i = i + 1) {
    for (j = 0; j < 8; j = j + 1) {
      if(i == j) {
        mat[i, j] = 1;
        print_int(mat[i,j]);
      }
      else {
        mat[i, j] = 0;
        print_int(mat[i,j]);
      }
    }
  }
  print("]");
  return 0;
}

}
// Identity 9x9
int id9() {
    int i;
    int j;

    int [9, 9] mat;
    print("[");
    for(i = 0; i < 9; i = i + 1) {
        for (j = 0; j < 9; j = j + 1) {
            if(i == j) {
                mat[i, j] = 1;
                print_int(mat[i,j]);
            } else {
                mat[i, j] = 0;
                print_int(mat[i,j]);
            }
        }
    }
    print("]");
    return 0;
}

// Identity 10x10
int id10() {
    int i;
    int j;

    int [10, 10] mat;
    print("[");
    for(i = 0; i < 10; i = i + 1) {
        for (j = 0; j < 10; j = j + 1) {
            if(i == j) {
                mat[i, j] = 1;
                print_int(mat[i,j]);
            } else {
                mat[i, j] = 0;
                print_int(mat[i,j]);
            }
        }
    }
    print("]");
    return 0;
}
for(i = 0; i < 10; i = i + 1) {
    for (j = 0; j < 10; j = j + 1) {
        if(i == j) {
            mat[i, j] = 1;
            print_int(mat[i,j]);
        }
        else {
            mat[i, j] = 0;
            print_int(mat[i,j]);
            print_int(mat[i,j]);
        }
    }
}
print("\]")
return 0;
}

// Print Vector Size 2
void print_vec2(int[2] a) {
    int i;
    int c;
    int counter;

    c = 2;
    counter = 0;

    print("["");
    while(c > 0) {
        if(counter == 0) {
            print_int(a[counter]);
            print_int(a[counter]);
        }
        counter = counter + 1;
    }
}
void print_vec3(int[3] a) {
    int i;
    int c;
    int counter;
    
    c = 3;
    counter = 0;
    
    print("[");
    while(c > 0) {
        if(counter == 0) {
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        } else {
            print(",");
            print_int(a[counter]);
        }
    }
    print("]");
}
```c
    counter = counter + 1;
    c = c - 1;

    } }
    print("]"); }

void print_vec4(int[4] a) {
    int i;
    int c;
    int counter;

    c = 4;
    counter = 0;

    print("[");
    while(c > 0) {
        if(counter == 0) {
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
        else {
            print(",");
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
    }
    print("]");
```
void print_vec5(int[5] a) {
    int i;
    int c;
    int counter;

    c = 5;
    counter = 0;

    print("[");
    while(c > 0) {
        if(counter == 0) {
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
        else {
            print(",");
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
    }
    print("]");
}

void print_vec6(int[6] a) {
    int i;
    int c;

    c = 6;
    counter = 0;

    print("[");
    while(c > 0) {
        if(counter == 0) {
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
        else {
            print(",");
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
    }
    print("]");
}
int counter;

    c = 6;
    counter = 0;

    print("[");
    while(c > 0) {
        if(counter == 0) {
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
        else {
            print(",");
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
    }
    print("]");
}

void print_vec7(int[7] a) {
    int i;
    int c;
    int counter;

    c = 7;
    counter = 0;

void print_vecf2(float[2] a) {
    int i;
    int c;
    int counter;

    c = 2;
    counter = 0;

    print("[");
    while(c > 0) {
        if(counter == 0) {
            print_float(a[counter]);
            counter = counter + 1;
            c = c - 1;
        } else {
            print(", ");
            print_int(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
    }
    print("]");
}

c = c - 1;
}
else {
    print(",");
    print_float(a[counter]);
    counter = counter + 1;
    c = c - 1;
}
}
print("]");

void print_vecf3(float[3] a) {
    int i;
    int c;
    int counter;

    c = 3;
    counter = 0;

    print("[");
    while(c > 0) {
        if(counter == 0) {
            print_float(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
        else {
            print(",");
            print_float(a[counter]);
        }
counter = counter + 1;
c = c - 1;
}
}
print("]");
}

void print_vecf4(float[4] a) {
    int i;
    int c;
    int counter;

c = 4;
counter = 0;

print("[");
while(c > 0) {
    if(counter == 0) {
        print_float(a[counter]);
counter = counter + 1;
c = c - 1;
    }
    else {
        print(",");
        print_float(a[counter]);
counter = counter + 1;
c = c - 1;
    }
}
print("]");
void print_vecf5(float[5] a) {
    int i;
    int c;
    int counter;

    c = 5;
    counter = 0;

    print("[");
    while(c > 0) {
        if(counter == 0) {
            print_float(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
        else {
            print(",");
            print_float(a[counter]);
            counter = counter + 1;
            c = c - 1;
        }
    }
    print("]");
}

void print_vecf6(float[6] a) {
    int i;
    int c;
int counter;

c = 6;
counter = 0;

print("[");
while(c > 0) {
    if(counter == 0) {
        print_float(a[counter]);
counter = counter + 1;
c = c - 1;
    }
    else {
        print(",");
        print_float(a[counter]);
counter = counter + 1;
c = c - 1;
    }
}
print("]");
}

// Print Vector Size 7 float
void print_vecf7(float[7] a) {
    int i;
    int c;
    int counter;

c = 7;
counter = 0;
print("[");
while(c > 0) {
    if(counter == 0) {
        print_float(a[counter]);
        counter = counter + 1;
        c = c - 1;
    } else {
        print(",");
        print_float(a[counter]);
        counter = counter + 1;
        c = c - 1;
    }
}
print("]");
}

void print3d(int[2,2,2] c) {
    int i;
    int j;
    int k;
    for(i = 0; i < 2; i++) {
        print("[");
        for(j = 0; j < 2; j++) {
            print("[");
            for(k = 0; k < 2; k++) {
                if(k != 1) {
                    print_int(c[i,j,k]);
                    print(",");
                } else {
                    print_int(c[i,j,k]);
                    print(" ");
                } else {
                    print_int(c[i,j,k]);
                    print(" ");
                }
            }
        }
    }
}

void print3d(int[2,2,2] c) {
    int i;
    int j;
    int k;
    for(i = 0; i < 2; i++) {
        print("[");
        for(j = 0; j < 2; j++) {
            print("[");
            for(k = 0; k < 2; k++) {
                if(k != 1) {
                    print_int(c[i,j,k]);
                    print(",");
                } else {
                    print_int(c[i,j,k]);
                    print(" ");
                }
            }
        }
    }
    print("]");
}

void print3d(int[2,2,2] c) {
    int i;
    int j;
    int k;
    for(i = 0; i < 2; i++) {
        print("[");
        for(j = 0; j < 2; j++) {
            print("[");
            for(k = 0; k < 2; k++) {
                if(k != 1) {
                    print_int(c[i,j,k]);
                    print(",");
                } else {
                    print_int(c[i,j,k]);
                    print(" ");
                }
            }
        }
    }
    print("]");
}

void print3d(int[2,2,2] c) {
    int i;
    int j;
    int k;
    for(i = 0; i < 2; i++) {
        print("[");
        for(j = 0; j < 2; j++) {
            print("[");
            for(k = 0; k < 2; k++) {
                if(k != 1) {
                    print_int(c[i,j,k]);
                    print(",");
                } else {
                    print_int(c[i,j,k]);
                    print(" ");
                }
            }
        }
    }
    print("]");
}
    print_int(c[i,j,k]);
    }
    }
    if(j != 1) {
        print("], ");
    } else {
        print("]");
    }
    }
    print("]\\n");
}