DARN: *A Matrix Manipulation Language*

Daisy Chaussee (dac2183)
Anthony Kim (ak3703)
Rafael Takasu (rgt2108)
Ignacio (**N**acho) Torras (it2216)

December 20, 2016

Contents

1 Introduction to the Language

1.1 Motivation

Matrices are immensely powerful tools with numerous applications, within mathematics and beyond. For example, taking advantage of a matrix's compact representation of a set of numbers, game theory and economics use the payoff matrix to encode the payoff for two players, depending on their choices. Text mining and thesaurus compilation make use of document-term matrices to track frequencies of words. Computer graphics uses matrices to represent objects and their transformations, while chemistry relies on matrices for quantum theory and molecular bonding. Matrix manipulation also plays a role in geometry, probability theory and statistics, physics, and circuitry.

Coined by James Joseph Sylvester in 1850, the term "matrix" can be thought of as "a rectangular array of terms, out of which different systems of determinants may be engendered as from the womb of a common parent." With so many applications and a history dating to the nineteenth century, matrices deserve their own programming language. Our goal with DARN is to create a language that excels in matrix manipulation, allowing users to easily and efficiently deal with a matrix.

1.2 Introduction

While many programming languages, such as Java, allow users to create a matrix with a two-dimensional array, they lack efficient and easy matrix manipulation. Filling this void, DARN is a programming language emphasizing matrix manipulation. Named after the first initials of our names, DARN includes a matrix data type and allows for efficient linear algebra calculations and easy access to rows and columns in matrices. For example, programmers can use DARN to populate matrices with arbitrary values, calculate the transpose or inverse of a matrix, find the determinant of a matrix, or compute scalar operations, matrix multiplication, matrix addition, and matrix subtraction.

DARN compiles to the Low Level Virtual Machine.

1.3 Features

DARN has a few key features, listed below.

- Strongly typed
- Imperative
- Supports control flow
- Includes matrix data type
- Efficient matrix manipulation
- Robust matrix-oriented standard library

2 Language Tutorial

2.1 Setup

DARN was developed in OCaml, which needs to be installed in order to use the compiler. To do this, install OPAM(OCaml Package Manager), which allows OCaml and related packages and libraries to be installed as well. When installing, make sure the version of the OCaml LLVM library matches the version of the LLVM system installed on your system.

2.2 Using the Compiler

Within the DARN folder, type 'make test' to generate the darn.native file. This file can be used to compile DARN code into LLVM code, which can be used in the LLVM compiler to print out a result. To write and execute a DARN program, the user must write a main function and follow the syntactical conventions of the language, outlined in the next sections.

2.3 Hello World

Before diving into the nitty-gritty details of DARN, let's first take a look at a simple Hello World program. The program below will print the string Hello, World! as output.

```
int main() {
        prints("Hello, World!\n");

}
```

2.4 Sample Program

Programs must define a main function with the following declaration:

```
int main() {


}
```

The main method can call other user-defined functions, which may be recursive. A user can define local and global variables and use control flow statements, such as if-else or for loops.

Here is an example of a program in DARN that creates a 1-Dimensional matrix with 10 integer elements. There are two for loops, one to initialize the values in the matrix and another to print them. The program prints 0123456789.

```
int main() {
        int i;
        int[10] x;
        for (i=0; i<10; i=i+1) {
                        x[i] = i;
        }
        for (i=0; i<10; i=i+1) {
                        print(x[i]);
        }
}
```

3 Language Reference Manual

DARN is a matrix manipulation language. Taking inspiration from the C language, DARN's design rests on efficient matrix handling and imperative programming.

3.1 Types

A data type is a classification of data which tells the compiler or interpreter how the programmer intends to use the data. In addition to primitive types, which are int, float, char, and bool, DARN includes an additional type: matrix. The table below outlines in more detail all of these types.

Type	Declaration	Description
int	int x;	32-bit integer data type, represented as binary signed two's complement bitstring internally
float	float y;	single-precision floating point number, floating point constants contain a decimal point or an exponent or both
char	char c;	1 byte character data type, including {A-Z}, {a-z}
bool	bool b;	1 byte Boolean data type, 0 represents false and 1 represents true internally
1D matrix	int[4] m;	one-dimensional matrix data type. All elements of a matrix must be of the same type. A matrix can only be composed of types int and float.
2D matrix	int[4][4] m;	two-dimensional matrix data type. All elements of a matrix must be of the same type. A matrix can only be composed of types int and float.
1D matrix pointer	int[ ] p;	pointer to a one-dimensional matrix
2D matrix pointer	int[ ][ ] p;	pointer to a two-dimensional matrix

3.1.1 Basic Types

A variable declaration specifies the variable type and variable name. In DARN, all variables must be declared before use and before writing any other statements of functions. Variables **cannot** be declared and initialized in the same line. Basic types are declared with the format:

type variable_name

Example:

```
/* declaration followed by initialization */
int x;
x = 2;


/* error: cannot declare and initialize in same line */
int x = 2;


/* error: must declare all variables in the beginning of program */
int a;
a = 2;
int b;
```

3.1.2 Matrices

Matrices in DARN can either be 1-Dimensional or 2-Dimensional. The elements of a matrix must be of the same type; a matrix can only be composed of integers (int) or floating point numbers (float).

Matrix Declaration, Initialization, and Access:

To declare a 1-D matrix with n number of elements, where n must be an integer, follow the format of

type[n] variable_name;

To access an element in the 1-D matrix and initialize it to an integer or float value, use the following format. The example below shows accessing of the element with index 1 in a 1-D matrix of size 5. This code will print 0.

```
int main() {
        int[5] m;
        m[1] = 0;
        print(m[1]);
}
```

To declare a 2-D matrix with m rows and n columns, where m and n are both integers:

type[m][n] variable_name;

To access an element in a 2-D matrix and initalize it to a value, see the example below, which shows initializing the element in the first row and first column (indices 0 for both) to 3. This code will print 3.

```
int main() {
        int[5][5] m;
        m[0][0] = 3;
        print(m[0][0]);
}
```

Matrix Built-In Functions

Matrices in DARN also have built-in functions, height, width, and len (appreviation for length).

len is only used for 1-Dimensional matrices and returns the number of elements in the matrix.

height and *width* are only for 2-Dimensional matrices, where height returns the number of rows and width returns the number of columns.

Example of height, which returns the number of rows, in this case it will print 5:

```
int main() {
        int[5][8] a;
        print(height(a));
}
```

Example of width, which returns the number of columns, in this case it will print 8:

```
int main() {
        int[5][8] a;
        print(width(a));
}
```

3.1.3 Pointers

One aspect of DARN is the ability to create pointers to matrices. This allows users to pass in references of matrices into functions without having to make copies of the matrix. Dereferencing the matrix will allow the user to access the elements of the matrix. The user can also increment the pointer to iterate over the elements of the matrix.

To get a pointer referencing a 1-D matrix, use the % symbol. For 2-D matrices, use %%. Below is an example that prints 9 in DARN.

```
int main() {
        /* Create a 1D matrix */
        int[4] x;
        /* Create a 1D matrix pointer */
        int[] y;
        int q;

        x[0] = 9;

        /* Point pointer to matrix reference */
        y = %x;

        /* Dereference the pointer to get the first value in matrix x */
        q = #y;
        print(q);
}
```

For pointer dereferencing, use the # symbol. Below is an example that prints 3.

```
int main() {
        int[5] y;
        int[] p;

        y[0] = 1;
        y[1] = 2;
        p = %y;
        p = ++p;
        #p = 3;
        print(y[1]);
}
```

To increment a pointer, use the ++ symbols. Below is an example that prints 2. Incrementing the pointer will increase the pointer's value by the size of the elements in the matrix, so that the pointer points to the next element in the matrix.

```
int main() {
        int[5] y;
        int[] p;

        y[0] = 1;
        y[1] = 2;
        p = %y;
        p = ++p;
        print(#p);
}
```

3.2 Lexical Conventions

3.2.1 Identifiers

Identifiers are sequences of characters used for naming DARN entities, such as variables or functions. Identifiers can be made up of upper and lower case letters, digits, and underscores. The first character of an identifier should be a lowercase letter, following the convention of Java and C languages. Upper and lowercase letters are distinct, so isEmpty is different from isempty. DARN's keywords may not be used as variable names. See the next section for details regarding keywords.

3.2.2 Keywords

Keywords are special identifiers reserved for use as part of the programming language itself, thus they may not be used for any other purpose. DARN recognizes the following keywords.

Keyword	Description
main	main function. The code within a main function will be executed when the executable file runs after compilation.
return	return function value
void	indicates no type
int,float,char, bool	basic types
for	for in a for loop*
if	if part of if-else or if-elif-else statements
else	else as part of if-else or if-elif-else statements
while	while in a while loop
true	Boolean literal value for true
false	Boolean literal value for false
height	number of rows of a matrix
width	number of columns of a matrix
len	length of a matrix

* see sections 3.4.1-3.4.3 for more information about statements and loops

3.2.3 Separators

A separator is a single-character that separates the tokens in a program.

Separator	Description
(	Left parenthesis. Used for function arguments, statements and loops.
)	Right parenthesis. Used for function arguments, statements and loops.
{	Left curly bracket. Part of block separator for functions.
}	Right curly bracket. Part of block separator for functions.
[	Left square bracket. Part of matrix declaration.
]	Right square bracket. Part of matrix declaration.
,	Comma.
.	Period.
;	Semi-colon.

3.2.4 Literals

A literal is a source code representation of a value of a primitive type.

Integer Literals:
An integer literal is expressed in decimal (base 10). It is represented with either the single ASCII digit 0, representing the integer zero, or an ASCII digit from 1 to 9 optionally followed by one or more ASCII digits from 0 to 9. That is, an integer can be expressed by the regular expression, ['0'-'9']+.

Float Literals:
A float literal is made up of an integer part, a decimal part (represented by the ASCII period),and a fraction part. The integer and fraction parts are defined by a single digit 0 or one digit from 1-9 followed by more ASCII digits from 0 to 9. That is, a float can be expressed by ['0'-'9']+ ['.'] ['0'-'9']+.

Boolean Literals:
A boolean (bool) literal is represented by ASCII characters. A bool literal is either true or false.

String Literals:
A string literal is represented as a sequence of zero or more ASCII characters enclosed in two double quotes, such as "hello, world". DARN does not include string data types, so the user cannot declare a string; however, he or she can print a string, as in:

```
prints(”Hello, World!”);
```

In the above example, the sequence of characters <hello, world>is the string literal.

3.2.5 Operators and Precedence

In mathematics and computer programming, an operator is a character that represents an action. For example, * is an arithmetic operator that represents multiplication. In computer programs, one of the most familiar sets of operators, the Boolean operators, is used to work with true/false values.

An operand is the part of a computer instruction which specifies what data is to be manipulated or operated on, while at the same time representing the data itself. The numbers 4 and 5 in the operation, 4 * 5, represent operands, while the * is the operator.

Operator	Description
=	Assignment operator. Note: the left and right hand sides of the assignment operator must be of the same data type.
*	Multiplication operator. Types of operands must match, such as int * int
/	Division operator. Types of operands must match.
+	Addition operation. Types of operands must match.
-	Subtraction operator. Types of operands must match.
<	Less than comparison. Type of operands must match. Returns a 1 or 0, for true or false respectively.
>	Greater than comparison. Type of operands must match. Returns a 1 or 0, for true or false respectively.
<=	Less than or equal to comparison. Type of operands must match. Returns a 1 or 0, for true or false respectively.
>=	Greater than or equal to comparison. Type of operands must match. Returns a 1 or 0, for true or false respectively.
==	Equal to comparison. Types of operands must match. Returns a 1 or 0, for true or false respectively.
!=	Not equal to comparison. Types of operands must match. Returns a 1 or 0, for true or false respectively.
&&	Logical AND operator. Types of operands must match. Returns a 1 or 0, for true or false respectively.
||	Logical OR operator. Types of operands must match. Returns a 1 or 0, for true or false respectively.
!	Logical NOT operator. Returns a 1 or 0, for true or false respectively.
-	Negation operator. Negates the value that follows it.
[ ]	1-D matrix operator. Use it to access the indices of the matrix.
[ ][ ]	2-D matrix operator. Use it to access rows or columns of a matrix.
%	1-D matrix pointer reference.
%%	2-D matrix pointer reference.
#	Dereference a pointer to a matrix, either 1-D or 2-D.
++	Increment a pointer.

For special matrix operations, see the Standard Library Functions, section 3.5.

Operator Precedence: If there is more than one operator present in a single expression, operations are performed according to operator precedence. Operators that share the same precedence are evaluated according to associativity. Left-associative operators evaluate from left to right, while right-associative operators evaluate from right to left. All operators are left-associative, except the assignment operator (=), not operator (!), and negation operator (-). The table below illustrates operator precedence in DARN.

Precedence	Operators
lowest	=
	||
	&&
	==, !=
	>, <, >=, <=
	+, -
	*, /, %
highest	!, -

3.2.6 Comments

Comments are useful when a user wants to make notes about his or her program code, as comments will be ignored by the compiler and excluded from the executable files. Comments are enclosed by a forward slash and an asterisk at the beginning and an asterisk and a forward slash at the end. The user cannot use nested comments. See below for examples of both single line and block line comments.

```
/* this is a single line comment */

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

3.3 Functions

Functions in DARN consist of a function header and a function body. The header contains the return type of the function, the name of the function (must be valid identifier), and an optional parameter list enclosed in parentheses. Each function must have a unique name. The function body is enclosed by a pair of curly braces. Below is the format for function declaration.

```
return_type function_name (parameters) {
    return return_value;
}
```

A function can return the following types:

- int
- float
- bool
- void

3.3.1 Function Calls and Usage

In order to be able to call a function, the function must have been declared already. If the function is part of the standard library, it does not need to be declared prior to use (see section 3.5). The function call will execute using the given parameters and return the value as defined by the function. All parameters will be passed by value, so a function can change the values of these parameters within the scope of its function block without affecting the arguments in the function call.

For all user-created programs, don't forget to include a main function.

```
int main() {


}
```

Here is a simple function declaration in DARN that takes in no parameters.

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

3.3.2 Recursion

DARN functions may also be used recursively. Recursion is a method in which the solution to a problem depends on solutions to smaller instances of the same problem.

One common example of recursion is the Fibonacci function, shown below, which prints 3.

```
int fib(int x) {
        if (x < 2) return 1;
        return fib(x-1) + fib(x-2);
}

print(fib(4));
```

3.3.3 Scoping Rules

DARN enforces scoping rules that give the program a clear structure. The scope of a name is the part of the program within which the name can be used. For a variable declared at the beginning of a function, the scope is the function in which the name is declared. Local variables of the same name in different functions are unrelated. The same is true of the parameters of the function. The scope of an global variable or a function lasts from the point when it is declared to the end of the file being compiled.

A variable is not accessible until after its declaration, when its scope begins.

```
/* y is not available here */

int y;
y = 10;

/* y is available from here on */
```

Another example using global and local variables. The first variable x's scope last for the entire file, while the x within foo is local to that function.

```
int x;
x = 5;

int foo() {
        int x;
        x = 2;
        print(x);
}

foo(); /* prints 2 */
print(x); /* prints 5 */
```

3.4 Control Flow

DARN supports if-else conditional statements, as well as while and for loops.

3.4.1 Conditional Statements

Conditional statements in DARN are denoted by the keywords if and else. They can be used in one of the following formats.

If Statement: For a single if statement, with no else statement, the program executes the statement if the expression evaluates to true. Otherwise, it continues on to subsequent lines. The user can use an if statement with the following format. Additionally, the user can omit the curly braces for a solo if statement.

```
if (expression) {
        statement;
}
```

Example:

```
/* prints 6 because x is greater than y */
int x;
int y;
x = 6;
y = 2;

if (x > y) {
        print(x);
}
```

If-Else Statement: With an else included, if the first expression evaluates to false, the statement following the else is executed. Ambiguity regarding else is resolved by connecting an else with the last encountered else-less if.

if (expression) {
        statement;
} else {
        statement;
}

Example:

```
/* prints 2 because y is greater than x */
int x;
int y;
x = 1;
y = 2;

if (x > y) {
        print(x);
} else {
        print(y);
}
```

3.4.2 Loops

There are two basic looping structures in DARN, the while loop and the for loop.

While Loop: A while loop will run the code inside the while block as long as the condition evaluates to true. The loop will not start unless this condition is met.

while (condition) {
        statement;
}

Example: This example would incrementally increase the variable "a" by 1, as long as a is still less than 5.

```
int main() {
        int a;
        a = 0;
        while (a<5) {
                a = a + 1;
        }
}
```

For Loop: In a for loop, the first expression specifies initialization for the loop; the second specifies a test or relational expression; and the third typically specifies an increment to be performed after each iteration. A program will begin with the first expression, check to make sure the second expression is true, then iterate through the block of code using the third expression. If the second expression is missing, the loop will run forever.

```
for(expression1; expression2; expression3) {
    statement;
}
```

Example:

```
/* prints 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 */
int i;
for (i=0; i<10; i=i+1) {
        print(i);
}
```

3.5 Standard Library Functions

The standard library of DARN has the following functions. It gets included using DARN's preprocess.ml file.

```
/* 1D Matrix Scalar Addition:
  Takes in one matrix pointer, a scalar, and the length of the
    matrix
  Adds to the matrix in memory */

void add_1D_scalar(int[] x, int scalar, int l) {

  int i;

  for (i=0; i<l; i=i+1) {
    #x = #x + scalar;
    x = ++x;
  }
}

/* 2D Matrix Scalar Addition:
  Takes in one matrix pointer, a scalar, and the height and
    width of the matrix
  Adds to the matrix in memory */

void add_2D_scalar(int[][] x, int scalar, int h, int w) {

```

```
  int i;

  for (i=0; i<(h*w); i=i+1) {
    #x = #x + scalar;
    x = ++x;
  }
}

/* 1D Matrix Scalar Subtraction:
  Takes in one matrix pointer, a scalar, and the length of the matrix
  Subtracts the values from the matrix in memory */

void sub_1D_scalar(int[] x, int scalar, int l) {

  int i;

  for (i=0; i<l; i=i+1) {
    #x = #x - scalar;
    x = ++x;
  }
}

/* 2D Matrix Scalar Subtraction:
  Takes in one matrix pointer, a scalar, and the height and width of the matrix
  Subtracts the values from the matrix in memory */

void sub_2D_scalar(int[][] x, int scalar, int h, int w) {

  int i;

  for (i=0; i<(h*w); i=i+1) {
    #x = #x - scalar;
    x = ++x;
  }
}

/* 1D Matrix Scalar Multiplication:
  Takes in one matrix pointer, a scalar, and the length of the matrix
  Multiplies the values from the matrix with the scalar in memory */

void mult_1D_scalar(int[] x, int scalar, int l) {

  int i;

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

```
    #x = #x * scalar;
    x = ++x;
  }
}

/* 2D Matrix Scalar Multiplication:
  Takes in one matrix pointer, a scalar, and the length of the matrix
  Multiplies the values from the matrix with the scalar in memory */

void mult_2D_scalar(int[][] x, int scalar, int h, int w) {

  int i;

  for (i=0; i<(h*w); i=i+1) {
    #x = #x * scalar;
    x = ++x;
  }
}

/* 1D Matrix Scalar Division:
  Takes in one matrix pointer, a scalar, and the length of the matrix
  Divides the values from the matrix with the scalar in memory */

void div_1D_scalar(int[] x, int scalar, int l) {

  int i;

  for (i=0; i<l; i=i+1) {
    #x = #x / scalar;
    x = ++x;
  }
}

/* 2D Matrix Scalar Division:
  Takes in one matrix pointer, a scalar, and the length of the matrix
  Divides the values from the matrix with the scalar in memory */

void div_2D_scalar(int[][] x, int scalar, int h, int w) {

  int i;

  for (i=0; i<(h*w); i=i+1) {
    #x = #x / scalar;
```

```
    x = ++x;
  }
}

/* 1D Int Matrix addition:
  Takes in two matrix pointers and the length of the matrices
  Adds the second matrix into the first in memory */

void add_1D_int(int[] x, int[] y, int l) {

  int i;

  for (i=0; i<l; i=i+1) {
    #x = #x + #y;
    x = ++x;
    y = ++y;
  }
}

/* 2D Int Matrix addition:
  Takes in two matrix pointers and the height and width of the matrices
  Adds the second matrix into the first in memory */

void add_2D_int(int[][] x, int[][] y, int h, int w) {

  int i;

  for (i=0; i<(h*w); i=i+1) {
    #x = #x + #y;
    x = ++x;
    y = ++y;
  }
}

/* 1D Float Matrix addition:
  Takes in two matrix pointers and the length of the matrices
  Adds the second matrix into the first in memory */

void add_1D_float(float[] x, float[] y, int l) {

  int i;

  for (i=0; i<l; i=i+1) {
    #x = #x + #y;
    x = ++x;
    y = ++y;
  }
}
```

```

/* 2D Float Matrix addition:
  Takes in two matrix pointers and the height and width of the matrices
  Adds the second matrix into the first in memory */

void add_2D_float(float[][] x, float[][] y, int h, int w) {

  int i;

  for (i=0; i<(h*w); i=i+1) {
    #x = #x + #y;
    x = ++x;
    y = ++y;
  }
}

/* 1D Int Matrix subtraction:
  Takes in two matrix pointers and the length of the matrices
  Subtracts the second matrix from the first in memory */

void sub_1D_int(int[] x, int[] y, int l) {

  int i;

  for (i=0; i<l; i=i+1) {
    #x = #x - #y;
    x = ++x;
    y = ++y;
  }
}

/* 2D Int Matrix subtraction:
  Takes in two matrix pointers and the height and width of the matrices
  Subtracts the second matrix from the first in memory */

void sub_2D_int(int[][] x, int[][] y, int h, int w) {

  int i;

  for (i=0; i<(h*w); i=i+1) {
    #x = #x - #y;
    x = ++x;
    y = ++y;
  }
}

/* 1D Float Matrix subtraction:
```

```
  Takes in two matrix pointers and the length of the matrices
  Subtracts the second matrix from the first in memory */

void sub_1D_float(float[] x, float[] y, int l) {

  int i;

  for (i=0; i<l; i=i+1) {
    #x = #x - #y;
    x = ++x;
    y = ++y;
  }
}

/* 2D Float Matrix subtraction:
  Takes in two matrix pointers and the height and width of the matrices
  Subtracts the second matrix from the first in memory */

void sub_2D_float(float[][] x, float[][] y, int h, int w) {

  int i;

  for (i=0; i<(h*w); i=i+1) {
    #x = #x - #y;
    x = ++x;
    y = ++y;
  }
}

/*
  2D Int Matrix Multiplication
  Takes in two matrices for multiplication and an output matrix.
  Takes in the height and width of the two input matrices
  The Output matrix must be of size height = height of 1st matrix
  and width = width of 2nd matrix.
  Store the variables in the output matrix. Returns nothing.

*/
void mult_2D_int(int[][] x, int[][] y, int[][] output, int h1, int w1, int h2, int w2) {

  int i;
  int j;
  int k;
  int l;
  int[][] temp_x;
  int[][] temp_y;
```

```
  int[][] temp_output;
  temp_output = output;

  /* Zero out output matrix*/
  for (i=0;i<h1;i=i+1) {
    for (j=0;j<w2;j=j+1) {
      #temp_output = 0;
      temp_output = ++temp_output;
    }
  }

  for (i=0;i<h1;i=i+1) {
    for (j=0;j<w2;j=j+1) {
      temp_x = x;
      temp_y = y;

      for (k=0;k<(i*w1);k=k+1){
        temp_x = ++temp_x;

      }
      for (l=0;l<j;l=l+1) {
        temp_y = ++temp_y;
      }

      for (k=0;k<w1;k=k+1) {
        #output = #output + (#temp_x * #temp_y);
        temp_x = ++temp_x;
        for (l=0;l<w2;l=l+1) {
          temp_y = ++temp_y;
        }
      }
      output = ++output;
    }
  }

}

/*
  2D Float Matrix Multiplication
  Takes in two matrices for multiplication and an output matrix.
  Takes in the height and width of the two input matrices
  The Output matrix must be of size height = height of 1st
    matrix
  and width = width of 2nd matrix.
  Store the variables in the output matrix. Returns nothing.

*/
void mult_2D_float(float[][] x, float[][] y, float[][] output,
    int h1, int w1, int h2, int w2) {
```

```

  int i;
  int j;
  int k;
  int l;
  float[][] temp_x;
  float[][] temp_y;
  float[][] temp_output;
  temp_output = output;

  /* Zero out output matrix*/
  for (i=0;i<h1;i=i+1) {
    for (j=0;j<w2;j=j+1) {
      #temp_output = 0.0;
      temp_output = ++temp_output;
    }
  }

  for (i=0;i<h1;i=i+1) {
    for (j=0;j<w2;j=j+1) {
      temp_x = x;
      temp_y = y;

      for (k=0;k<(i*w1);k=k+1){
        temp_x = ++temp_x;

      }
      for (l=0;l<j;l=l+1) {
        temp_y = ++temp_y;
      }

      for (k=0;k<w1;k=k+1) {
        #output = #output + (#temp_x * #temp_y);
        temp_x = ++temp_x;
        for (l=0;l<w2;l=l+1) {
          temp_y = ++temp_y;
        }
      }
      output = ++output;
    }
  }

}

/*
  2D Int Matrix Transpose
  Takes in one input matrix and an output matrix.
  Takes in the height and width of the input matrix
```

```
  The Output matrix must be of size height = width of input
    matrix
  and width = height of input matrix.
  Computes the transpose of the input matrix.
  Store the variables in the output matrix. Returns nothing.

*/

void transpose_2D_int(int[][] x, int[][] output, int h, int w) {

  int i;
  int j;
  int k;
  int[][] temp_x;
  int[][] temp_output;
  temp_x = x;
  temp_output = output;

  /* Zero out output matrix*/
  for (i=0;i<w;i=i+1) {
    for (j=0;j<h;j=j+1) {
      #temp_output = 0;
      temp_output = ++temp_output;
    }
  }

  /* Copy into output matrix */
  for (i=0;i<w;i=i+1) {
    for (j=0;j<h;j=j+1) {
      temp_x = x;
      for (k=0;k<i;k=k+1) {
        temp_x = ++temp_x;
      }

      for (k=0;k<(j*w);k=k+1) {
        temp_x = ++temp_x;
      }

      #output = #temp_x;

      output = ++output;

    }
  }
}


/*
  2D Float Matrix Transpose
```

```
  Takes in one input matrix and an output matrix.
  Takes in the height and width of the input matrix
  The Output matrix must be of size height = width of input matrix
  and width = height of input matrix.
  Computes the transpose of the input matrix.
  Store the variables in the output matrix. Returns nothing.

*/

void transpose_2D_float(float[][] x, float[][] output, int h, int w) {

  int i;
  int j;
  int k;
  float[][] temp_x;
  float[][] temp_output;
  temp_x = x;
  temp_output = output;

  /* Zero out output matrix*/
  for (i=0;i<w;i=i+1) {
    for (j=0;j<h;j=j+1) {
      #temp_output = 0.0;
      temp_output = ++temp_output;
    }
  }

  /* Copy into output matrix */
  for (i=0;i<w;i=i+1) {
    for (j=0;j<h;j=j+1) {
      temp_x = x;
      for (k=0;k<i;k=k+1) {
        temp_x = ++temp_x;
      }

      for (k=0;k<(j*w);k=k+1) {
        temp_x = ++temp_x;
      }

      #output = #temp_x;

      output = ++output;

    }
  }
}

```

```
/*
  Takes in 1D matrix pointer and the matrix length
  populates it with zeros
*/
void zero_1D_int(int[] x, int l) {
  populate_1D_int(x,0,l);
}

/*
  Takes in 2D matrix pointer and the matrix height and width
  populates it with zeros
*/

void zero_2D_int(int[][] x, int h, int w) {
  populate_2D_int(x,0,h,w);
}

/*
  Takes in 1D matrix pointer and the matrix length
  populates it with a scalar 'a'
*/

void populate_1D_int(int[] x, int a, int l) {
  int i;
  for (i=0;i<l;i=i+1) {
    #x = a;
    x = ++x;
  }
}

/*
  Takes in 2D matrix pointer and the matrix height and width
  populates it with a scalar 'a'
*/

void populate_2D_int(int[][] x, int a, int h, int w) {
  int i;
  for (i=0;i<(h*w);i=i+1) {
    #x = a;
    x = ++x;
  }
}

/* Determinant of 2x2 and 3x3 for Ints:
  Takes in 2D matrix pointer and matrix height and width
  Finds the determinant of a matrix of ints */

int det_int(int[][] x, int he, int w) {
  int a;
```

```
  int b;
  int c;
  int d;
  int e;
  int f;
  int g;
  int h;
  int i;
  int det;
  if ((he==2 && w==2) || (he==3 && w==3))  {
    a = #x;
    x = ++x;
    b = #x;
    x = ++x;
    c = #x;
    x = ++x;
    d = #x;
    x = ++x;
    if (w==2){
      det = (a*d)-(b*c);
    } else {
      e = #x;
      x = ++x;
      f = #x;
      x = ++x;
      g = #x;
      x = ++x;
      h = #x;
      x = ++x;
      i = #x;
      det = a * (e * i - f * h) - b * (d * i - f * g) + c * (d * h - e * g);
    }
  } else {
    return 0;
  }
  return det;
}

/* Determinant of 2x2 and 3x3 for Floats:
  Takes in 2D matrix pointer and matrix height and width
  Finds the determinant of a matrix of floats */

float det_float(float[][] x, int he, int w) {
  float a;
  float b;
  float c;
  float d;
  float e;
```

```
  float f;
  float g;
  float h;
  float i;
  float det;
  if ((he==2 && w==2) || (he==3 && w==3))  {
    a = #x;
    x = ++x;
    b = #x;
    x = ++x;
    c = #x;
    x = ++x;
    d = #x;
    x = ++x;
    if (w==2){
      det = (a*d)-(b*c);
    } else {
      e = #x;
      x = ++x;
      f = #x;
      x = ++x;
      g = #x;
      x = ++x;
      h = #x;
      x = ++x;
      i = #x;
      det = a * (e * i - f * h) - b * (d * i - f * g) + c * (d * h - e * g);
    }
  } else {
    return 0.0;
  }
  return det;
}

/* Computes the inverse of a 2D float matrix
  Takes in matrix pointer, height and width
  returns the inverse
  */

float inverse_float(float[][] x, int h, int w){
  float ret;
  if ((h==3 && w==3) || (h==2 && w==2)){
    ret = det_float(x, h, w);
    if (ret != 0.0){
      return 1.0/ret;
    }
    return 0.0;
  } else {
```

```
    return 0.0;
  }
}

/* ------------- PRETTY PRINTING -------------*/

/* Print 1D matrix of ints, takes in matrix pointer and matrix length */

void print_1D_int(int[] x, int l) {
  int i;
  prints("[\t");
  for (i=0; i<l; i=i+1) {
    print(#x);
    prints("\t");
    x = ++x;
  }
  prints("]\n");
}

/* Print 1D matrix of floats, takes in matrix pointer and matrix length */

void print_1D_float(float[] x, int l) {
  int i;
  prints("[\t");
  for (i=0; i<l; i=i+1) {
    printf(#x);
    prints("\t");
    x = ++x;
  }
  prints("]\n");
}

/* Print 2D matrix of ints, takes in matrix pointer and matrix height and width */

void print_2D_int(int[][] x, int h, int w) {
  int i;
  int j;
  prints("[\n");
  for (i=0; i<h; i=i+1) {
    prints("|\t");
    for (j=0; j<w; j=j+1) {
      print(#x);
      prints("\t");
      x = ++x;
    }
    prints("|\n");
```

```
  }
  prints("]\n");
}

/* Print 2D matrix of floats, takes in matrix pointer and matrix
    height and width */

void print_2D_float(float[][] x, int h, int w) {
  int i;
  int j;
  prints("[\n");
  for (i=0; i<h; i=i+1) {
    prints("|\t");
    for (j=0; j<w; j=j+1) {
      printf(#x);
      prints("\t");
      x = ++x;
    }
    prints("|\n");
  }
  prints("]\n");
}
```

4 Project Plan

4.1 Planning Process

To begin this project, the DARN team first assigned project roles and set up a weekly meeting time. While we didn't always meet on this day each week, we generally chose to work on Wednesday or Friday evenings. Every Monday at 5:30pm, we would report to our TA, Alexandra Medway, who helped us track our progress and resolve any issues we encountered.

Regarding tools employed, we used Github as a repository for our code and a group text message to collaborate and plan.

4.2 Specification

Throughout our development process, the C language served as our inspiration. Many features and design ideas in DARN have been influenced by C, such as function declarations. Our original specification for DARN was outlined in the initial Language Reference Manual. From then on, the specification was built iteratively as we coded. Our final specification was detailed in our LRM. Whenever DARN diverged from the LRM, we updated the LRM to maintain consistency.

4.3 Development and Testing

Our development process followed the stages of the compiler. We tried to finish the scanner and parser quickly, so that semantic analysis and code generation could be tackled. Once we had our skeleton of a compiler, we built each feature from end to end, ie. from AST to codegen. We also placed tests at the center of our development process and coupled every feature with a set of accompanying test cases.

4.4 Style Guide

We used the following conventions while programming our DARN compiler, in order to ensure consistency, readability, and transparency.

- OCaml editing and formatting style to write code for compiler architecture
- C language editing and formatting style for inspiration for DARN program code

A few other style guidelines to note:

- File names end in .darn
- Variable identifiers begin with a lowercase letter and are camelcase
- Function identifiers begin with a lowercase letter and are camelcase
- Always include a main function in DARN programs

4.5 Timeline

Date	Milestone
September 28	Proposal
October 11	Scanner
October 19	Parser
October 24	AST
October 26	LRM
October 30	Pretty Printing
November 5	Test Suite Reorganized
November 16	Semantic Analysis
November 19	Codegen
November 21	Hello World
November 30	Matrix Handling
December 5	Tests Added
December 16	Tuples
December 18	Final Touches and Bug Fixes
December 20	Final Report

4.6 Challenges

One of the greatest challenges we faced was determining the features and identity of our language. Numerous questions arose. Do we want to focus on file input and output and image editing processes? Should we incorporate three-dimensional matrices? Do we want to make our language a mathematical matrix manipulation language? What should be included in our Standard Library?

After grappling with these questions and receiving feedback on our initial Language Reference Manual, we chose to design a matrix manipulation language that excels in mathematical calculations. This simplified our process and re-focused our intentions. From this challenge to the many smaller ones we encountered, collaboration and communication were keys to our success.

4.7 Roles and Responsibilities

Manager- Timely completion of deliverables: Daisy Chaussee
Language Guru- Language design: Ignacio Torras

System Architect- Compiler architecture, system environment: Rafael Takasu
Tester- Test plan, test suites: Anthony Kim

4.8 Software Development Environment

Operating Systems: Mac OS Systems, Ubuntu 15.10 on Virtual Box
Languages: OCaml (used OPAM to install), C (for inspiration)
Text Editor: Sublime, Vim
Version Control: Git, GitHub
Documentation: LaTeX

4.9 Project Log

```
2a0926f84d6a2381740d2457ffb0bea55de5af8c merging
46e32d183d0217f5a423721ec04a9c7d2e755775 commenting stdlib
9ce540d353592fd3c89a7476280aced5b6c377f8 cleaning up code
e1d84875a06a0b8f37447d593f60f44b1d7a5e7a Merge branch 'master' of https://github.com/ak3703/DARN
28b5a5f6366e747320f8416c1c727c36892a9f4a demo 3
6fc398658d80e9920ac4938459e41acb5ff585e6 remove end of stdlib
da79269a003b0a77ab122edbee54bd0c450d4a93 Merge branch 'matrix_literals'
c4e4e21a47c072cb2b5732a89441d357c5af6ae2 added matrix literals for 1d
ca057c8d72db3a1e505149e451593825683e1519 Demo2: bubblesort
03322bebc8d0a8c1593942bc10ec4a94ea0fcca2 Merge branch 'master' of https://github.com/ak3703/DARN
c0001b5ef244301bf3e57b5b25353456b376b5ca demos
eb92dc5c306b8aee3cc2bcd4b26ee0aede25f87f inverse for float added
1e8d3d7d5e708d061350eef4ac4ef7b70e129360 added float det for matrices
c467fc9e43b79a5b8366321965f161c61153b7bb you can now return floats
d0c89aa75e9dd645f43fad0b736e5db69f96906e refactor
e923b940d266186a3d53799a61a09751b75dc9e8 fixed test
4b345206b07443ed738bcfd71e383a856532aa4d merging
8414cb75ea86ee3ba71d76989bfad1b5269fb7a3 removed newline from print functions, pretty print for matrices
```

```
cf6e7ea257e9a8a1b8369e7722c15ebb85151df1 added det_int to stdlib
b1c082044d1352bf25806cdb2f2077993bdfccb6 Merge branch 'master' of https://github.com/ak3703/DARN into det_and_neg
c5b37168cafd982921fb807b09aa663e63bc0c4c added negate and tests
69b100857f546fc5e5ce7042947b0ddb0b35c0eb merged
c57e13094493f0f9602f12ec798356dd47804956 matrix transpose
202d6db72c374f81039c5c796efdff88374193fe print 1d and 2d floats and ints
3d2031cc6ae2b4717162248ba1068829a2e71ae9 Merge branch 'master' of https://github.com/ak3703/DARN
6378c9b7a0c79f58ce2deb4d0c77c691b6e5cb8b added matrix multiplication
1065b1ede64c2c7bdd78228a9f73168884340413 more test
99e5766b88cb516b74e30dd5c73abf019141d449 int binops now also predict out of bounds
93e64ecccd5df60530b965258a36bdfeb6c04ab4 Merge branch 'master' of https://github.com/ak3703/DARN
372525fae392dc9050408067d742774a155dcba9 added out of bounds
43ec676da818db3f042263fd3e0682e9be0bfa5a added scalar and basic matrix operations
8f5552a8620d0e6d297769919d242e467670163a all tests pass
9757d0ab261d7be75efddbb84f43d8fa26162d99 Use: ./darn.native -c file.darn dummy_stdlib.txt
5ffb70cedafed44095ffbe35ab63743a0c54a8a5 Merge branch 'master' of github.com:ak3703/DARN
62cc125623d6e02f33c39b92c748cd7946cb6306 added reading standard library
162d32c5ce01e86515fd8d263e324c30a06b7d72 fixed warning
724686c251f825cbe3ce81d73dd48ac6754e9ff2 fixed some tests and added some
004d5769daf5b6ad8e0517f53f297f4ea1f4ab23 adding floats
287db6060379917b75290b124bb63f92cd324556 fixed height/width error
5e561021b6681bc6e4218b98d4a4c9d08fce0d60 Merge branch 'rows'
39b914ea66424686e99b304ff29e547b41a8b916 added len, height, and width
04af7af896ee1fafce037a6db9de132c5ce382dd Merge branch 'failTestsContd'
53031b6fcc64c6d32dae90f83ba4c6ad7afed8a3 merging
3d3b5df77ebc0211f047016228aae3f86e6b825b Merge branch 'master' of https://github.com/ak3703/DARN
dc65af64ec261102d9e8184df5b601fe12eaab87 pointer increments, changed pointer reference to %
e8a8862c05bf6be787835ef4315d7359502dd824 Merge branch 'master' of https://github.com/ak3703/DARN
1b9c4722397fc2d1fb5375988590769b0f9efb0f added fail tests for func, main
3c766356a240b9f653e15bffe2dbad0a7deb3459 pointer test
2118fab6cebb6025a0b5be554167a57f9eddeb9a added fail test for func
2d9029bebdcabbbb53b1873681e2afca0939e0ba added fail test suite
dbd6f6891520be8b48579092879b2e8dd403131a Merge pull request #2 from ak3703/failTests
2020ac732d9c845a27264b3f04b52113b79ac314 added pointers
ec7818cef3802ed4cf9e0b5152c04099838a6437 Added fail tests for expr, for, assign
5f922d49c022a1efbc95c47fb4ed64651272193b added while test
798511c2ddc21b39389d5008bb2439e3eb3f8cb6 Merge branch 'master' of https://github.com/ak3703/DARN
c6e2d7b9d355778881ed95b5e5ebbd6826a683fc pushing
9969d03e7b08be2b2192c2fd728c3175f95b4612 added test for global var
06a349804e0c3a5f6b4f0f5957f813416ca95264 2D matrices working on codegen
6dcf861c1273bba3e13d74be4dd68104c34dce98 added tests for nested for loops
7864bc9019038c5635c8c992971393d1223f2d50 2d matrices working in parser
```

```
fa96076937246ac323abaca7b54879f42f84fdf4 added fibonacci example
5fe69d3d963608d16661f6ee07d5cb6f5004f73c added tests for for-loops
441d352957aad51a91cc66d17e807ca783d10e01 updated .gitignore to avoid adding unnecessary files
bdcebd9457d8cc3de291176486b5ddf54698f3ea added printing tests for floats, booleans, and strings
827b32eae1df73c47e3ff14f59f520b5deddcdbd adding return
5dc0e7a2f2be790823f22bf2a6c37a0321b8e85f new test
7106f13f7ede59f0f8b96587cb5cd33d924156a8 don't add stuff form ocaml
650c78d4d08fffa4af2f1ca9a1972c4b31ba789d changed test output
0fe272532e4ec45e242dc485ffe1ffff3622c638 Merge branch 'master' of github.com:ak3703/DARN
3eb8a7a3b8a98bd8fed895d89131b7386a92f221 changed pretty print and updated output files in parser test
f0021d5f8555ca2ca1366710637a1458b82bcd6c for and while loops on parser (cleaned make)
43a71a4eab735aa0638eaa0b9f67562157ff4657 for and while loops on parser:
eb90dc8876125e634651cc16ec93813f441ce75e if statement on parser
5585dd58af9b3a21b84c1691e9a66f3c43a0f48c pushing
3035eb292258c5cdf47d92f80c1485efacdf9af6 added char and string, but string is only in scanner
ea78206fc799b185866be5ad8c0bbca81dc08bca  added exceptions, test_custom.sh will test using user input
e6f3f8ef902496ee2ee43600d825ec590214e000 pretty prints the errors
d8277a79b6e5f3cfca3882f312a23bcb70bde454 assigning to matrix index and matrix access working on codegen
103e285ffc8e28b4a9923a44fe3e73197a38f28e now you can assign to matrices
df2cc39a0c4314430769e038fc53a1ffabec001d Matrix added to parser, but we need to change assign from string * expr to expr * expr
11bcadff5df5610aa73f3c0e91469f3494daaa80 added printf for floats
06275b401e5d3b224f53a47edc5f30221f5edede editing travis.ci settings
cb4eb701c63afb2342b540e31220af77d0c29d21 Codegen for printing an integer and added tests for the compiler
040c8b6651c21c4ca22e42f646b7360bbefb97cd dependencies
957e50111a9a13067aa9d05095df2c00cbb0ef6d starting codegen
e90dacbc0d8e57561ca07f8834acbc66eee231bd right now we can only print ints since we do not have type String so changed test function 2. Added semantic
check to parserize, and added ast and semantic to makefile for tests
4064bafad60660e3ddff98d514c61a3affbee5e1 added functions from parserize to ast, and now semant works all the way through except for floats
fc7af931f77ee5c35bd3e08a2cf3c6edb47821cc added semant to makefile
7e27a4d250d5c497b9b3dcb36e65c400792b54c4 Added semant.ml file
227d5ae5a65038c0db40de6f2d8e83c026ebd6f0 added fdecls and tests
bbf69561ee296dd0cdcc5e5dd31860a0cc8b662b changed parserize
863bf345e5800e909b36cf6df72d43ce765beeb7 added a short example
b368eb4ddbc69da6a111ed91172ba29ee134dede added make clean for parser
6d7baeea8d15b972338a26ea51b245ad6dfc299a got variable initialization to work and reorganized
eb4a2583de8333904f202650a367cc445d2a25bf code
d4fc8753097c706997cada353e977ab36753f99e reorganized compiler and test suite
8e753f37fa6dcebf1c2d1b8ad44f6293ce60ede7 Parser can take in a list of expressions and pretty print it
cd3315fc5bf446ac28af41ecac0798c14acf7e94 Created tokens.in/.out
ed3e415b0ed1e4af7d8c5d1343e5fafeabb42dfb Added printer for Scanner and organized test files / Makefiles
fb3c5f073ef293f3895b337b0b0998e5e2db098b organized files
19e09c700353d0f7da5be200fefa285488d113b4 edited test suite
e3a2127144ebb492d0c7f5deb301401ee7f1a7fd moved assign to expr in ast
abfff4d09de462b5928253573244a3031c9b5a42 added bool
abebc6cfa93203dec6c7bf9240fae598dba35c48 added bool
a50102b49f4d6af07582c932a13ce8082b03876a added bool
da60417a18d8458f814bcd36bb8c1d52e1812e68 Update parser.mly
008ce082c10b1bdbc741f111a0d4a7289d8e43fc added statements
84b1706434c459d6764db98bf0297e3d3216027a Update ast.mli
b53e4fa620dedacca66b7af58995d3f2a3657888 Update parser.mly
a4c9b148670e6ee04aa9fb8339b41709cc4e2bae Merge branch 'master' of https://github.com/ak3703/DARN
0e53b878b29de415368f90f5214e901075b1f87b make clean
843c11acbbadf04a8be07909358a3864ef86b5ce added assignment operator
79d87e654245254dba07387f0a92a93e9781f30f Update parser.mly
965a0ceb0afeb6c7d22365a128b527986b1b6bb9 added sudo
109dc1d30c8b36d95c14f97ec8320dddb9361ce7 edited travis.yml file
bd908960368b0aeabc93af923dd239f60256573b Merge branch 'master' of github.com:ak3703/DARN
bd3cc8e1ffb64ea4bf129cac859360f38dd108e7 adding config for installing ocaml
420fca0efa6d14d517e83977c01bb2656f31ee34 Merge branch 'master' of https://github.com/ak3703/DARN
6d1f52bb9dc5774f69039336628233acee80fa0b added the files that are made into the gitignore so that we dont accidentaly push all the extra files
2e54c447934452f63871ad65c73e72995658c70d Merge branch 'master' of github.com:ak3703/DARN
0384d4fcad51123603c67266a5b4ae0cdf3620f6 updated yml file
647ed5e6b0c61a405fe00f816a03e8a14c72b07f removed unecessary files'
fc1fd0e91aff8e01a84265576bb2daadbb98ce22 Merge branch 'master' of https://github.com/ak3703/DARN
8d0c7244616a3c9e997e636f801e3010528c7377 added primitives and true and false to tokens
9ea2aa1c4a038df2c119d7cd22a2d60287b81215 added test code when using make
7ca63c54bed4f1a6f9c21fead4ceed18d967492a added equal to and not equal to
ff10112b09fff52920aab47a6d66d6c6342c4e6c added floats and changed variable to Id because that seems like the way everyonen else does it
5d50c6a4052460bb99b8ae07643c83954c343b3f added Or and And expressions
ada06f910893b3fd183869ecab2deb881ea5c37c Added gt, lt, leq, geq. Changed Makefile for calc -> darn change.
965a4e75d4818c9751257ef80398fddb563b455c Update scanner.mll
eac2383a2862b401843f95a649e059a2faa1a63d forgot to make clean
d14e755fd85a7f09f944331c1055789958c44689 added unop and NOT
49a99b328b751bc8036eb9cd9ce79c3c6e70217a VARIABLE to Variable and added match in calc.ml
f57843dd9d35c3acb11d32a20b085ccc1eed2586 Merge pull request #1 from ak3703/adding_ids
dad2f8d23c78a545bc65af816d58398351a30f83 added variables
9321daec407d62e2ed10c35fcbcb7bb0d5a6243c Added tokens and associativity for assignment, punctuation, etc.
fb0b982d81feb11fe334a611bc590f2a8934505a Added tokens for punctuation, assignment, &&, etc.
f4816839a45a35d080e5ff6a550f4ea35d8b86f2 added empty test folder / script and added print_prog for arithmetic
f4822de70a23a35aae079af3735928a2987baeef Added Literal to parser, scanner, and AST
5ac10e06267635aa6a257646bc930063d663e19f Basic setup
```

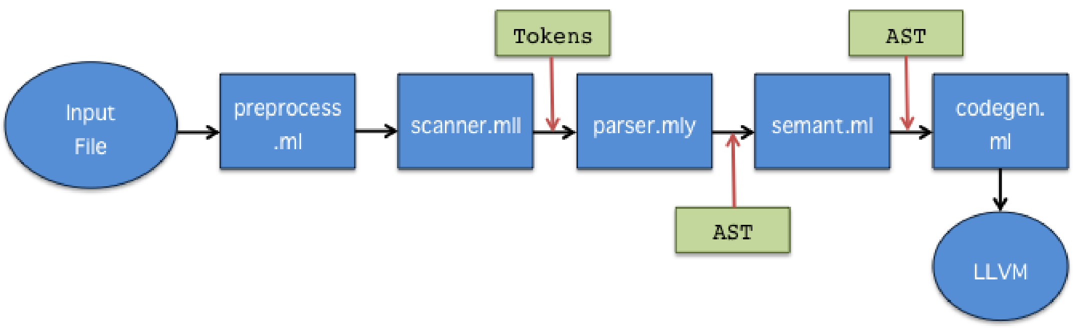
5 Architectural Design

The DARN compiler runs a program through the following components sequentially.

- Pre-Processor
- Scanner
- Parser
- Semantic Analysis
- Code Generation

5.1 Architecture Diagram

The following diagram illustrates the architecture and major components of the DARN compiler.



5.2 Pre-Processor (preprocess.ml)

The Pre-Processor is needed to include standard library functions.

5.3 Scanner

The scanner takes in a raw program file as input and generates tokens from the program. Tokens include identifiers, keywords, operators, literals, and important symbols and separators. The scanner also removes spaces and comments. It will report an error for any unrecognized symbols or incorrect tokens.

5.4 Parser

The parser takes the tokenized program from the scanner and uses a defined grammar to match tokens. If there are mismatches between the tokens and the grammar, the parser raises a syntax error, causing the compiler to exit. If there aren't any syntax errors, the parser generates an Abstract Syntax Tree (AST). The AST represents the syntactic structure of the source code. The composition of the AST is defined in ast.ml. This structure is then passed on to semant.ml for semantic analysis.

5.5 Semantic Analysis

The semantic analysis component of the DARN compiler takes the Abstract Syntax Tree structure and performs checks on it. These checks include checking if values and functions are redefined (adhering to scoping rules as well), checking if DARN keywords are redefined in the code, checking if correct names and expressions are referenced, and overall enforcing semantic constraints of the DARN language.

5.6 LLVM Code Generation

The Low Level Virtual Machine (LLVM) code generation uses the Abstract Syntax Tree from semant.ml to construct the LLVM IR, the final stage of the compiler. The LLVM generator first iterates through the tree and produces LLVM code for each function, statement, and expression. Once this inheritance code is generated, the code generator iterates through the entire semantically checked Abstract Syntax Tree and again produces the necessary LLVM code for each function, statement, and expression. This is done using the OCaml LLVM module. The LLVM code produced from codegen.ml can then be compiled using the LLVM compiler to produce output.

6 Test Plan

Below are two representative source language programs along with the target language program generated in LLVM for each. The first example shows basic 1-D matrix declaration and initialization. The second example shows multiplication of 2-D integer matrices.

6.1 Test Example 1

6.1.1 Example 1 in Native Language

```
int main() {
  int[5] m;
  int j;
  j = 1;
  m[j] = 0;
  print(m[1]);
}
```

6.1.2 Example 1 in Target Language

```
; ModuleID = 'DARN'

@fmt = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.1 = private unnamed_addr constant [3 x i8] c"%f\00"

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

define i32 @main() {
entry:
  %m = alloca [5 x i32]
  %j = alloca i32
  store i32 1, i32* %j
  %j1 = load i32, i32* %j
  %m2 = getelementptr [5 x i32], [5 x i32]* %m, i32 0, i32 %j1
  store i32 0, i32* %m2
  %m3 = getelementptr [5 x i32], [5 x i32]* %m, i32 0, i32 1
  %m4 = load i32, i32* %m3
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
    inbounds ([3 x i8], [3 x i8]* @fmt, i32 0, i32 0), i32 %m4)
  ret i32 0
}
```

6.2 Test Example 2

6.2.1 Example 2 in Native Language

```
void mult_2D_int(int[][] x, int[][] y, int[][] output, int h1,
    int w1, int h2, int w2) {

  int i;
  int j;
  int k;
  int l;
  int[][] temp_x;
  int[][] temp_y;
```

```
  int[][] temp_output;
  temp_output = output;

  /* Zero out output matrix*/
  for (i=0;i<h1;i=i+1) {
    for (j=0;j<w2;j=j+1) {
      #temp_output = 0;
      temp_output = ++temp_output;
    }
  }

  for (i=0;i<h1;i=i+1) {
    for (j=0;j<w2;j=j+1) {
      temp_x = x;
      temp_y = y;

      for (k=0;k<(i*w1);k=k+1){
        temp_x = ++temp_x;

      }
      for (l=0;l<j;l=l+1) {
        temp_y = ++temp_y;
      }

      for (k=0;k<w1;k=k+1) {
        #output = #output + (#temp_x * #temp_y);
        temp_x = ++temp_x;
        for (l=0;l<w2;l=l+1) {
          temp_y = ++temp_y;
        }
      }
      output = ++output;
    }
  }

}

int main() {

  int[4][3] a;
  int[3][4] b;
  int[4][4] c;

  int i;
  int j;

  for (i=0; i<height(a); i=i+1) {
    for (j=0;j<width(a);j=j+1) {
      a[i][j] = i+j;
```

```
    }
  }

  for (i=0; i<height(b); i=i+1) {
    for (j=0;j<width(b);j=j+1) {
      b[i][j] = i+j;
    }
  }

  mult_2D_int(%%a, %%b, %%c, height(a), width(a), height(b),
    width(c));

  print(c[3][3]);


}
```

6.2.2 Example 2 in Target Language

```
; ModuleID = 'DARN'

@fmt = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.1 = private unnamed_addr constant [3 x i8] c"%f\00"
@fmt.2 = private unnamed_addr constant [3 x i8] c"%d\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%f\00"

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

define i32 @main() {
entry:
  %a = alloca [4 x [3 x i32]]
  %b = alloca [3 x [4 x i32]]
  %c = alloca [4 x [4 x i32]]
  %i = alloca i32
  %j = alloca i32
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %
    merge, %entry
  %i14 = load i32, i32* %i
  %tmp15 = icmp slt i32 %i14, 4
  br i1 %tmp15, label %while_body, label %merge16

while_body:                                       ; preds = %
    while
  store i32 0, i32* %j
  br label %while1

```

```
while1:                                                  ; preds = %while_body2, %while_body
  %j10 = load i32, i32* %j
  %tmp11 = icmp slt i32 %j10, 3
  br i1 %tmp11, label %while_body2, label %merge

while_body2:                                             ; preds = %while1
  %i3 = load i32, i32* %i
  %j4 = load i32, i32* %j
  %a5 = getelementptr [4 x [3 x i32]], [4 x [3 x i32]]* %a, i32 0, i32 %i3, i32 %j4
  %i6 = load i32, i32* %i
  %j7 = load i32, i32* %j
  %tmp = add i32 %i6, %j7
  store i32 %tmp, i32* %a5
  %j8 = load i32, i32* %j
  %tmp9 = add i32 %j8, 1
  store i32 %tmp9, i32* %j
  br label %while1

merge:                                                   ; preds = %while1
  %i12 = load i32, i32* %i
  %tmp13 = add i32 %i12, 1
  store i32 %tmp13, i32* %i
  br label %while

merge16:                                                 ; preds = %while
  store i32 0, i32* %i
  br label %while17

while17:                                                 ; preds = %merge31, %merge16
  %i34 = load i32, i32* %i
  %tmp35 = icmp slt i32 %i34, 3
  br i1 %tmp35, label %while_body18, label %merge36

while_body18:                                            ; preds = %while17
  store i32 0, i32* %j
  br label %while19

while19:                                                 ; preds = %while_body20, %while_body18
  %j29 = load i32, i32* %j
  %tmp30 = icmp slt i32 %j29, 4
  br i1 %tmp30, label %while_body20, label %merge31
```

```

while_body20:                                     ; preds = %while19
  %i21 = load i32, i32* %i
  %j22 = load i32, i32* %j
  %b23 = getelementptr [3 x [4 x i32]], [3 x [4 x i32]]* %b, i32 0, i32 %i21, i32 %j22
  %i24 = load i32, i32* %i
  %j25 = load i32, i32* %j
  %tmp26 = add i32 %i24, %j25
  store i32 %tmp26, i32* %b23
  %j27 = load i32, i32* %j
  %tmp28 = add i32 %j27, 1
  store i32 %tmp28, i32* %j
  br label %while19

merge31:                                          ; preds = %while19
  %i32 = load i32, i32* %i
  %tmp33 = add i32 %i32, 1
  store i32 %tmp33, i32* %i
  br label %while17

merge36:                                          ; preds = %while17
  %c37 = getelementptr inbounds [4 x [4 x i32]], [4 x [4 x i32]]* %c, i32 0, i32 0, i32 0
  %b38 = getelementptr inbounds [3 x [4 x i32]], [3 x [4 x i32]]* %b, i32 0, i32 0, i32 0
  %a39 = getelementptr inbounds [4 x [3 x i32]], [4 x [3 x i32]]* %a, i32 0, i32 0, i32 0
  call void @mult_2D_int(i32* %a39, i32* %b38, i32* %c37, i32 4, i32 3, i32 3, i32 4)
  %c40 = getelementptr [4 x [4 x i32]], [4 x [4 x i32]]* %c, i32 0, i32 3, i32 3
  %c41 = load i32, i32* %c40
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt, i32 0, i32 0), i32 %c41)
  ret i32 0
}

define void @mult_2D_int(i32* %x, i32* %y, i32* %output, i32 %h1, i32 %w1, i32 %h2, i32 %w2) {
entry:
  %x1 = alloca i32*
  store i32* %x, i32** %x1
  %y2 = alloca i32*
  store i32* %y, i32** %y2
  %output3 = alloca i32*
```

```
  store i32* %output, i32** %output3
  %h14 = alloca i32
  store i32 %h1, i32* %h14
  %w15 = alloca i32
  store i32 %w1, i32* %w15
  %h26 = alloca i32
  store i32 %h2, i32* %h26
  %w27 = alloca i32
  store i32 %w2, i32* %w27
  %i = alloca i32
  %j = alloca i32
  %k = alloca i32
  %l = alloca i32
  %temp_x = alloca i32*
  %temp_y = alloca i32*
  %temp_output = alloca i32*
  %output8 = load i32*, i32** %output3
  store i32* %output8, i32** %temp_output
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge, %entry
  %i21 = load i32, i32* %i
  %h122 = load i32, i32* %h14
  %tmp23 = icmp slt i32 %i21, %h122
  br i1 %tmp23, label %while_body, label %merge24

while_body:                                       ; preds = %while
  store i32 0, i32* %j
  br label %while9

while9:                                           ; preds = %while_body10, %while_body
  %j16 = load i32, i32* %j
  %w217 = load i32, i32* %w27
  %tmp18 = icmp slt i32 %j16, %w217
  br i1 %tmp18, label %while_body10, label %merge

while_body10:                                     ; preds = %while9
  %temp_output11 = load i32*, i32** %temp_output
  store i32 0, i32* %temp_output11
  %temp_output12 = getelementptr inbounds i32*, i32** %temp_output, i32 0
  %temp_output13 = load i32*, i32** %temp_output12
  %temp_output14 = getelementptr inbounds i32, i32* %temp_output13, i32 1
```

```
  store i32* %temp_output14, i32** %temp_output
  %j15 = load i32, i32* %j
  %tmp = add i32 %j15, 1
  store i32 %tmp, i32* %j
  br label %while9

merge:                                            ; preds = %while9
  %i19 = load i32, i32* %i
  %tmp20 = add i32 %i19, 1
  store i32 %tmp20, i32* %i
  br label %while

merge24:                                          ; preds = %while
  store i32 0, i32* %i
  br label %while25

while25:                                          ; preds = %merge94, %merge24
  %i97 = load i32, i32* %i
  %h198 = load i32, i32* %h14
  %tmp99 = icmp slt i32 %i97, %h198
  br i1 %tmp99, label %while_body26, label %merge100

while_body26:                                     ; preds = %while25
  store i32 0, i32* %j
  br label %while27

while27:                                          ; preds = %merge85, %while_body26
  %j91 = load i32, i32* %j
  %w292 = load i32, i32* %w27
  %tmp93 = icmp slt i32 %j91, %w292
  br i1 %tmp93, label %while_body28, label %merge94

while_body28:                                     ; preds = %while27
  %x29 = load i32*, i32** %x1
  store i32* %x29, i32** %temp_x
  %y30 = load i32*, i32** %y2
  store i32* %y30, i32** %temp_y
  store i32 0, i32* %k
  br label %while31

while31:                                          ; preds = %while_body32, %while_body28
  %k38 = load i32, i32* %k
```

```
  %i39 = load i32, i32* %i
  %w140 = load i32, i32* %w15
  %tmp41 = mul i32 %i39, %w140
  %tmp42 = icmp slt i32 %k38, %tmp41
  br i1 %tmp42, label %while_body32, label %merge43

while_body32:                                     ; preds = %while31
  %temp_x33 = getelementptr inbounds i32*, i32** %temp_x, i32 0
  %temp_x34 = load i32*, i32** %temp_x33
  %temp_x35 = getelementptr inbounds i32, i32* %temp_x34, i32 1
  store i32* %temp_x35, i32** %temp_x
  %k36 = load i32, i32* %k
  %tmp37 = add i32 %k36, 1
  store i32 %tmp37, i32* %k
  br label %while31

merge43:                                          ; preds = %while31
  store i32 0, i32* %l
  br label %while44

while44:                                          ; preds = %while_body45, %merge43
  %l51 = load i32, i32* %l
  %j52 = load i32, i32* %j
  %tmp53 = icmp slt i32 %l51, %j52
  br i1 %tmp53, label %while_body45, label %merge54

while_body45:                                     ; preds = %while44
  %temp_y46 = getelementptr inbounds i32*, i32** %temp_y, i32 0
  %temp_y47 = load i32*, i32** %temp_y46
  %temp_y48 = getelementptr inbounds i32, i32* %temp_y47, i32 1
  store i32* %temp_y48, i32** %temp_y
  %l49 = load i32, i32* %l
  %tmp50 = add i32 %l49, 1
  store i32 %tmp50, i32* %l
  br label %while44

merge54:                                          ; preds = %while44
  store i32 0, i32* %k
  br label %while55

while55:                                          ; preds = %merge79, %merge54
  %k82 = load i32, i32* %k
  %w183 = load i32, i32* %w15
```

```
  %tmp84 = icmp slt i32 %k82, %w183
  br i1 %tmp84, label %while_body56, label %merge85

while_body56:                                          ; preds = %
    while55
  %output57 = load i32*, i32** %output3
  %output58 = load i32*, i32** %output3
  %output59 = load i32, i32* %output58
  %temp_x60 = load i32*, i32** %temp_x
  %temp_x61 = load i32, i32* %temp_x60
  %temp_y62 = load i32*, i32** %temp_y
  %temp_y63 = load i32, i32* %temp_y62
  %tmp64 = mul i32 %temp_x61, %temp_y63
  %tmp65 = add i32 %output59, %tmp64
  store i32 %tmp65, i32* %output57
  %temp_x66 = getelementptr inbounds i32*, i32** %temp_x, i32 0
  %temp_x67 = load i32*, i32** %temp_x66
  %temp_x68 = getelementptr inbounds i32, i32* %temp_x67, i32 1
  store i32* %temp_x68, i32** %temp_x
  store i32 0, i32* %l
  br label %while69

while69:                                               ; preds = %
    while_body70, %while_body56
  %l76 = load i32, i32* %l
  %w277 = load i32, i32* %w27
  %tmp78 = icmp slt i32 %l76, %w277
  br i1 %tmp78, label %while_body70, label %merge79

while_body70:                                          ; preds = %
    while69
  %temp_y71 = getelementptr inbounds i32*, i32** %temp_y, i32 0
  %temp_y72 = load i32*, i32** %temp_y71
  %temp_y73 = getelementptr inbounds i32, i32* %temp_y72, i32 1
  store i32* %temp_y73, i32** %temp_y
  %l74 = load i32, i32* %l
  %tmp75 = add i32 %l74, 1
  store i32 %tmp75, i32* %l
  br label %while69

merge79:                                               ; preds = %
    while69
  %k80 = load i32, i32* %k
  %tmp81 = add i32 %k80, 1
  store i32 %tmp81, i32* %k
  br label %while55

merge85:                                               ; preds = %
    while55
```

```
  %output86 = getelementptr inbounds i32*, i32** %output3, i32 0
  %output87 = load i32*, i32** %output86
  %output88 = getelementptr inbounds i32, i32* %output87, i32 1
  store i32* %output88, i32** %output3
  %j89 = load i32, i32* %j
  %tmp90 = add i32 %j89, 1
  store i32 %tmp90, i32* %j
  br label %while27

merge94:                                          ; preds = %while27
  %i95 = load i32, i32* %i
  %tmp96 = add i32 %i95, 1
  store i32 %tmp96, i32* %i
  br label %while25

merge100:                                         ; preds = %while25
  ret void
}
```

6.3 Test Suite and Automation

The directory, test, contains all of our tests and test scripts. Within test, there are directories for compiler, parser, scanner, and compiler_fail tests. Our testing automation program can be invoked separately with the test scripts corresponding to each of these directories. Calling any of these test scripts, such as ./compiler_test.sh runs each file that ends with ".darn" and then compares it to its corresponding ".out" file. The "test" files compare the output of the execution of ".darn" file with the expected output in the ".out" file. If the expected output matches the actual output, "Success" gets printed. For fail tests within compiler_fail, if the expected error matches the actual error, "Success" gets printed.

Lastly, we also employ continuous integration with Travis CI setup that automatically checks and runs all our test cases whenever a commit is pushed or a pull-request is opened on our repository.

6.4 Test Cases

The directory, test, contains all of our tests and test scripts. We tried to add as many tests as possible, including fail tests (to check things that should fail in DARN), to create consistent and all-encompassing test cases.

7 Lessons Learned

7.1 Anthony Kim

Working in groups is never an easy task. I learned how to delegate work and utilize the power of pair programming. This was an invaluable experience in working with fellow students on a semester long project where there were multiple complications and setbacks, but we worked together and communicated well to finish the project.

7.2 Daisy Chaussee

Besides learning all the details of creating a compiler and the architecture of it, from the scanner and parser to semantic analysis and code generation, I learned to be realistic in setting goals and marking milestones. As Manager of this project, I was reminded of the process of goal-setting and learned to be less idealistic about making progress. Progress takes time and dedication. Remember, all goals should be SMART: Specific, Measurable, Attainable, Realistic, and Time-bound. Staying on track with a timeline and including small, incremental improvements in addition to the large "Hello World" milestones will help determine a successful project.

7.3 Ignacio Torras

The thing I learned the most is how important it is to stay up to date and learn all the pieces of the project. To be able to contribute to the team you not only have to do your role but also understand the overview of the code. It's important to understand how the program moves from the scanner to the codegen. Once you fully understand that you can actively contribute and add new features from beginning to end. It is also crucial to have a team that you can work with because this is too much for one or two people. To keep the project moving forward throughout the semester everyone has to learn and contribute so that no one person is stuck doing everything.

7.4 Rafael Takasu

Working in groups is always a challenge, and it can get even more challenging once the group is dealing with a technology that no one is really familiar with. A lesson learned for me is to make sure that when dealing with challenging new concepts, it is important to take some time to try to understand the basics before trying to implement things. Having the basics of the concepts down, dividing up the tasks becomes much easier, and the project flows in a more natural way.

7.5 Advice to Future Groups

The best piece of advice we have for future groups is to master OCaml early on. It's not enough to simply memorize it for the first homework assignment. Additionally, solidifying and clarifying the features of your language and its "identity" will prove valuable throughout the design and development processes. Try to think of a problem you want your language to solve or a unique feature it can implement.

8 Appendix

Attached is a complete code listing of our DARN translator. The formatting and style may be a bit off due to LaTeX's incompatability with OCaml code files.

8.1 preprocess.ml

```

let process_files filename1 filename2 =
  let read_all_lines file_name =
  let in_channel = open_in file_name in
  let rec read_recursive lines =
  try
    Scanf.fscanf in_channel "%[^\r\n]\n" (fun x ->
    read_recursive (x :: lines))
  with
    End_of_file ->
      lines in
  let lines = read_recursive [] in
  let _ = close_in_noerr in_channel in
  List.rev (lines) in

  let concat = List.fold_left (fun a x -> a ^ x) ""
in " \n " ^ concat (read_all_lines filename1) ^ " \n " ^ concat
    (read_all_lines filename2)
```

8.2 scanner.mll

```
{

  open Parser
```

```
  let unescape s =
      Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)

}

let whitespace = [' ' '\t' '\r' '\n']
let esc = '\\' ['\\' ''' '"' 'n' 'r' 't']
let esc_ch = ''' (esc) '''
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let digits = ['0'-'9']
let alphabet = ['a'-'z' 'A'-'Z']
let alphanumund = alphabet | digits | '_'
let integer = digits+
let decimal = ['.']
let float = digits* decimal digits+ | digits+ decimal digits*
let string = '"' ( (ascii | esc)* as s ) '"'
let char = ''' ( ascii | digits ) '''
let id = alphabet alphanumund*

rule token = parse

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

(* Comment *)
| "/*"  { comment lexbuf }

(* Punctuation *)
|    '(' { LPAREN }   |    ')' { RPAREN }   |    '[' { LBRACK }
|    ']' { RBRACK }   |    '{' { LCURLY }   |    '}' { RCURLY }
|    ';' { SEMI }     |    ',' { COMMA }    |    ':' { COLON }

(* Arithmetic *)
|    '+' { PLUS }     |    '-' { MINUS }
|    '*' { TIMES }    |    '/' { DIVIDE }

(* Assignment *)
|    '='      { ASSIGN }

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

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

(* Reference Dereference *)
| '%' { PERCENT } | '#' { OCTOTHORP }
```

```

(* Conditional and Loops *)
|   "if"    { IF }  |   "else"  { ELSE }    |   "elif"  { ELIF }
|   "for"   { FOR } |   "while" { WHILE }

(* Return *)
|   "return"    { RETURN }

(* Types *)
|   "true"  { TRUE }    |   "false" { FALSE }   |   "char"  {
    CHAR }
|   "int"   { INT }     |   "float" { FLOAT }   |   "bool"  {
    BOOL }
|   "void"   { VOID }

(* Matrices *)
|  "len"  { LEN }   |   "height" { HEIGHT } | "width" { WIDTH }

(* Literal *)
|   ['0'-'9']+ as lxm   { INTLITERAL(int_of_string lxm) }
|   float as lxm        { FLOATLITERAL(float_of_string lxm) }
|   string              { STRINGLITERAL(unescape s) }
|   char    as lxm { CHARLITERAL(String.get lxm 1) }
|   id      as lxm { ID(lxm) }

(* EOF *)
|   eof { EOF }

and comment = parse
"*/"    { token lexbuf }
|   _   { comment lexbuf }
```

8.3 parser.mly

```
%{ open Ast %}

/* Punctuation */
%token SEMI LPAREN RPAREN LCURLY RCURLY LBRACK RBRACK COMMA
    COLON

/* Arithmetic */
%token PLUS MINUS TIMES DIVIDE

/* Boolean Value */
%token TRUE FALSE

/* Conditional Operators */
%token IF ELSE ELIF FOR WHILE

```

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

/* Logical Operators */
%token AND OR NOT

/* Matrices */
%token LEN HEIGHT WIDTH

/* Assignment */
%token ASSIGN

/* Variable Type */
%token BOOL INT FLOAT CHAR VOID

/* Functional Keywords */
%token RETURN

/* Reference and Dereference */
%token OCTOTHORP PERCENT

/* End Of File */
%token EOF

/* Literals */
%token <int> INTLITERAL
%token <float> FLOATLITERAL
%token <string> STRINGLITERAL
%token <char> CHARLITERAL

%token <string> ID

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

%start program
%type <Ast.program> program

%%
```

```
program:
    decls EOF { $1 }

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

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

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

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

typ:
  INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | FLOAT { Float }
  | CHAR { Char }
  | matrix1D_typ { $1 }
  | matrix2D_typ { $1 }
  | matrix1D_pointer_typ { $1 }
  | matrix2D_pointer_typ { $1 }

matrix1D_typ:
    typ LBRACK INTLITERAL RBRACK %prec NOLBRACK  { Matrix1DType($1, $3) }

matrix2D_typ:
    typ LBRACK INTLITERAL RBRACK LBRACK INTLITERAL RBRACK  { Matrix2DType($1, $3, $6) }

matrix1D_pointer_typ:
  typ LBRACK RBRACK %prec NOLBRACK { Matrix1DPointer($1)}

matrix2D_pointer_typ:
```

```
  typ LBRACK RBRACK LBRACK RBRACK { Matrix2DPointer($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 }
    | LCURLY stmt_list RCURLY { 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) }
    /* add conditional statements and return */

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

expr:
    arith_ops                                          { $1 }
    | bool_ops                                         { $1 }
    | primitives                                       { $1 }
    | expr ASSIGN expr                                 { Assign($1, $3)  }
    | LPAREN expr RPAREN                               { $2 }
    | CHARLITERAL                                      { CharLiteral($1) }
    | STRINGLITERAL                                    { StringLiteral($1) }
    | TRUE                                             { BoolLiteral(true) }
    | FALSE                                            { BoolLiteral(false) }
    | ID LPAREN actuals_opt RPAREN                     { Call($1, $3)}
    | LBRACK matrix_literal RBRACK                     { MatrixLiteral(List.rev $2) }
```

```
    | ID LBRACK expr  RBRACK %prec NOLBRACK          { Matrix1DAccess($1, $3)}
    | ID LBRACK expr  RBRACK LBRACK expr  RBRACK     { Matrix2DAccess($1, $3, $6)}
    | LEN LPAREN ID RPAREN                           { Len($3) }
    | HEIGHT LPAREN ID RPAREN                        { Height($3) }
    | WIDTH LPAREN ID RPAREN                         { Width($3) }
    | ID                                             { Id($1)}
    | PERCENT ID                                     { Matrix1DReference($2)}
    | PERCENT PERCENT ID                             { Matrix2DReference($3)}
    | OCTOTHORP ID                                   { Dereference($2)}
    | PLUS PLUS ID                                   { PointerIncrement($3) }

primitives:
    INTLITERAL                                     { IntLiteral($1)   }
  | FLOATLITERAL                                   { FloatLiteral($1) }

matrix_literal:
    primitives                          { [$1] }
  | matrix_literal COMMA primitives { $3 :: $1 }

arith_ops:
    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)  }

bool_ops:
    expr LT expr        {Binop($1, Less, $3) }
    | expr GT expr      {Binop($1, Greater, $3) }
    | expr LEQ expr     {Binop($1, Leq, $3)  }
    | expr GEQ expr     {Binop($1, Geq, $3)  }
    | expr NEQ expr     {Binop($1, Neq, $3)  }
    | expr EQ expr      {Binop($1, Eq, $3)   }
    | expr OR expr      {Binop($1, Or, $3)   }
    | expr AND expr     {Binop($1, And, $3)  }
    | NOT expr          {Unop(Not, $2)    }
    | MINUS expr %prec NEG { Unop(Neg, $2) }

actuals_opt:
    /* nothing */ { [] }
```

```
  | actuals_list  { List.rev $1 }

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

8.4 ast.ml

```
type op = Add | Sub | Mul | Div | Less | Greater
          | Leq | Geq | Or | And | Eq | Neq

type uop = Not | Neg

type typ =
    Int
    | Bool
    | Void
    | Float
    | Char
    | String
    | Matrix1DType of typ * int
    | Matrix2DType of typ * int * int
    | Matrix1DPointer of typ
    | Matrix2DPointer of typ

type bind = typ * string

type expr =
    IntLiteral of int
    | FloatLiteral of float
    | BoolLiteral of bool
    | CharLiteral of char
    | StringLiteral of string
    | Id of string
    | Binop of expr * op * expr
    | Unop of uop * expr
    | Assign of expr * expr
    | PointerIncrement of string
    | MatrixLiteral of expr list
    | Matrix1DAccess of string * expr
    | Matrix2DAccess of string * expr * expr
    | Len of string
    | Height of string
    | Width of string
    | Call of string * expr list
    | Noexpr
    | Matrix1DReference of string
    | Matrix2DReference of string
    | Dereference of string
```

```

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 Printer *)
let string_of_bop = function
      Add -> "+"
      | Sub -> "-"
      | Mul -> "*"
      | Div -> "/"
      | Less -> "<"
      | Greater -> ">"
      | Leq -> "<="
      | Geq -> ">="
      | Or -> "||"
      | And -> "&&"
      | Eq -> "=="
      | Neq -> "!="

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

let string_of_matrix m =
  let rec string_of_matrix_lit = function
      [] -> "]"
    | [hd] -> (match hd with
                  IntLiteral(i) -> string_of_int i
                | FloatLiteral(i) -> string_of_float i
                | BoolLiteral(i) -> string_of_bool i
                | Id(s) -> s
                | _ -> raise( Failure("Illegal expression in
```

```
    matrix literal")  )) ^ string_of_matrix_lit []
    | hd::tl -> (match hd with
                    IntLiteral(i) -> string_of_int i ^ ", "
                  | FloatLiteral(i) -> string_of_float i ^ ", "
                  | BoolLiteral(i) -> string_of_bool i ^ ", "
                  | Id(s) -> s
                  | _ -> raise( Failure("Illegal expression in
    matrix literal")  )) ^ string_of_matrix_lit tl
  in
  "[" ^ string_of_matrix_lit m

let rec string_of_expr = function
    IntLiteral(i) -> string_of_int i
    | FloatLiteral(i) -> string_of_float i
    | BoolLiteral(i) -> string_of_bool i
    | CharLiteral(i) ->  String.make 1 i
    | StringLiteral(i) -> i
    | Id(i) -> i
    | Unop(uop, r1) -> (string_of_uop uop) ^ string_of_expr r1
    | Binop(r1, bop, r2) -> string_of_expr r1 ^ " " ^ (
    string_of_bop
    bop) ^ " " ^ (string_of_expr r2)
    | PointerIncrement(s) -> "++" ^ s
    | Assign(r1, r2) -> (string_of_expr r1) ^ " =  " ^ (
    string_of_expr r2)
    | MatrixLiteral(m) -> string_of_matrix m
    | Matrix1DAccess(s, r1) -> s ^ "[" ^ (string_of_expr r1) ^
    "]"
    | Matrix2DAccess(s, r1, r2) -> s ^ "[" ^ (string_of_expr r1)
     ^ "]" ^ "[" ^ (string_of_expr r2) ^ "]"
    | Len(s) -> "len(" ^ s ^ ")"
    | Height(s) -> "height(" ^ s ^ ")"
    | Width(s) -> "width(" ^ s ^ ")"
    | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el)
    ^ ")"
    | Noexpr -> ""
    | Matrix1DReference(s) -> "%" ^ s
    | Matrix2DReference(s) -> "%%" ^ s
    | Dereference(s) -> "#" ^ s

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
     "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
    string_of_stmt s
```

```
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^
    " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
    string_of_stmt s

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | Float -> "float"
  | Char -> "char"
  | String -> "string"
  | Matrix1DType(t, i1) -> string_of_typ t ^ "[" ^ string_of_int
     i1 ^ "]"
  | Matrix2DType(t, i1, i2) -> string_of_typ t ^ "[" ^
    string_of_int i1 ^ "]" ^ "[" ^ string_of_int i2 ^ "]"
  | Matrix1DPointer(t) -> string_of_typ t ^ "[]"
  | Matrix2DPointer(t) -> string_of_typ t ^ "[][]"

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

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

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

8.5 semant.ml

```
open Ast

module StringMap = Map.Make(String)

let check (globals, functions) =

(* From MicroC *)

(* 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)
  in

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

(* Raise an exception of the given rvalue type cannot be
    assigned to
     the given lvalue type *)
  let check_assign lvaluet rvaluet err =
     if lvaluet = rvaluet then lvaluet else raise err
  in


List.iter (check_not_void (fun n -> "Illegal void global " ^ n))
     globals;

report_duplicate (fun n -> "Duplicate global " ^ n) (List.map
    snd globals);


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);


let built_in_decls = StringMap.add "print"
  { typ = Void; fname = "print"; formals = [(Int, "x")];
    locals = []; body = [] } (StringMap.add "printf"
  { typ = Void; fname = "printf"; formals = [(Float, "x")];
    locals = []; body = [] } (StringMap.add "prints"
  { typ = Void; fname = "prints"; formals = [(String, "x")];
    locals = []; body = [] } (StringMap.singleton "printb"
  { typ = Void; fname = "printb"; formals = [(Bool, "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))
in

let _ = function_decl "main" in

(* A function that is used to check each 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);

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

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

    let matrix_access_type = function
      Matrix1DType(t, _) -> t
      | Matrix2DType(t, _, _) -> t
      | _ -> raise (Failure ("illegal matrix access") )
    in
```

```
    let check_pointer_type = function
        Matrix1DPointer(t) -> Matrix1DPointer(t)
      | Matrix2DPointer(t) -> Matrix2DPointer(t)
      | _ -> raise ( Failure ("cannot increment a non-pointer type") )
    in

    let check_matrix1D_pointer_type = function
      Matrix1DType(p, _) -> Matrix1DPointer(p)
      | _ -> raise ( Failure ("cannont reference non-1Dmatrix pointer type"))
    in

    let check_matrix2D_pointer_type = function
      Matrix2DType(p, _, _) -> Matrix2DPointer(p)
      | _ -> raise ( Failure ("cannont reference non-2Dmatrix pointer type"))
    in

    let pointer_type = function
    | Matrix1DPointer(t) -> t
    | Matrix2DPointer(t) -> t
    | _ -> raise ( Failure ("cannot dereference a non-pointer type")) in

    let matrix_type s = match (List.hd s) with
    | IntLiteral _ -> Matrix1DType(Int, List.length s)
    | FloatLiteral _ -> Matrix1DType(Float, List.length s)
    | BoolLiteral _ -> Matrix1DType(Bool, List.length s)
    | _ -> raise ( Failure ("Cannot instantiate a matrix of that type")) in

    let rec check_all_matrix_literal m ty idx =
      let length = List.length m in
        match (ty, List.nth m idx) with
      (Matrix1DType(Int, _), IntLiteral _) -> if idx == length - 1 then Matrix1DType(Int, length) else check_all_matrix_literal m (Matrix1DType(Int, length)) (succ idx)
    | (Matrix1DType(Float, _), FloatLiteral _) -> if idx == length - 1 then Matrix1DType(Float, length) else check_all_matrix_literal m (Matrix1DType(Float, length)) (succ idx)
    | (Matrix1DType(Bool, _), BoolLiteral _) -> if idx == length - 1 then Matrix1DType(Bool, length) else check_all_matrix_literal m (Matrix1DType(Bool, length)) (succ idx)
    | _ -> raise (Failure ("illegal matrix literal"))
```

```
  in

  let rec expr = function
       IntLiteral _ -> Int
     | FloatLiteral _ -> Float
     | BoolLiteral _ -> Bool
     | CharLiteral _ -> Char
     | StringLiteral _ -> String
     | Id s -> type_of_identifier s
     | PointerIncrement(s) -> check_pointer_type (type_of_identifier s)
     | MatrixLiteral s -> check_all_matrix_literal s (matrix_type s) 0
     | Matrix1DAccess(s, e1) -> let _ = (match (expr e1) with
                                            Int -> Int
                                          | _ -> raise (Failure ("attempting to access with a non-integer type"))) in
                                matrix_access_type (type_of_identifier s)
     | Matrix2DAccess(s, e1, e2) -> let _ = (match (expr e1) with
                                            Int -> Int
                                          | _ -> raise (Failure ("attempting to access with a non-integer type")))
                                and _ = (match (expr e2) with
                                            Int -> Int
                                          | _ -> raise (Failure ("attempting to access with a non-integer type"))) in
                                matrix_access_type (type_of_identifier s)
     | Len(s) -> (match (type_of_identifier s) with
                    Matrix1DType(_, _) -> Int
                  | _ -> raise(Failure ("cannot get the length of non-1d-matrix")))
     | Height(s) -> (match (type_of_identifier s) with
                    Matrix2DType(_, _, _) -> Int
                  | _ -> raise(Failure ("cannot get the height of non-2d-matrix")))
     | Width(s) -> (match (type_of_identifier s) with
                    Matrix2DType(_, _, _) -> Int
                  | _ -> raise(Failure ("cannot get the width of non-2d-matrix")))
     | Dereference(s) -> pointer_type (type_of_identifier s)
     | Matrix1DReference(s) -> check_matrix1D_pointer_type( type_of_identifier s )
     | Matrix2DReference(s) -> check_matrix2D_pointer_type( type_of_identifier s )
     | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
```

```
(match op with
        Add | Sub | Mul | Div when t1 = Int && t2 = Int -> Int
     |   Add | Sub | Mul | Div when t1 = Float && t2 = Float
  -> Float
    | Eq | 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_bop 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 = Int -> Int
| Neg when t = Float -> Float
| Not when t = Bool -> Bool
      | _ -> raise (Failure ("Illegal unary operator " ^
  string_of_uop op ^
        string_of_typ t ^ " in " ^ string_of_expr ex)))
    | Noexpr -> Void
    | Assign(e1, e2) as ex -> let lt = ( match e1 with
                                        | Matrix1DAccess(s,
  _) -> (match (type_of_identifier s) with

       Matrix1DType(t, _) -> (match t with

                                    Int -> Int

                                  | Float -> Float

                                  | _ -> raise ( Failure ("
  illegal matrix of matrices") )

                                )

       | _ -> raise ( Failure ("cannot access a primitive") )

    )
                                        | Matrix2DAccess(s,
  _, _) -> (match (type_of_identifier s) with

       Matrix2DType(t, _, _) -> (match t with

                                    Int -> Int

```

```
                                        | Float -> Float
                                        | Matrix1DType(p, l) -> Matrix1DType(p, l)
                                        | _ -> raise ( Failure ("illegal matrix of matrices") )
                                      )
          | _ -> raise ( Failure ("cannot access a primitive") )
      )
                                                | _ -> expr e1)
                                 and rt = expr e2 in
      check_assign lt rt (Failure ("Illegal assignment " ^ string_of_typ lt ^
            " = " ^ string_of_typ rt ^ " 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
  in

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

  (* Verify or throw 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))

      | 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
```

8.6 exceptions.ml

```
exception UnsupportedMatrixType

exception IllegalAssignment

exception IllegalPointerType

exception MatrixOutOfBounds

exception IllegalUnop

exception WrongReturn
```

8.7 codegen.ml

```
(* 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 Exceptions

module StringMap = Map.Make(String)

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

  let ltype_of_typ = function
      A.Int -> i32_t
    | A.Bool -> i1_t
    | A.Float -> float_t
    | A.Char   -> i8_t
    | A.String -> pointer_t i8_t
    | A.Void -> void_t
    | A.Matrix1DType(typ, size) -> (match typ with
                                        A.Int -> array_t i32_t size
                                      | A.Float -> array_t float_t size
                                      | A.Bool -> array_t i1_t size
                                      | A.Matrix2DType(typ, size1, size2) -> (match typ with
                      A.Int -> array_t (array_t i32_t size2) size1
                    | A.Float -> array_t (array_t float_t size2) size1
                    | _ -> raise ( UnsupportedMatrixType )
                  )
                                      | _ -> raise (
```

```
  UnsupportedMatrixType )
                                            )
    | A.Matrix2DType(typ, size1, size2) -> (match typ with
                                              A.Int -> array_t (
  array_t i32_t size2) size1
                                            | A.Float -> array_t (
  array_t float_t size2) size1
                                            | A.Matrix1DType(typ1,
   size3) -> (match typ1 with

          | A.Int -> array_t (array_t (array_t i32_t size3)
  size2) size1

          | A.Float -> array_t (array_t (array_t float_t size3
  ) size2) size1

          | _ -> raise (UnsupportedMatrixType)

         )
                                            | _ -> raise (
  UnsupportedMatrixType )
                                            )
    | A.Matrix1DPointer(t) -> (match t with
                                  A.Int -> pointer_t i32_t
                                | A.Float -> pointer_t float_t
                                | _ -> raise (
  IllegalPointerType))
    | A.Matrix2DPointer(t) -> (match t with
                                  A.Int -> pointer_t i32_t
                                | A.Float -> pointer_t float_t
                                | _ -> raise (
  IllegalPointerType))

in

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

(* Declare printf(), which the print built-in function will
  call *)
let printf_t = L.var_arg_function_type i32_t [| L.pointer_type
   i8_t |] in
let printf_func = L.declare_function "printf" printf_t
```

```
    the_module in

  (* Define each function (arguments and return type) so we can
    call it *)
  let function_decls =
    let function_decl m fdecl =
      let name = fdecl.A.fname
      and formal_types =
  Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.
    formals)
      in let ftype = L.function_type (ltype_of_typ fdecl.A.typ)
    formal_types in
      StringMap.add name (L.define_function name ftype
    the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.A.fname
    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
    and float_format_str = L.build_global_stringptr "%f" "fmt"
    builder in
    (* add float... and float_format_str = L.
    build_global_stringptr "%f\n" "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.build_alloca (ltype_of_typ t) n builder in
  ignore (L.build_store p local builder);
  StringMap.add n local m in

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

      let formals = List.fold_left2 add_formal StringMap.empty
    fdecl.A.formals
          (Array.to_list (L.params the_function)) in
```

```
      List.fold_left add_local formals fdecl.A.locals 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

    let check_function =
        List.fold_left (fun m (t, n) -> StringMap.add n t m)
        StringMap.empty (globals @ fdecl.A.formals @ fdecl.A.locals)
    in

    let type_of_identifier s =
      let symbols = check_function in
      StringMap.find s symbols
    in

    let build_1D_matrix_argument s builder =
      L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int i32_t 0 |] s builder
    in

    let build_2D_matrix_argument s builder =
      L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0; L.const_int i32_t 0; L.const_int i32_t 0 |] s builder
    in


    let build_1D_matrix_access s i1 i2 builder isAssign =
      if isAssign
        then L.build_gep (lookup s) [| i1; i2 |] s builder
      else
         L.build_load (L.build_gep (lookup s) [| i1; i2 |] s builder) s builder
    in

    let build_2D_matrix_access s i1 i2 i3 builder isAssign =
      if isAssign
        then L.build_gep (lookup s) [| i1; i2; i3 |] s builder
      else
         L.build_load (L.build_gep (lookup s) [| i1; i2; i3 |] s builder) s builder
    in

    let build_pointer_dereference s builder isAssign =
      if isAssign
        then L.build_load (lookup s) s builder
```

```
      else
        L.build_load (L.build_load (lookup s) s builder) s builder
    in

    let build_pointer_increment s builder isAssign =
      if isAssign
        then L.build_load (L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 1 |] s builder) s builder
      else
        L.build_in_bounds_gep (L.build_load (L.build_in_bounds_gep (lookup s) [| L.const_int i32_t 0 |] s builder) s builder) [| L.const_int i32_t 1 |] s builder
    in

    let rec matrix_expression e =
       match e with
       | A.IntLiteral i -> i
       | A.Binop (e1, op, e2) -> (match op with
              A.Add     -> (matrix_expression e1) + (matrix_expression e2)
            | A.Sub     -> (matrix_expression e1) - (matrix_expression e2)
            | A.Mul     -> (matrix_expression e1) * (matrix_expression e2)
            | A.Div     -> (matrix_expression e1) / (matrix_expression e2)
            | _ -> 0)
       | _ -> 0
    in

    let find_matrix_type matrix =
      match (List.hd matrix) with
        A.IntLiteral _ -> ltype_of_typ (A.Int)
      | A.FloatLiteral _ -> ltype_of_typ (A.Float)
      | A.BoolLiteral _ -> ltype_of_typ (A.Bool)
      | _ -> raise (UnsupportedMatrixType) in

    (* Construct code for an expression; return its value *)
    let rec expr builder = function
    A.IntLiteral i -> L.const_int i32_t i
      | A.FloatLiteral f -> L.const_float float_t f
      | A.BoolLiteral b -> L.const_int i1_t (if b then 1 else 0)
      | A.CharLiteral c -> L.const_int i8_t (Char.code c)
      | A.StringLiteral s -> L.const_string context s
      | A.Noexpr -> L.const_int i32_t 0
      | A.Id s -> L.build_load (lookup s) s builder
      | A.MatrixLiteral s -> L.const_array (find_matrix_type s) (Array.of_list (List.map (expr builder) s))
```

```
    | A.Matrix1DReference (s) -> build_1D_matrix_argument s builder
    | A.Matrix2DReference (s) -> build_2D_matrix_argument s builder
    | A.Len s -> (match (type_of_identifier s) with A.Matrix1DType(_, l) -> L.const_int i32_t l
                                                                | _ -> L.const_int i32_t 0 )
    | A.Height s -> (match (type_of_identifier s) with A.Matrix2DType(_, l, _) -> L.const_int i32_t l
                                                                | _ -> L.const_int i32_t 0 )
    | A.Width s -> (match (type_of_identifier s) with A.Matrix2DType(_, _, l) -> L.const_int i32_t l
                                                                | _ -> L.const_int i32_t 0 )
    | A.Matrix1DAccess (s, e1) -> let i1 = expr builder e1 in (match (type_of_identifier s) with
                                                               A.Matrix1DType(_, l) -> (
                                                                  if (matrix_expression e1) >= l then raise(MatrixOutOfBounds)
                                                                  else build_1D_matrix_access s (L.const_int i32_t 0) i1 builder false)
                                                                | _ -> build_1D_matrix_access s (L.const_int i32_t 0) i1 builder false )
    | A.Matrix2DAccess (s, e1, e2) -> let i1 = expr builder e1 and i2 = expr builder e2 in (match (type_of_identifier s) with
                                                               A.Matrix2DType(_, l1, l2) -> (
                                                                  if (matrix_expression e1) >= l1 then raise(MatrixOutOfBounds)
                                                                  else if (matrix_expression e2) >= l2 then raise(MatrixOutOfBounds)
                                                                  else build_2D_matrix_access s (L.const_int i32_t 0) i1 i2 builder false)
                                                                | _ -> build_2D_matrix_access s (L.const_int i32_t 0) i1 i2 builder false )
    | A.PointerIncrement (s) ->  build_pointer_increment s builder false
    | A.Dereference (s) -> build_pointer_dereference s builder false
    | A.Binop (e1, op, e2) ->
      let e1' = expr builder e1
```

```
      and e2' = expr builder e2 in
        let float_bop operator =
          (match operator with
            A.Add     -> L.build_fadd
          | A.Sub     -> L.build_fsub
          | A.Mul    -> L.build_fmul
          | A.Div     -> L.build_fdiv
          | A.And     -> L.build_and
          | A.Or      -> L.build_or
          | 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
          ) e1' e2' "tmp" builder
        in

        let int_bop operator =
          (match operator with
            A.Add     -> L.build_add
          | A.Sub     -> L.build_sub
          | A.Mul    -> L.build_mul
                | A.Div     -> L.build_sdiv
          | A.And     -> L.build_and
          | A.Or      -> L.build_or
          | 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
          ) e1' e2' "tmp" builder
        in

      let string_of_e1'_llvalue = L.string_of_llvalue e1'
      and string_of_e2'_llvalue = L.string_of_llvalue e2' in

      let space = Str.regexp " " in

      let list_of_e1'_llvalue = Str.split space string_of_e1'
  _llvalue
      and list_of_e2'_llvalue = Str.split space string_of_e2'
  _llvalue in

      let i32_re = Str.regexp "i32\\|i32*\\|i8\\|i8*\\|i1\\|i1
  *"
      and float_re = Str.regexp "double\\|double*" in

```

```
      let rec match_string regexp str_list i =
       let length = List.length str_list in
       match (Str.string_match regexp (List.nth str_list i) 0) with
          true -> true
       | false -> if (i > length - 2) then false else match_string regexp str_list (succ i) in

      let get_type llvalue =
          match (match_string i32_re llvalue 0) with
            true  -> "int"
          | false -> (match (match_string float_re llvalue 0) with
                       true -> "float"
                     | false -> "") in

      let e1'_type = get_type list_of_e1'_llvalue
      and e2'_type = get_type list_of_e2'_llvalue in

      let build_ops_with_types typ1 typ2 =
        match (typ1, typ2) with
          "int", "int" -> int_bop op
        | "float" , "float" -> float_bop op
        | _, _ -> raise(IllegalAssignment)
      in
      build_ops_with_types e1'_type e2'_type
    | A.Unop(op, e) ->
      let e' = expr builder e in

      let float_uops operator =
        match operator with
          A.Neg -> L.build_fneg e' "tmp" builder
        | A.Not -> raise(IllegalUnop)  in

      let int_uops operator =
        match operator with
          A.Neg -> L.build_neg e' "tmp" builder
        | A.Not -> L.build_not e' "tmp" builder in

      let bool_uops operator =
        match operator with
        A.Neg -> L.build_neg e' "tmp" builder
        | A.Not -> L.build_not e' "tmp" builder in

      let string_of_e'_llvalue = L.string_of_llvalue e' in

      let space = Str.regexp " " in

      let list_of_e'_llvalue = Str.split space string_of_e'
```

```
_llvalue in

      let i32_re = Str.regexp "i32\\|i32*"
      and float_re = Str.regexp "double\\|double*"
      and bool_re = Str.regexp "i1\\|i1*" in

      let rec match_string regexp str_list i =
         let length = List.length str_list in
         match (Str.string_match regexp (List.nth str_list i)
0) with
            true -> true
         | false -> if (i > length - 2) then false else
match_string regexp str_list (succ i) in

      let get_type llvalue =
         match (match_string i32_re llvalue 0) with
            true  -> "int"
         | false -> (match (match_string float_re llvalue 0)
with
                        true -> "float"
                      | false -> (match (match_string bool_re
llvalue 0) with
                                    true -> "bool"
                                  | false -> "")) in

      let e'_type = get_type list_of_e'_llvalue  in

      let build_ops_with_type typ =
        match typ with
          "int" -> int_uops op
        | "float" -> float_uops op
        | "bool" -> bool_uops op
        | _ -> raise(IllegalAssignment)
      in

      build_ops_with_type e'_type
    | A.Assign (e1, e2) -> let e1' = (match e1 with
                                            A.Id s -> lookup s
                                          | A.Matrix1DAccess (s,
 e1) -> let i1 = expr builder e1 in (match (
type_of_identifier s) with
                                                          A.
Matrix1DType(_, l) -> (
                                                            if (
matrix_expression e1) >= l then raise(MatrixOutOfBounds)
                                                            else
build_1D_matrix_access s (L.const_int i32_t 0) i1 builder
true)
                                                          | _ ->
```

```
build_1D_matrix_access s (L.const_int i32_t 0) i1 builder true )
                                            | A.Matrix2DAccess (s, e1, e2) -> let i1 = expr builder e1 and i2 = expr builder e2 in (match (type_of_identifier s) with
                                                         A.Matrix2DType(_, l1, l2) -> (
                                                           if (matrix_expression e1) >= l1 then raise(MatrixOutOfBounds)
                                                           else if (matrix_expression e2) >= l2 then raise(MatrixOutOfBounds)
                                                           else build_2D_matrix_access s (L.const_int i32_t 0) i1 i2 builder true)
                                                         | _ -> build_2D_matrix_access s (L.const_int i32_t 0) i1 i2 builder true )
                                            | A.PointerIncrement(s) -> build_pointer_increment s builder true
                                            | A.Dereference(s) -> build_pointer_dereference s builder true
                                            | _ -> raise (IllegalAssignment)
                                            )
                                  and e2' = expr builder e2 in
                             ignore (L.build_store e2' e1' builder); e2'
  | A.Call ("print", [e]) | A.Call ("printb", [e]) ->
L.build_call printf_func [| int_format_str ; (expr builder e) |]
  "printf" builder
  | A.Call ("printf", [e]) ->
L.build_call printf_func [| float_format_str ; (expr builder e) |]
  "printf" builder
  | A.Call ("prints", [e]) -> let get_string = function A.StringLiteral s -> s | _ -> "" in
  let s_ptr = L.build_global_stringptr ((get_string e)) ".str" builder in
L.build_call printf_func [| s_ptr |]
  "printf" builder
  | A.Call (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.A.typ with A.Void -> ""
                                            | _ -> f ^ "_result") in
       L.build_call fdef (Array.of_list actuals) result builder
```

```
    in

    (* 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
  A.Block sl -> List.fold_left stmt builder sl
      | A.Expr e -> ignore (expr builder e); builder
      | A.Return e -> ignore (match fdecl.A.typ with
    A.Void -> L.build_ret_void builder
  | _ -> L.build_ret (expr builder e) builder); builder
      | A.If (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

      | A.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

      | A.For (e1, e2, e3, body) -> stmt builder
      ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (A.Block fdecl.A.body) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.A.typ with
        A.Void -> L.build_ret_void
      | A.Int -> L.build_ret (L.const_int i32_t 0)
      | A.Float -> L.build_ret (L.const_float float_t 0.0)
      | A.Bool -> L.build_ret (L.const_int i1_t 0)
      | A.Char -> L.build_ret (L.const_int i8_t 0)
      | _ -> raise (WrongReturn))
  in

  List.iter build_function_body functions;
  the_module
```

8.8 darn.ml

```
(* ./darn.native -c file.darn standardlib.darn *)

type action = AST | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", AST);  (* Print the AST only *)
            ("-l", LLVM_IR);  (* Generate LLVM, don't check *)
            ("-c", Compile) ] (* Generate, check LLVM IR *)
    else Compile in
    let lexbuf = Lexing.from_string (Preprocess.process_files Sys.argv.(2) Sys.argv.(3) )in
      let ast = Parser.program Scanner.token lexbuf in
      Semant.check ast;
      match action with
          AST -> print_string (Ast.string_of_program ast)
        | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate ast))
        | Compile -> let m = Codegen.translate ast in
          Llvm_analysis.assert_valid_module m;
          print_string (Llvm.string_of_llmodule m)
```

8.9 Makefile

```
TARFILES = Makefile scanner.mll parser.mly ast.ml darn.ml semant.
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

# Easiest way to build: using ocamlbuild, which in turn uses ocamlfind

.PHONY : darn.native

darn.native :
	ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,str -cflags -w,+a-4 \
		darn.native

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf testall.log *.diff darn test_darn scanner.ml parser.ml parser.mli
	rm -rf *.cmx *.cmi *.cmo *.cmx *.o

# More detailed: build using ocamlc/ocamlopt + ocamlfind to locate LLVM
TESTOBJS = parser.cmo scanner.cmo

.PHONY : test
test : darn.native test_parser_scanner

.PHONY : test_parser_scanner

test_parser_scanner : $(TESTOBJS)


OBJS = ast.cmx codegen.cmx parser.cmx scanner.cmx darn.cmx semant.cmx

darn : $(OBJS)
	ocamlfind ocamlopt -linkpkg -package llvm -package llvm.analysis $(OBJS) -o darn

scanner.ml : scanner.mll
	ocamllex scanner.mll

```

```
parser.ml parser.mli : parser.mly
	ocamlyacc parser.mly

%.cmo : %.ml
	ocamlc -c $<

%.cmi : %.mli
	ocamlc -c $<

%.cmx : %.ml
	ocamlfind ocamlopt -c -package llvm $<

#.PHONY : clean
#clean :
#	rm -f darn parser.ml parser.mli scanner.ml *.cmo *.cmi

# Generated by ocamldep *.ml *.mli
calc.cmo: scanner.cmo parser.cmi ast.cmo
calc.cmx: scanner.cmx parser.cmx ast.cmx
parser.cmo: ast.cmo parser.cmi
parser.cmx: ast.cmx parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
semant.cmo: ast.cmo
semant.cmx: ast.cmx
parser.cmi: ast.cmo
```