# XIRTAM

A Convenient Language for Matrices

Lior Attias
Andrew Peter Yevsey Gorovoy
Bailey Nozomu Hwa
Shida Jing
Annie Wang

# Table of Contents

# 1. Introduction

The XIRTAM language is an object-oriented C-style language for manipulating matrices. Besides matrix declarations and operations, XIRTAM behaves analogously to C. In Xirtam, a user can create complex matrix-centric algorithms by using a built-in `xirtam` type, which is a representation of a matrix. The user of Xirtam ignores implementation details, and instead can focus on simple linear algebra procedures facilitated by simple instantiation, modification, augmentation, and algebraic manipulation of matrices.

## 1.1. C Operations

Xirtam broadly supports simple C operations in C-syntax, including control flow structures of for-loops, while-loops, and if-else blocks.

## 1.2. Matrix Operations

Xirtam matrices are built to allow for simple algebraic expressions involving two or more matrices, such as multiplication of two matrices. In addition, Xirtam supports popular single-matrix operations such as inversion of matrices.

## 1.3. Numeric Type

Unlike other matrix-oriented languages, Xirtam supports traditional numeric types of long-floats, integers, and doubles. Xirtam introduces a new type--`num`. That is a generic numeric type that can encompass integers, longs, floats, or doubles. On the backend, num types are converted to long floats. The num type increases the usability of Xirtam matrices, since the user interface provides flexibility in having one matrix support multiple basic types. Users need only instantiate a Xirtam matrix object when they specifically desire to use a matrix in their program.

## 1.4. Autocorrection and Convenient Error Checking

A unique feature of Xitram is its ability to autocorrect. Only code in `main()` will be executed and its return type is `num`. If the function header is of the incorrect type, Xirtam automatically corrects the header on the backend. Xirtam will also automatically correct the size of matrices. For example, if a matrix has a size of a decimal, Xirtam will round down to the nearest whole number. This is a usability feature in Xirtam, to make things convenient for the user.

Moreover, Xirtam provides error checking for undeclared variables and tells the user which expression is causing the error.

# 2. Language Tutorial

Compiling the project:

```
$ make
```

If the test script does not run due to "permission denied" upon attempting to run the test files, you can use the "chmod 777 ./testall.sh" command to change the permissions, then try again.

Please note that your environment must include clang.

If it says "./testall.sh not found", your system might use CRLF as the end-of-line character, when in fact it should use LF as the end-of-line character. Your editor might have ways to change this.

Testing a specific test file in the project:

```
$ make
$ ./xirtam.native -c ./tests/test-matadd.xirt
```

XIRTAM Language can be best described as MicroC with matrices, and built in operations on matrices.

When writing a XIRTAM Language program, the user needs simply to write a program using MicroC syntax. For novice users, MicroC syntax is exactly C syntax. XIRTAM Language programs support basic arithmetic on integers, doubles, and long floats. Control flow provided are for loops, while loops, and if-else statements. Additionally, string print and integer print functionality is provided. Finally, Xirtam allows users to create their own functions using C-like syntax.

A user can instantiate a matrix by using the Xirtam type. The Xirtam matrix may be instantiated as empty, or can be instantiated with a predefined 2-d array representing a matrix, in the following syntax:

```
xirtam matrix_2D;
matrix_2D = [[1,2,3],[1,2,2]];
```

Note that all Xirtam matrices must be represented as 2-D arrays. 1-D matrices (arrays) can be instantiated with the syntax shown in the `matrix_1D` object.

Below is a short example of a simple Xirtam program demonstrating some control flow structures. Please note that Xirtam Language programs begin by executing the contents of the main function automatically. The main function does not need to be subsequently invoked after instantiation.

```
num main(){
      bool value;
       num one;
       num one_point_one;
       num seven;
       value = true;
      if ( value ){
            seven = 7;
            one = 1;
            one_point_one = 1.1;
            printn(seven + one + one_point_one);
      else{
            print("this is a string print method");
      }
      function_example(1);
}
void function_example(num x){
       num var;num x;num var;num y;num z;

      for ( var = 1.1; var < 10; var = var + 1){
            while( x = -10; x > -1; x = x + 1){
                  y = -10.0;
                  z = -(y);
            }
      }
}
```

# 3. Language Manual

## 3.1. Lexical Conventions

In XIRTAM, newlines, spaces, tabs, and comments as described below are ignored. If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token. XIRTAM contains the following tokens:

### 3.1.1. Comments

The characters `/*` introduce a comment, which terminates with the characters `*/`.

### 3.1.2. Identifiers

A letter is defined to be one of the 26 English letters, either uppercase or lowercase. A digit is defined to be one of 0 through 9. An identifier is a sequence of letters and digits, with which the first character must be a letter. The underscore "_" counts as a letter. Upper and lower case letters are considered different.

Identifiers are used as corollary labels for variables, functions, and Xirtam matrix objects. The above definition of identifiers provides restrictions on valid identifiers.

### 3.1.3. Punctuators

Punctuators in XIRTAM have syntactic and semantic meaning to the compiler but do not, of themselves, specify an operation that yields a value. Any of the following in it of themselves are considered as punctuators.

```
! & * ( ) - + = { } | [ ] \ ; ' : " < > , . /
```

### 3.1.4. Keywords

The following keywords are reserved and cannot be used otherwise.

```
num
string
bool
xirtam
if
else
for
while
return
true
false
void
```

## 3.2. Basic Types

### 3.2.1. Strings

A string is a sequence of characters surrounded by double quotation marks `"`, declared by the keyword `string`. Double quotation marks are not allowed to appear in a string.

### 3.2.2. Numeric

A numeric is a number, encompassing integers and floating point numbers. It's represented as `num` and it could be negative, 0, or positive. An operation on two numerics always returns another numeric. An operation between a numeric and another type has different return types, depending on the situation.

### 3.2.3. Boolean

A `bool` is either the keyword `true` or the keyword `false`.

### 3.2.4. Xirtam

A `xirtam` is an array of arrays of type `num`. (see Section 3.5.)

## 3.3. Xirtam Expressions

### 3.3.1. General Overview

For non-Xirtam object types (see Section 3.5.), common unary operators such as `=, ==, !=, +, -, *, /, <, >, =<, >=, &&, ||` work the same way as they do in C. More specifically:

- The assignment operator `=` must have an identifier on the left, and anything on the right. If the right side is a value, it assigns that value to the identifier. If the right side is an expression, it evaluates the expression and assigns the value to the identifier. The assignment is kept within scope.
- `==, !=` evaluates both sides and returns a bool, analogous to C. `==` only evaluates to 1 or `true` if both the left and right side evaluate to the same logical conclusion. `!=` only evaluates to 1 or `true` if left and right side evaluate to opposite values.
- `+, -, *, /` can be used on numerics to perform arithmetic operations.
- `<, >, =<, >=` can only be used on numerics, analogous to C.
- `&&, ||` are logical operators used on booleans. It cannot be used on any other type. `&&` only evaluates to 1 or `true` if both the left and right side evaluate to 1 or `true`. `||` only evaluates to 1 or `true` if either the left and right side evaluate to 1 or `true`.

### 3.3.2. Operator Precedence

Operator precedence will take place in the following order (with the exception of assignment, Xirtam objects which have their own methods for dealing with certain operations as outlined in Section 3.5.). Within the same level of precedency, they are evaluated from left to right, and top to down. Parenthesis will override the precedence order, with expressions inside the parenthesis evaluated first:

1. Boolean negation, negative-sign
2. Multiplication, division
3. Addition, Subtraction
4. Equal not-equal, and other boolean operations
5. Assignment

## 3.4. Declarations

### 3.4.1. Local Variables

In a code block inside a function, all local variables that are going to be used need to be declared at the top of the block, before any control flow and assignments are made. Declarations are used to give identifiers certain values. To declare a variable, you must specify the type, followed by the identifier and semicolon. Following the statements of variable declarations, assignments happen in which you have an identifier followed by `=`, followed by the value you wish to assign. The following is the required basic pattern for declaring single variables (< > aren't part of the actual code):

```
<return_type> <function_name>(parameter list) {
    <type> <identifier>;
    …

    <identifier> = <value>;
}
```

If a value is of type `string` then the value must be defined in between two double quotes as stated previously inside a function.

```
<return_type> <function_name>(parameter list) {
    <string> <identifier>;
    …
    <identifier> = "<value>";
}
```

If a value is of type `num` then the value must be either a decimal or integer value inside a function.

```
<return_type> <function_name>(parameter list) {
    <num> <identifier>;
    …
    <identifier> = <value>;
}
```

If a value is of type `bool` then the value must be either `true` or `false` inside a function

```
<return_type> <function_name>(parameter list) {
    <bool> <identifier>;
    …
    <identifier> = <true or false>;
}
```

```
num main(){
    string s; num n; bool b;
    s = "Hello";
    n = 14;
    b = true;
}
```

This program declares a string type identifier **s** with the value “Hello”, a num type identifier **n** with value 14, and a bool type identifier **b** set to true, respectively.

### 3.4.2. Global Variables

Global variables can be initialized outside a function, at the top of the file but can only be assigned in the body of a function.

```
num global;
num main(){
    global = 1;
    return global;
}
```

### 3.4.3. Functions

XIRTAM supports function declarations in a standard C style. The function declaration must adhere to the following pattern:

```
<return_type> <function_name>(parameter list) {
     body of the function
}
```

`<return_type>` - is the type returned by the function
`<function_name>` - is the identifier used to reference the function declaration
`(parameter list)` - is the list of arguments passed to the function. Arguments must take the following form `<type>` `<identifier>`

For example, to write a function that adds two integers in XIRTAM it looks like this:

```
num foo(num a, num b){
      return (a+b);
}
```

### 3.4.4. Function Calls

In order to call a function, the function name must be called followed by parentheses inside `num main()`. If the function declaration has arguments, then the necessary arguments identifiers must be passed into the function reference. The following pattern must be followed:

```
num main(){
    function_name() /*if no arguments*/
    function_name(parameter list) /*if function declaration takes
arguments*/
}
```

A function that returns nothing has a declared type `void` with the exception of an empty `main()` as it will always have a return type of `num`.

## 3.5. Xirtam Matrix Objects

### 3.5.1. Overview

To declare a matrix object, you must use the `xirtam` keyword, followed in order by the identifier, assignment operator `=`, by the matrix expression, and finally by the matrix. Matrices can be expressed using an array of arrays format. The following is the required pattern for a matric array:

```
[ [<num>,<num>,<num>], [<num>,<num>,<num>], [<num>,<num>,<num>]... ]
```

The Xirtam matrix object can only take numeric type values or expressions that evaluate to the numeric type. There is no multi-type matrix object.The Xirtam matrix object is not dynamic. Once an object has been defined with a given number of rows and columns, it can no longer be changed.

The following is then the required pattern for declaring a Xirtam matrix object:

```
xirtam <identifier> = new matrix([[<num>,<num>,<num>],
[<num>,<num>,<num>], [<num>,<num>,<num>]... ]);
```

There are also multiple other ways to declare a Xirtam matrix object

```
xirtam matrix;

matrix = [ [1+5, 2, 3], [4, 5, 6], [7, 8, 9]];
matrix = autofill(2, 3, 0); /* a 2 row, 3 column matrix containing
the int value 0. */
```

### 3.5.2. Xirtam Matrix Operations

XIRTAM matrices have built-in operations for the basic operations such as elementwise, addition, multiplication. For equality, we return true for two matrices if (they are of the same size and they are equal elementwise). XIRTAM also has functions specific to matrix-operations beyond what is mentioned above:

#### 3.5.2.1. Print Matrix

Printing matrix requires a special function called `printm(m)` where `m` is of `xirtam` object. The `num` type values of the matrix will be printed as a float in the shape of the matrix as shown below:

```
xirtam m;
m = [[1, 2, 3],[ 4, 5, 6]];
printm(m);
/*
   1.00 2.00 3.00
   4.00 5.00 6.00
/*
```

#### 3.5.2.2. Basic Operations

Basic operations include adding two matrices with `matadd(m1, m2)` and multiplying two matrices aka dot product with `matmult(m1, m2)` where m1 and m2 are `xirtam` objects as show in the following. Both `matmult` and `matadd` return a new xirtam object, the matrix representing the result of each operation:

```
xirtam m1;
xirtam m2;
xirtam m3;
xirtam m4;
m1 = [[1, 2, 3],[ 4, 5, 6]];
m2 = [[10,11],[20,21],[30,31]];
m3 = matadd(m1, m1);
/* m3 should yield 2x3 matrix:
 2.00 4.00 6.00
 8.00 10.00 12.00
*/
m4 = matmult(m1, m2);
/* m4 should yield 2x2 matrix:
 140.00 146.00
 320.00 335.00
*/
```

### 3.5.2.3. Get and Set

To get the value of a certain position of a matrix use `matget(m1, row_num, col_num)`. To replace the value of a certain position in a matrix use `matset(m1, row_num, col_num)` where `m1` is a `xirtam` object, `row_num` is the row number starting from 0 and `col_num` is the column number starting from 0. Both `matget` and `matset` do not return a new matrix, but rather modify the matrix passed in as a parameter (here, 'm'), on the backend.

```
xirtam m;
num r;
m = [[4, 2], [422, 21], [0.4, 6.2]];
matset(m, 2, 0, -1.233);
r = matget(m, 2, 0);
printn(r); /* 0.4 */
```

### 3.5.2.4. Number of Rows and Columns

To get the number of columns of a matrix use `getcols(m)` and to get the number of rows of a matrix use `getrows(m)` where `m` is a `xitram` object. Both `getcols` and `getrows` return a num type.

```
xirtam m;
```

```
num row_num;
num col_num;
m = [[1,2,1,3],[1,2,2,3]];
row_num = getrows(m); /* 2 */
row_num = getcols(m); /* 4 */
```

#### 3.5.2.5. Autofill

`autofill(num_of_rows, num_of_cols, value)` creates a new matrix of `xirtam` type with certain `num_of_rows` and `num_of_cols` populated with only this specific `num` type `value`. `Autofill` returns a new xirtam object. For example:

```
xirtam m;
m = autofill(3,3,1);
printm(m);
/*
   1.00 1.00 1.00
   1.00 1.00 1.00
   1.00 1.00 1.00
/*
```

#### 3.5.2.6. Transpose

To find the transpose of a matrix use `trans(m)` where `m` is a `xirtam` object. This function will return a new xirtam object.

```
xirtam m;
m = [[1, 2], [3, 4]];
printm(trans(m));
/*
1.00 3.00
2.00 4.00
*/
```

## 3.6. Statements and Control Flow

### 3.6.1. Print Statement

```
printn(num); /* only for numerics */
printm(matrix); /* only for xirtam objects */
```

### 3.6.2. Conditional Statements

```
if (condition1) {
   statement(s)
}
else {
   statement(s)
}
```

Statements in the `if` block is evaluated only if the condition evaluates to true. Otherwise, the program checks to see if an `else ` or block exists. If not, proceed to the next statement.

### 3.6.3. For Loop

```
for ( init; condition; increment ) {
   statement(s);
}
```

**Init** is executed first and only once. It allows you to declare and initialize any loop control variables. Then the **condition** is evaluated. If it is true, the **statement(s)** will be evaluated and afterwards, the **increment** will happen. This loop is repeated until the condition evaluates to `false`.

### 3.6.4. While Loop

```
while ( condition ) {
   statement(s);
}
```

While the **condition** is true, **statement(s)** will be executed until the **condition** becomes false. Then the line immediately after the while loop will be executed.

# 4. Project Plan

## 4.1. Planning Process

We began by selecting each role each member was interested in pursuing. Then, we brainstormed the overall functionality of our language, answering questions about the types of programs that our language would facilitate. Due to an overall interest in linear algebra in the group, we began to develop a language that would make it easier for an end user to perform matrix operations in linear algebra programs.

We wrote our Language Reference Manual and specified the functionality and syntax of different parts of our language.

Then, we began writing the syntax of our language using the appropriate files. As we did this, we iterated on the Language Reference Monitor to match more closely the direction we went with for the Xirtam language compiler. Finally, we developed the syntactical rules of the language, and iterated again on the rules, syntax, and functions of our language while staying true to the main goal of our language-- increased usability of matrix objects.

Throughout this process, we met once a week to discuss our progress from a high level perspective and make sure we were all staying on track.

## 4.2. Specification Process

We first created our general requirements via several meetings. We set the general functionality of Xirtam and brainstormed about a sample program that we would be able to execute using the Xirtam language.

We then converted the general requirements into a google doc. Once we felt comfortable with the requirements, we transported the requirements into the Language Reference Manual.

Then, we added details to the LRM to describe our language's specifications in greater detail. We also created a test suite to allow for Test Driven Development, which helped connect our abstract specifications to concrete requirements in code.

Specification was documented in the Language Reference Manual. As we modified the code, we modified the language reference manual accordingly. In the end, the Language Reference Manual (section 3 in this report) reflects the final specifications of Xirtam.

## 4.3. Development Process

Overall development process:

> We first created the Parser and Scanner based off of the specifications in our initial Language Reference Manual. We then created a blank AST, a basic semant and sast, and a minimal codgen file, and a basic testing suite.
>
> Once we linked up our basic components, we bolstered and modified all the components, increasing our test suite as we completed new features.

Once we finished coding and testing a compiler to handle basic C functionality, we focused on implementing the matrix objects in Xirtam.

Development process for each team member:

Every team member followed the same process for development. We each created a development environment. Some used the docker file and others used a unique virtual environment. Team members that could not create the testing environment were paired with team members who could, and paired program together.

Code was written locally, and tested locally. Small changes were pushed directly to the master branch on Github if checked by multiple people.  Larger changes were first pushed into a branch, and then after a thorough review were pushed to master.

Whenever a new team member pulled from master or a branch, the test suite was re-run before any modifications were made to ensure that nothing is broken as new functionality is added.

## 4.4. Testing Process

We developed an automatic testing using a bash script. As we developed the codegen, the test suite was in place to prevent regression errors.

## 4.5. Roles and Responsibilities

Initially, the roles listed in the table below were the initial roles but it became more fluid as everyone was working on different parts at the same time.

The responsibilities listed in the table below does not mean only that individual was doing all the work. They just have a more dominant role. Everyone chimed in via pair programming, debugging as a group, working around different people's busy schedules and so on.

| **Team Member** | **Roles** | **Responsibilities/Results** |
|---|---|---|
| Bailey Nozomu Hwa | Co-manager, Co-Architect (Compiler Design) | Designed the Xirtam Language, Co-developing the compiler, testing, codegen development, semant development, managing the product + timeline, ensuring things get done |

| | | |
|---|---|---|
| Shida Jing | Co-manager, Co-Architect (Compiler Design), Lead Tester | Co-developing the compiler , testing, debugging/environment setup, semant development |
| Lior Attias | Language/Tester (Environment Selection) | Testing, debugging, feature dev of core xirtam compiler, compilation and development environment, development of matrix features,  Final Report |
| Andrew Peter Yevsey Gorovoy | Manager/Architect (Environment Selection) Test suite lead | Testing, co development of codegen and semant, test environment, make, compilation environment |
| Annie Wang | Manager | Final Report, Testing, test environment, make, compilation environment, organizing meetings and keeping team on track with deliverables |

## 4.6. Project Timeline

| **Date** | **Milestone** |
|---|---|
| February 24 | Parser, Scanner, Ast version 1 |
| Early March | Parser, Scanner, Ast version 2 |
| Mid-march | LLVM environment set up |
| March 24th | Hello world milestone |
| Late march | Codgen compiles LLVM code |
| Early april | Codegen compiles control structures, basic types, special types (num), and print statements. Produced LLVM files execute successfully on tests. |
| Early april | Automatic test suite solidified, make file solidified, increase variety and breadth of tests in test suite |

|  |  |
|---|---|
| April 5-7th | Added major feature of uninitialized variable checking and error code debugging for user; |
| Mid April | Implement internal matrix object + testing |
| Late April | Finalize environment, add built in matrix operations: matget, matset, madadd, matmult, trans |

## 4.7. Project Log

```
commit a4729ba47075be4b1fc4c10263fa9b54b4dbe246
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Apr 25 19:34:43 2021 -0500

    Added citation

commit 96161c0e01d3162aa77378a1d97ee8360abd0a44
Author: Chimer2017 <apg2165@columbia.edu>
Date:   Sun Apr 25 18:40:20 2021 -0400

    removed fail test dependent on machince

commit c64237a76ea40047387ca02a81ae5b55a615803b
Author: Chimer2017 <apg2165@columbia.edu>
Date:   Sun Apr 25 18:36:01 2021 -0400

    added another fail test, small edit on error call in matrix.c file

commit d128b15ea5dbf85856c571d49d83a3d077b0fd29
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Apr 25 15:55:32 2021 -0500

    Got rid of the new empty return tests

commit f9f8c34c14c20413bd56068fdd81b4c48ee1ff7b
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Apr 25 15:42:03 2021 -0500

    Added a test for non-return type

commit 5e6d376901f5c13638e908da9f288f7e2aca6749
```

```
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Apr 25 15:38:24 2021 -0500

    Added tests and fail tests for global variable

commit 15f49aead048e39c99143c08c83b1672d2f840e9
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Apr 25 15:06:35 2021 -0500

    Added a citation to a test program

commit e439c24be44ef492738c22e2bcffaceeee3873c9
Author: Annie Wang <anwang911@gmail.com>
Date:   Sun Apr 25 16:01:52 2021 -0400

    deleted unnecessary whitespace

commit c47a775f319ec749020e7483b42144e14eb069f9
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Apr 25 10:07:18 2021 -0500

    modified demo program

commit bda8b1a706d1aa4b51502627cf6c99ada7b12220
Author: Chimer2017 <apg2165@columbia.edu>
Date:   Sun Apr 25 00:58:11 2021 -0400

    added in a number of fail tests

commit 315d997160d868a624147e71bfef7839bee63824
Merge: 0656804 de0a7f1
Author: Chimer2017 <apg2165@columbia.edu>
Date:   Sat Apr 24 17:01:00 2021 -0400

    git issues

commit 0656804f55d222257630447c2ba6d50416f96d27
Author: Chimer2017 <apg2165@columbia.edu>
Date:   Sat Apr 24 16:59:45 2021 -0400

    added autofill function

commit de0a7f1e2750d1feaf224a9172de22ce8426b4a4
```

```
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sat Apr 24 15:55:04 2021 -0500

    Cleaning and adding citations

commit 6f98d164aacfb7fd981ffe9c92dc80416af227d5
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Sat Apr 24 16:35:03 2021 -0400

    cleaned code

commit 386b8dc9587aa0fd7eeda06ce0b0cb2845e2c7f1
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sat Apr 24 15:30:43 2021 -0500

    Cleaning the code

commit a8fa58477c58f2938a2e666f0b1ffb49c984913c
Merge: 685cf10 9607492
Author: Chimer2017 <apg2165@columbia.edu>
Date:   Sat Apr 24 16:03:12 2021 -0400

    git issues

commit 685cf10935f6b8a0810474b048125cb32f0e5100
Author: Chimer2017 <apg2165@columbia.edu>
Date:   Sat Apr 24 15:59:49 2021 -0400

    Added getrows and getcols functionality to codegen as well as tests to
the test suite. Andrew Gorovoy, Lior Attias

commit 9607492f121ab4cfd281968b5af8b327002cc8f4
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sat Apr 24 14:42:09 2021 -0500

    Getting rid of mod operation

commit ccf29d3c1f8b8c8435d41915b92c7adc902675bd
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sat Apr 24 13:50:51 2021 -0500

    Added a test for transpose
```

```
commit 27c365efaa49c1ce918294fec9af37722abaf069
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Sat Apr 24 12:38:58 2021 -0400

    test transpose: to call in xirtam, use  trans(matrix);

commit 326c826ae7b716deae9b8795f81d71cbec429c75
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sat Apr 24 11:12:39 2021 -0500

    Update testall.sh

commit 9426076087011129d3ee96826ffe655ad3f18def
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Sat Apr 24 11:33:29 2021 -0400

    removed unused scanner tokens

commit f44b6bf860d110246603555a73fde05e9774b850
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Sat Apr 24 11:31:43 2021 -0400

    added author tags to beginning of files

commit 90fce1c9aa5f625ecf316f9ef162780c21e33ce9
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Sat Apr 24 01:16:30 2021 -0400

    Readded Shida and Andrew's tests

    Re-added Shida and Andrew's tests

commit 6eca99eb64eaa1ea50b1acc024d658c75fc016d7
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 22 17:50:17 2021 -0500

    Added a cool program

commit ed78c365fdb497db2c58dc484dd56f886d5e79ab
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 22 16:53:15 2021 -0500

    Added more test cases.
```

```
commit 2bba48d773fd809a03c8fff5d9a0e7ad904e56e8
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Wed Apr 21 20:26:53 2021 -0500

    Added more tests: matrix mult, set, get

    Andrew, Bailey, Shida

commit 50f0bd000c1aef83638d686aedd12ea834d795fe
Merge: 37873f8 092de7f
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 21 21:03:47 2021 -0400

    Merge branch 'matrix_alt_implementation'

commit 37873f8a07e1469b34f24a10189fcd55dde684c4
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 21 21:01:38 2021 -0400

    get rid of faulty tests; merge my new revamped changes;

    session: Bailey, Shida, and Andrew;
    almost ALL matrix.c from scratch as original implementation

commit 092de7fbe634150d81f1ddc3b01eeace9a618141
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 21 16:21:46 2021 -0400

    alternate original implementatino of matrix that fixes segfaults

commit 4f63d2af9b370bbd310a96741aadd8442329a60a
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Tue Apr 20 23:46:04 2021 -0400

    fixed more row column swaps in matrix.c

commit 45aa084c75599d3cb601866fa283e73056f38717
Merge: e6ccfdc c8c1571
Author: Dkyse <jingshid@grinnell.edu>
Date:   Tue Apr 20 21:45:53 2021 -0500

    Merge pull request #11 from bnhwa/Attempt-for-mx
    Matrix add now works

commit c8c15717740e65d970404fe47d0e7a65d3ef316d
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Apr 20 21:45:10 2021 -0500

    Matrix add now works

commit e6ccfdc0030672407b2a05799baf13a33d8f002c
Merge: dc2d71f dff9152
Author: Dkyse <jingshid@grinnell.edu>
Date:   Tue Apr 20 21:23:16 2021 -0500

    Merge pull request #10 from bnhwa/Attempt-for-mx

    Reverted semant to Bailey's version. Matrix print and declaration
works.

commit dff9152ad959887f94c22a4a1bfabb4a4038c30b
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Apr 20 21:21:57 2021 -0500

    Printm and declaration.

    Reverted semant back to bailey's version.

    Lior
    Bailey
    Andrew
    Shida

commit 72e357303a352be9163f424799aa3df30adb64a9
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Apr 20 21:05:30 2021 -0500

    added matrix addition and multiplication, needs debug

commit dc2d71f72aa2205526af94609cd28bd54ec84435
Author: Lior Attias <lattias@email.arizona.edu>
Date:   Tue Apr 20 17:32:09 2021 -0700

    fixing tests
    fixing tests

commit cc9609f6346a278dd328776eca427582437798a9
Author: Lior Attias <lattias@email.arizona.edu>
Date:   Tue Apr 20 17:04:46 2021 -0700

    Create test-matrix5.xirt

commit dbb8e75e1284d1d6e9fd43a4ad8ec3b913cbbe92
Author: Lior Attias <lattias@email.arizona.edu>
Date:   Tue Apr 20 17:02:59 2021 -0700

    Create test-matrix4.xirt

commit fd512469e72ebbdff468de96c286e5e85df55c52
Author: Lior Attias <lattias@email.arizona.edu>
Date:   Tue Apr 20 16:50:32 2021 -0700

    Create test-matrix3.xirt

    testing main, call funct, invoke matrix op

commit 5bed35e1faf2cb385cdd994e0052e9a7957e934f
Author: Lior Attias <lattias@email.arizona.edu>
Date:   Tue Apr 20 16:49:06 2021 -0700

    Create test-matrix2.xirt

commit a83fb2fc0682cc1a855b1e8eb97607f3c2fd9300
Author: Lior Attias <lattias@email.arizona.edu>
Date:   Tue Apr 20 16:45:00 2021 -0700

    added matrix initial test

commit 35067b0b96208cabb24698ce414b492c349f45c9
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 15 14:49:47 2021 -0500

    Made it compile

commit 021bf340d5be3f6e3e196624ef4f5a0e4d9c8ace
Author: ShidaJingTetra <jingshid@grinnell.edu>
```

```
Date:   Wed Apr 14 19:12:16 2021 -0500

    bug when compiling

commit 65874083da92dd73a25d4976e1f92d693402bcef
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Apr 13 16:51:54 2021 -0500

    debugging the .c file

commit 79d5f207587c21a1a4570cd51a3a0e81460d71f8
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Mon Apr 12 16:46:48 2021 -0500

    implementing matrix print

commit 678b8caad5093037e7f08168aed1d0390c8ad0e8
Merge: dd5a5ad 64e674b
Author: Dkyse <jingshid@grinnell.edu>
Date:   Fri Apr 9 17:49:50 2021 -0500

    Merge pull request #9 from bnhwa/debug-semant-and-add-matrix-to-codegen

    Fixed warnings in semant. Added dimensions to matrices.

commit 64e674b9834cae78215674bcc0064997b49f2f88
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Apr 9 17:47:53 2021 -0500

    Continued

commit dd5a5ad43e88e854d0713f8f276b23cd4146f0f5
Merge: 6f61091 db6d8ac
Author: Dkyse <jingshid@grinnell.edu>
Date:   Fri Apr 9 08:28:49 2021 -0500

    Merge pull request #8 from bnhwa/debug-semant-and-add-matrix-to-codegen

    Debug semant

commit db6d8ac24872ff12becea9a60f47c57375c292f4
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Apr 9 08:28:18 2021 -0500
```

```
    delete test log

commit 3c21c192e6827b71d1bc78dc6ff38ba8aedf322d
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 8 16:48:04 2021 -0500

    debug semant

    1. Int type not accounted for in codegen. Added it so that it treats
Int as Num

    2. Deleted the exponent function in codegen

    3. Cannot fix error in semant.ml regarding init_check_helper

    4.

commit 6f61091c364226acaffaf6f610c11d40d1bf3b1e
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 19:36:21 2021 -0400

    Modified readme, added xirtam logo which I designed

commit 777c36fdc22b8a3d6b2694b7172686001f3b321d
Merge: cdfcd1c be8aa78
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 14:45:14 2021 -0400

    Merge branch 'master' of https://github.com/bnhwa/PLT-Project

commit cdfcd1ca5bcc4c81337e0699ed8286c06dec4f96
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 14:45:10 2021 -0400

    seemingly fixed print issue with call semant checking

commit be8aa78e899053a29c9a447bb2068b4696b0fce8
Author: Chimer2017 <apg2165@columbia.edu>
Date:   Wed Apr 7 18:39:34 2021 +0000

    fail-tests
```

```
commit 99fa42222c4d9a33ab52123ba1d1b751e4ff56a4
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 14:33:11 2021 -0400

    i think i fixed the negative issue

commit 8e65d21a63cc6bc8ab6eab22f10d456eed5563a0
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 13:51:33 2021 -0400

    added uninitialized variable check for returns

commit d77473c932aa93da0602062e97fae1fa1d03f047
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 13:44:42 2021 -0400

    updated todo and cleaned code for semant regarding uninitialized
variable checks

commit 367630a76151256adf5eb27cf802e8687fd22a55
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 13:41:20 2021 -0400

    added uninitialized var checking for matrix elements

commit e008e9be9dc868d9f8af47a55c25756e4071b8c6
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 13:35:14 2021 -0400

    fixed uninitialized check, works for everything now!!!!!!!

commit 80a6f954e3db782e02338b0a347a4923c88b890a
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 13:13:29 2021 -0400

    added init checks for ifs loops

commit f76be11dffa9798ac03d37f5cc0ef9e4b9733715
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Wed Apr 7 12:55:28 2021 -0400

    added initialization type checking for variables (unfinished)
```

```
commit 98a83937492f06564754ed9c08607ede8b69b1d3
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Tue Apr 6 22:43:31 2021 -0400

    quick error code fix on staggered columns

commit fba8adc5c7456821e0990d76c85e9ed6b616c339
Author: Bailey Hwa <bb339933@gmail.com>
Date:   Tue Apr 6 22:40:08 2021 -0400

    Added semantic checking for matrices, including checks for expression
types within matrices, staggered rows, and removed continue

commit 31379f0d97e1a4df9c791a42feeca099d3fa0fc1
Author: Bailey <bb339933@gmail.com>
Date:   Mon Apr 5 17:11:28 2021 -0400

    added matrix stuff into parser and ast

commit 7f8bbb8dd5476d1b132d09fc67e080aafcae827b
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Apr 2 20:32:08 2021 -0500

    Added tests for functions and whiles

    ignoring mod for now.

commit 7d6976798cb3c8f57729cf9886aef16d905e13a5
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Apr 2 20:11:16 2021 -0500

    make tests for if and while

commit 410eb1efad6ac5039b7a127d26491424581f7a22
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Apr 2 18:10:45 2021 -0500

    updated tests for if and unary operators

    while and for loops don't work

commit 6809cf6c7c94c4e1fa4a4140acd781ca94aa26cb
Author: ShidaJingTetra <jingshid@grinnell.edu>
```

```
Date:   Fri Apr 2 17:13:53 2021 -0500

    File extension fixed

commit b720a326202a1818662993198df4b7f5629cb85d
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Apr 2 17:06:37 2021 -0500

    Added arithmetic test 1/2

commit 47bb33d9efc5711af085ab43ddabb27d134c6d04
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 16:33:21 2021 -0400

    re added the mod operator for nums

commit 1862a635f2346f019ddca0d457d1ecd4547b2a91
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 16:08:58 2021 -0400

    added readme.md

commit 6a2a9771eacaf0d27051825a852236caf688616d
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 16:06:51 2021 -0400

    modified Readme with concurrent info

commit f6dea793e38be8c2d3a2f3f6be3622abb1861378
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 16:00:45 2021 -0400

    made semant return error if user tries to return something from the
main function

commit d28cabf40317a34b36dec34b3eda7f6704f388ab
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 15:34:18 2021 -0400

    updated todo, Entry point is main is always made to be hidden int type,
decl/return is same type

commit 79213f6fd3b4a1eba669ad3c44a042b5433e23e6
```

```
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 15:19:56 2021 -0400

    fix return type for int main

commit 657b156a8d5ab2fc01de3e51766b544a10578131
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 14:53:34 2021 -0400

    potential int return type mismatch fix

commit cfa77ed103cc7c7164109b1d520059969a9b38ce
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 14:46:45 2021 -0400

    printn fix

commit 7e59c9cba7c3393637fa2262ea8b1121fe9f7208
Author: Bailey <bb339933@gmail.com>
Date:   Fri Apr 2 14:41:18 2021 -0400

    added hidden int type that is only used for function main, whatever
type main is declared as, in semant it will become int type so llvm has
proper entry

commit 118d70e82cbcf1c0a2db21da2b4b2510962e8bd1
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 1 22:00:14 2021 -0500

    Update TODO.txt

commit 6ade4fb6a07788176d6033084e7e4aedd62c3edc
Author: lattias <lattias@email.arizona.edu>
Date:   Thu Apr 1 21:43:05 2021 +0000

    remove printbig files and reflect this change in make and testall

commit c13a277bc793bb84bcb98cb2e8da903e978e689f
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 1 15:41:00 2021 -0500

    attempt to fix printn
```

```
commit f788302d525bd4f1b742c6b68cd2e2fdc57f2952
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 1 15:36:29 2021 -0500

    Modified quotation mark

commit 7ced48032c3e9fdd9ac9c55aecf6e31fd9e50500
Merge: 745e102 df7cb8b
Author: Bailey <bb339933@gmail.com>
Date:   Thu Apr 1 15:52:15 2021 -0400

    Merge branch 'master' of https://github.com/bnhwa/PLT-Project into
master

commit 745e102dfafe8fbae39fe18a224e5d9c756c4805
Author: Bailey <bb339933@gmail.com>
Date:   Thu Apr 1 15:51:56 2021 -0400

    codgegen works for variable assignment/reassignment,print numerics,
function declaration, llvm modules output fine, we should physically run
the llvm module

commit df7cb8b977bd546527b979f2bc5506f0fe57c134
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 1 14:49:53 2021 -0500

    Delete printbig.o

commit dc7949ccc344f82d4fab7b7e50454171a95d11a7
Author: Bailey <bb339933@gmail.com>
Date:   Thu Apr 1 15:40:26 2021 -0400

    add strings in codegen, make llvm e1 e2 syntax compact

commit 65dccde40b41c643cd26c3e64e74e94fc5d24dbb
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 1 14:33:52 2021 -0500

    Fixed a quotation mark

commit e97cad630248d4bed38d903d6b3261ffecb14f99
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Thu Apr 1 14:23:26 2021 -0500
```

```
    Added TODO list and temporarily moved test files until makefile works.

commit 28bb5cd9383cb15a6721fc90cef2142a59f57d34
Author: Annie Wang <anwang911@gmail.com>
Date:   Wed Mar 31 23:49:08 2021 -0400

    resolved error testall.sh

commit b5a2863c38c0b35ae4d5b7812c5d0868dd1a4623
Author: Annie Wang <anwang911@gmail.com>
Date:   Wed Mar 31 23:39:38 2021 -0400

    automated testing and added test cases

commit de587ed2b541c41432e11360b9f5c76c1480d408
Merge: a203fd7 9b7c0c7
Author: Dkyse <jingshid@grinnell.edu>
Date:   Tue Mar 30 14:37:06 2021 -0500

    Merge pull request #7 from bnhwa/fix-codegen

    Fix codegen

commit 9b7c0c78d4808071404ff453805bb6569de9e0aa
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Mon Mar 29 21:10:12 2021 -0500

    codegen conti

commit cde8e6a53b2b7795ce04711eccbbc244a97195ef
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Mon Mar 29 20:58:47 2021 -0500

    Fixing codegen

    1. we redefined the pointer type in codegen

    2. We are defining the i32 type for the printf to return

    3. Fixed SNumLit because it is not a string in our code but it is a
string in the microc.
```

```
    4. Commented out mod and exp for now in ast and codegen.

commit a203fd774781c16788d5f7e1bf4680b792a78775
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Mon Mar 29 18:17:20 2021 -0500

    IMPORTANT: added tags file

commit 9b3e0bb6ea9a6f2075f908ee17403e122aca44fd
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 22:53:29 2021 -0400

    name fix for greater operator in codegen

commit a8e88dd04b9408d21b34a358d5e53c565dd95645
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Mar 28 21:02:16 2021 -0500

    Trying to run the program

commit 2a4527eab2089abdf8439fdde7f32c1365c43bd0
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 21:23:56 2021 -0400

    fixed the parser for mod and exp

commit ca357db9b960cc0e1a89a7ee6bf50254ee8b6786
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Mar 28 20:17:30 2021 -0500

    exp and printf

commit 9969dc94b27c3167bccb2eedbe13848025c17af0
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Mar 28 20:16:28 2021 -0500

    codegen fix

commit e2a31b72f972bb71cc2dd5afdc1f3ef194546489
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 21:01:59 2021 -0400

    small fix for power function llvm
```

```
commit 645c24d7eb416a75609871ac7d3d822ef8603420
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 20:58:31 2021 -0400

    get rid of A.float because we are using A.Num

commit 3c2c176ae91f10d870181df65db1b249f890a41a
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 20:56:58 2021 -0400

    added statements in codegen and SBinops for specific datatypes please
check this in case

commit faa7842dce15eeeaed1385392af54a52de66cb7f
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 20:08:46 2021 -0400

    added exp and mod operators, for codegen since we are using nums
basically floats, look at LLVM const_frem which should do the trick for mod
for power look further into llvm manual

commit 55afb68d8e215ab99aea47fd27950609004c0813
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 19:38:04 2021 -0400

    comment out fix in parser

commit 8492c8f538895e7889d3430a2d0213e891f4df78
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 19:33:57 2021 -0400

    fixed issue with if statement in parser, see new hello txt demo

commit 83a0f606ef186104c54e79c1a05c0060a02b69e8
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 19:05:14 2021 -0400

    added and tested semants/ast/sast/ for binary operators

commit b9eec12ca43d5de62932cc2b02ff7c31ccc09f69
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 18:56:22 2021 -0400
```

```
    added and tested semants/ast/sast for unary operators

commit 80f9a835d082cd333e17eaa86f38f5f34ac201b6
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 18:43:20 2021 -0400

    small change in duplicate arg/local error message

commit d2391e1caccd32d4b715525275f78183e0933977
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 18:40:50 2021 -0400

    added in function_check checking for duplicate local and arg variables

commit 4ff831a5b466d62bebbb3829b430a4e9d35b4792
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 18:35:02 2021 -0400

    added string compatability with semant

commit 5ca35ee30e3ffcd216ce49efcbc0cb52e442544b
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 28 16:39:57 2021 -0400

    more codegen additions to handle var dec/formals and 3rd optional
assignment value

commit 218fc4f5183b5c90cc024a76f0a6ac17fbad98fa
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Mar 28 14:44:12 2021 -0500

    codegen still needs statements

commit 40d2b5fd6e9cfcec5e2c43b81e7743bfa336c5af
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sat Mar 27 21:37:08 2021 -0500

    codegen continued

commit 94a97678a022416c77de5365ff60c0e1aee3588d
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sat Mar 27 18:46:12 2021 -0500
```

```
    Starting on codegen

commit 3406313a30f39f770851baa38c16916387491603
Author: Bailey <bb339933@gmail.com>
Date:   Fri Mar 26 22:16:53 2021 -0400

    revamped structure of code, parser now follows suit with microc style
declarations, fixed sast bindings, restructured code to be more simple,
function delclaration and locals now work great with semant.

commit 94d13c0e2dbc700ff6b54dee6a534461e55a4f91
Merge: d4e0bc5 c7b0222
Author: Bailey <bb339933@gmail.com>
Date:   Fri Mar 26 21:35:43 2021 -0400

    Merge branch 'master' of https://github.com/bnhwa/PLT-Project into
master

commit d4e0bc5009639daae9c1c88823d55014741d5e05
Author: Bailey <bb339933@gmail.com>
Date:   Fri Mar 26 21:34:40 2021 -0400

    bailey beta test version of project

commit c7b02229fe2f13afd0b717cc9e76cfb14dadeed7
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Wed Mar 24 21:06:22 2021 -0500

    Added readme

commit 8f014dc01697d65f4653f724d41bac8464ec591d
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 24 21:08:09 2021 -0400

    delete alt old version scanner

commit 71d7406827bb2f5ad7b914a70499ac23610a8fe3
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 24 21:06:50 2021 -0400

    remove extraneous files
```

```
commit 6505e30b1b6b5fe0cee4c7affa373e37751b0843
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 24 19:54:14 2021 -0400

    new helloworld

commit 41dafe1a9b7f5e1ccba5a0a88741577299cfa779
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 24 19:35:50 2021 -0400

    sast fix

commit 5e01d48cf9e09f01a35fb2ec1eecbd6a91009ea4
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 24 15:52:00 2021 -0400

    fixed!

commit bed755b9a59b699818c709e9a64b0a8d629120e1
Merge: f437013 0d62b97
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 24 15:50:33 2021 -0400

    asdf

commit f437013ef8de0cc99ba1d94c7a2b387c409dcf91
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 24 15:49:09 2021 -0400

    pretty print working now!!

commit 0d62b97cdfcbcb8f5913ec222eb9c11f0a1a27b0
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Wed Mar 24 14:08:41 2021 -0500

    Added debug code in parser

commit c2616bccd9036bec7c823c976178e579fbe78889
Merge: 20d04aa c5066e3
Author: Dkyse <jingshid@grinnell.edu>
Date:   Tue Mar 23 21:32:31 2021 -0500

    Merge pull request #6 from bnhwa/Fix-semant
    Fix semant

commit c5066e3aad7623b481205bc05e07791613c2c86d
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Mar 23 21:32:15 2021 -0500

    Continued

commit 3570106e8829e2ffef554816a3724daa802a3f02
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Mar 23 21:06:22 2021 -0500

    Testing the program

commit 20d04aa235b0307eb160ebb4f935ef9ed85d2af6
Merge: c9dca36 0ad3179
Author: Dkyse <jingshid@grinnell.edu>
Date:   Tue Mar 23 20:11:01 2021 -0500

    Merge pull request #5 from bnhwa/Fixing-AST

    Fixing ast and semant, in progress

commit 0ad317981b3141e6ac8a6856b55c2c95a6ee7654
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Mar 23 20:10:34 2021 -0500

    Fixing semant

commit 32419540b8cdb7f762085f0d861a156d15f310e7
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Mar 23 20:02:59 2021 -0500

    In progress

commit a38d3cd9ddaba068a570ae2d1e9c636dd0107a6e
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Mar 21 20:53:32 2021 -0500

    Trying to debug sast

commit ea4874e716c7fd0818daee79ed967bf3198d1ded
```

```
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sun Mar 21 20:21:38 2021 -0500

    Made xirtam.ml copy of microC

commit d405c4974aa9e856c3685fea260a7cc15aa882d5
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Sat Mar 20 22:26:43 2021 -0500

    Semant and Sast untested

commit 79bd01bee281cbc8c24ab2187e8df9dd499fd8d5
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Mar 19 22:06:06 2021 -0500

    Fixed ast, I think.......

commit 851660c0a0dde30e00f251ddb4002256221c167c
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Mar 19 21:15:34 2021 -0500

    Fixing AST

    1. Deleted Mod and Exponent operation
    2. Commented out XirtamDec_lit_ and XirtamDec_rc_

commit c9dca369235a0e4b19f56cbb2ae33599e2b686d3
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Fri Mar 19 12:34:17 2021 -0500

    Debugging  AST

commit c04b0665d46e063917acbbb618a4698e9295c2ac
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 17 20:42:20 2021 -0400

    reorder ast operators akin to microc

commit c6b6881a4af0ff22f58921e46a0387013a21a934
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 17 20:39:05 2021 -0400

    more astAlt corrections
```

```
commit de9dac427a88289e34672534b9e57315ca4ce32c
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 17 19:45:56 2021 -0400

    minor fix

commit 7805f78710a14086025291015ad22fd26a16db58
Author: Bailey <bb339933@gmail.com>
Date:   Wed Mar 17 19:42:39 2021 -0400

    scanner fix, ast matrix declaration print string error handling
implemented

commit ae10bf4263a46ee3d8c693a68f21e0cc58964858
Author: Bailey <bb339933@gmail.com>
Date:   Tue Mar 16 21:01:24 2021 -0400

    minor ast edits

commit cc82eebafc824ded8ca07587f65eb3c84437fad3
Author: Bailey <bb339933@gmail.com>
Date:   Tue Mar 16 20:55:48 2021 -0400

    parser and ast, caution, ast isnt finished

commit 010c970d74b941130361e874a3fb48bb110c6aa2
Author: Bailey <bb339933@gmail.com>
Date:   Tue Mar 16 15:08:09 2021 -0400

    parser fix

commit eef637ce225c8e200f25c1c836820801ecedc2b8
Merge: 4a98250 db95971
Author: AndrewG <34191548+Chimer2017@users.noreply.github.com>
Date:   Mon Mar 15 23:20:20 2021 -0600

    Merge pull request #4 from bnhwa/andrew_feature

    Andrew feature

commit 4a98250b78e43c5385981f5bbd80363294ea7aff
Merge: 0f4a449 648b40c
```

```
Author: Dkyse <jingshid@grinnell.edu>
Date:   Tue Mar 16 00:20:01 2021 -0500

    Merge pull request #3 from bnhwa/hello_world_in_progress

    Helloworld stuck at bug

commit 648b40cd89ef0c5bba1fc2328ec4ead37e1bdf3d
Author: ShidaJingTetra <jingshid@grinnell.edu>
Date:   Tue Mar 16 00:19:45 2021 -0500

    Helloworld stuck at bug

commit db959712a4aa8d76992ff5b19d6bd0f575955bcb
Author: Andrew Gorovoy <andrewgroovy@gmail.com>
Date:   Mon Mar 15 23:18:26 2021 -0600

    added test folder and script

commit 0f4a449221b50b8cbbc7481b3fb0a7f25f023248
Merge: cc3a71e 36e8e0b
Author: Dkyse <jingshid@grinnell.edu>
Date:   Mon Mar 15 22:58:28 2021 -0500

    Merge pull request #2 from bnhwa/hello_world_shida

    Hello world in progress

commit 36e8e0b5b161168b8baa520184066c5abe6ff081
Merge: 6348f4c cc3a71e
Author: Dkyse <jingshid@grinnell.edu>
Date:   Mon Mar 15 22:58:16 2021 -0500

    Merge branch 'master' into hello_world_shida

commit 6348f4c62b65946113b569811d77f643b0a56d58
Author: ShidaJingTetra <sjing@tetrascience.com>
Date:   Mon Mar 15 22:55:03 2021 -0500

    helloworld in progress

commit 17020b023da9f175b78ffed8020476e06fff11b9
Author: ShidaJingTetra <sjing@tetrascience.com>
```

```
Date:   Mon Mar 15 22:44:24 2021 -0500

    helloworld in progress

commit cc3a71ef9b32286537946fdf68cd7da314b426b4
Author: Bailey <bb339933@gmail.com>
Date:   Mon Mar 15 19:24:20 2021 -0400

    new changes

commit 79bd62cc9c67236591a57458b7fe8285eb0166b6
Author: Andrew Gorovoy <andrewgroovy@gmail.com>
Date:   Mon Mar 15 16:26:43 2021 -0600

    added testfile on personal feature branch

commit d091b646c4039265beee2cd27ffe81d855a6176b
Author: Bailey <bb339933@gmail.com>
Date:   Mon Mar 15 15:39:11 2021 -0400

    modify helloworld txt to match xirtam syntax

commit 2e4578e872518babca82a4e640763a9ce1eef952
Author: Bailey <bb339933@gmail.com>
Date:   Mon Mar 15 13:18:09 2021 -0400

    xirtam functions added to parseralt and scanner alt

commit b6cfe47454e7a6385bf1069500ce2736275a86b8
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 14 21:06:40 2021 -0400

    redesigned scanner and parser completely, 0 shift reduce errors

commit 2bd3ea0df4a97922d7f266ffa8614bf9517f3f57
Author: Bailey <bb339933@gmail.com>
Date:   Sun Mar 14 18:17:37 2021 -0400

    alt files

commit 6b047e32a41420bcd7fa017595cbb99a4d42426d
Merge: fbc4cc9 0a4e706
Author: Dkyse <jingshid@grinnell.edu>
```

```
Date:   Sun Mar 14 15:39:00 2021 -0500

    Merge pull request #1 from bnhwa/hello_world_shida

    Hello world initialization

commit 0a4e706dcb149fe52b76ad15f57300dc1ce9940d
Author: ShidaJingTetra <sjing@tetrascience.com>
Date:   Sun Mar 14 15:38:38 2021 -0500

    Initialization on helloworld

commit 248f85e10e55014825f2abaa469a4fed0cf51815
Author: ShidaJingTetra <sjing@tetrascience.com>
Date:   Thu Mar 11 19:18:39 2021 -0600

    Sample hello-world program

commit fbc4cc9f96f9da90d884c8c0a6c9966d1a7a42fa
Author: Bailey <bb339933@gmail.com>
Date:   Thu Mar 11 20:10:11 2021 -0500

    initial commit
```

## 4.8. Software Development Environment

- Docker environment used to compile LLVM code.
- Internal software development done in Visual Studio Code and Sublime.
- Source Control done via Github for code and Google Drive for documents.

**List of dependencies:**

- Clang
- LLVM 10
- Python 2.7
- Cmake
- Opam
- Ocaml
- Menhir
- Pkg-config
- aspcud
- ca-certificates
- LLVM 10-dev

## 4.9. Programming Style Guide

Utility functions

For functions that are used multiple times, a utility function will be implemented. Additionally, for functions that are unreasonably long, a utility function will be implemented for added readability.

Commenting

All comments were set as block comments or individual lines describing functionality to ensure that all working or reviewing the code understands what is going on. Block comments are indented to be in a single column, and spaces separate the start and end. For example:

```
/* one line comment */
/* Multi
  Line
  Comment */
```

Parser

Formatting on parser is meant to enhance readability. Every new pattern is separated by a double new line. The first rule does not contain a bar, all remaining rules do contain a bar. The rules are left-aligned to the start of the first character. The actions are left-aligned to the start of the leftmost curly bracket. If there is no pattern for a rule, then a comment reading “nothing” is included. When there is no action, “Null” is set as the function to call. There is no spacing within the curly brackets. For example:

```
rule_one:
      Pattern_one {functionA}
     |Pattern_2   {functionB}

rule_two:
      /* nothing */ {Null}
     |Pattern_one {functionA}
     |Pattern_two   {functionB}
```

AST conventions

All AST types will be lowercase in naming. Names of each type are capitalized. For example:

```
type type_one:
     Name_one of other_type
```

```
      | Name_two of different_type
```

Ocaml conventions

All Ocaml will be aligned to clearly indicate the nesting of 'let' and 'in'.
The declarations of 'let' and 'in' will be left aligned. For multiple rules, the two will be on separate lines. For small rules, they can be on one line.

Comments are used both within and outside of 'let' and 'in' blocks to clarify ambiguity.

For example:

```
let value =
     /* many lines of rules */
in   /* continue code */

let value = ... in ...
```

## 5. Architectural Design

A Xirtam language program goes through several steps. At the end of the compilation process, the Xirtam language program has been translated into LLVM code. This LLVM code is translated further down into another LLVM compiler (abstracted away from this project), which finally executes the LLVM translation of the Xirtam Language Program.

First, the Xirtam Language Program is compiled via Xirtam.native. The Xirtam Language Program is also compiled with matrix.c, a c-file that is used to implement complex matrix operations.

Xirtam.native first builds the AST tree, which contains the syntax rules.

The AST tree is built by having scanner first convert the XLP into tokens, and then having parser define the possible combinations of these tokens.

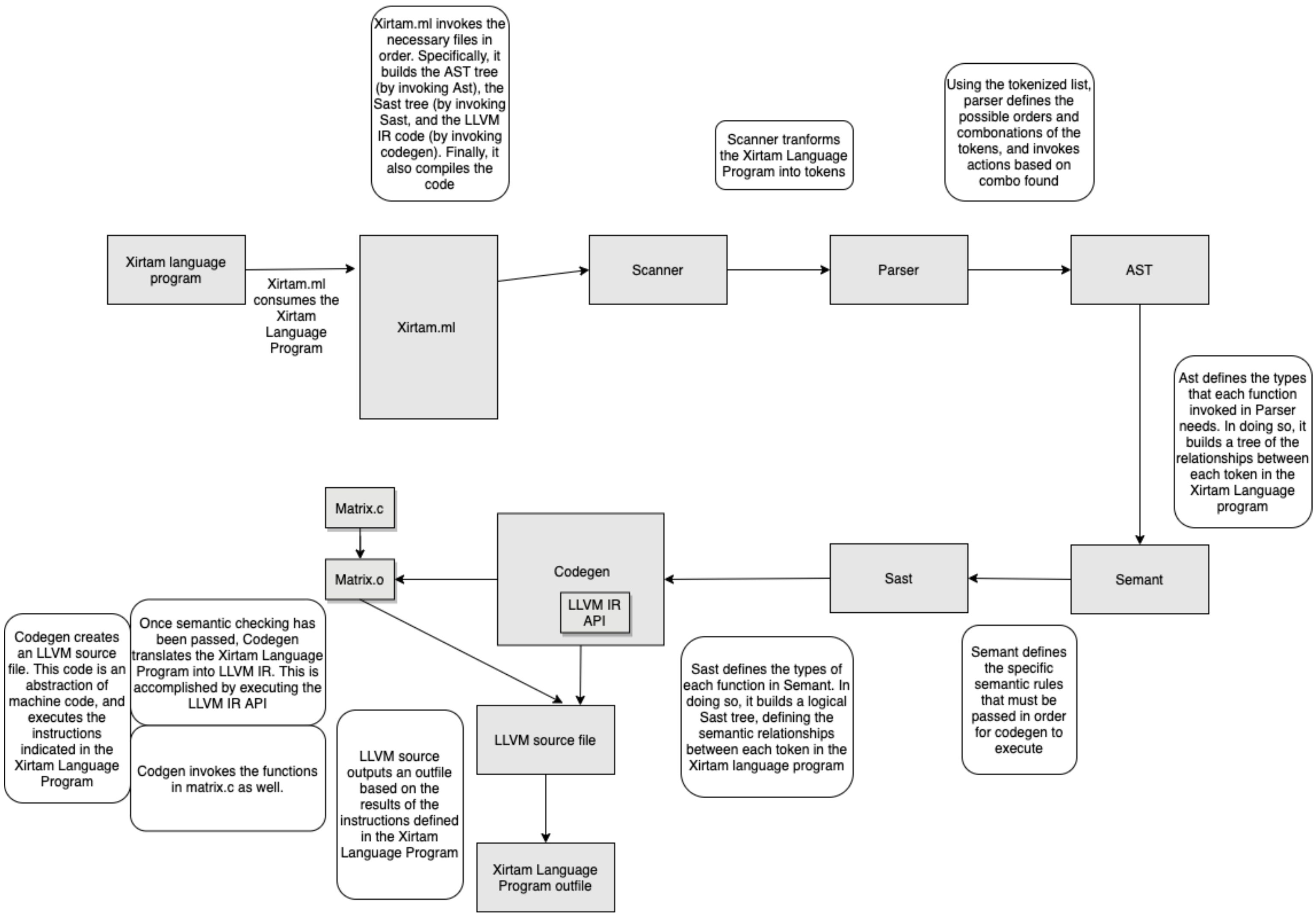
If the AST tree is built correctly using the rules in parser and the token defined in scanner, then the XLP is syntactically correct.

Next, the Xirtam.native builds the SAST tree, which is the semantically checked tree. The semant consumes the AST tree and checks if it is semantically correct. If so, it outputs the SAST tree, which is the semantically correct tree representation of the XLP. Once this stage is passed, codgen consumes the SAST tree and converts the SAST tree into LLVM code.

In the last step, the LLVM code is executed by an LLVM compiler on the backend. Finally, a .out file is generated which is the output of the LLVM code.

At the end, the LLVM code logically matches the XLP the user provided to be executed. Below is a diagram.

Figure 1. Code flow -- big picture



Xirtam.ml invokes the necessary files in order. Specifically, it builds the AST tree (by invoking Ast), the Sast tree (by invoking Sast, and the LLVM IR code (by invoking codegen). Finally, it also compiles the code
Scanner tranforms the Xirtam Language Program into tokens
Using the tokenized list, parser defines the possible orders and combonations of the tokens, and invokes actions based on combo found
Xirtam language program
Xirtam.ml consumes the Xirtam Language Program
Xirtam.ml
Scanner
Parser
AST
Ast defines the types that each function invoked in Parser needs. In doing so, it builds a tree of the relationships between each token in the Xirtam Language program
Matrix.c
Matrix.o
Codegen
LLVM IR API
Sast
Semant
Codegen creates an LLVM source file. This code is an abstraction of machine code, and executes the instructions indicated in the Xirtam Language Program
Once semantic checking has been passed, Codegen translates the Xirtam Language Program into LLVM IR. This is accomplished by executing the LLVM IR API
Codgen invokes the functions in matrix.c as well.
LLVM source outputs an outfile based on the results of the instructions defined in the Xirtam Language Program
LLVM source file
Xirtam Language Program outfile
Sast defines the types of each function in Semant. In doing so, it builds a logical Sast tree, defining the semantic relationships between each token in the Xirtam language program
Semant defines the specific semantic rules that must be passed in order for codegen to execute

An original Parser, Scanner, and AST were created by Lior and Shida. Bailey then from scratch, created a functional Parser, Scanner, and AST. Bailey and Shida then expanded and tested all of these files, creating a Semant file. This involved Bailey and Shida creating a xirtam.native file which was used to test the ast, sast, and semant files which were created in such order. Annie and Andrew worked to create a makefile to compile the program. Lior, Annie, and Andrew worked on creating an environment (docker) that could be used to compile this program. With the environment set up, the group worked together to debug the files. As the environment was being solidified (including make files and test environment), Bailey and Shida focused their efforts on starting a codegen file. With this direction, Bailey, Shida, and Andrew implemented and tested the codgegen and matrix.c file. For the testing suite, Shida, Lior, Andrew, and Annie worked to debug the codegen file so that it could build LLVM code. Shida, Lior, Annie, and Andrew focused on running this code and creating the test suite that would actually execute the LLVM code.

Once a very simple empty main function was able to be compiled, the project quickly transformed as the most difficult parts-- the mechanics of getting LLVM to compile, having the correct environment, having all the required files work together without bugs, were ironed out. Bailey and Shida designed and implemented basic types, algebraic types, and function declaration types. Andrew, Lior, and Annie supported debugging and feature development via pair programming. Finally, the Xirtam matrix types were created and implemented procedurally from parser, ast,sast/semant, and finally in codgen with the team supporting pair-programming and getting the code to compile, run, and test correctly.

The final step of this process was to get the "fun" features of Xirtam completed. The entire team worked together to get the final built-in features of Xirtam completed. The core functionality of the built-ins resided in matrix.c, which was linked into the project via clang.

When we first started this project we had set labels for each team member. However, as the project progressed every team member contributed to overcome difficulties with the code, from providing the necessary environment to test on, debugging code, defining the direction and code structures, and ironing out language details and implementation best practices questions. We learned that in designing a language, every role and function informs the other.

## 5.1. Group member credit

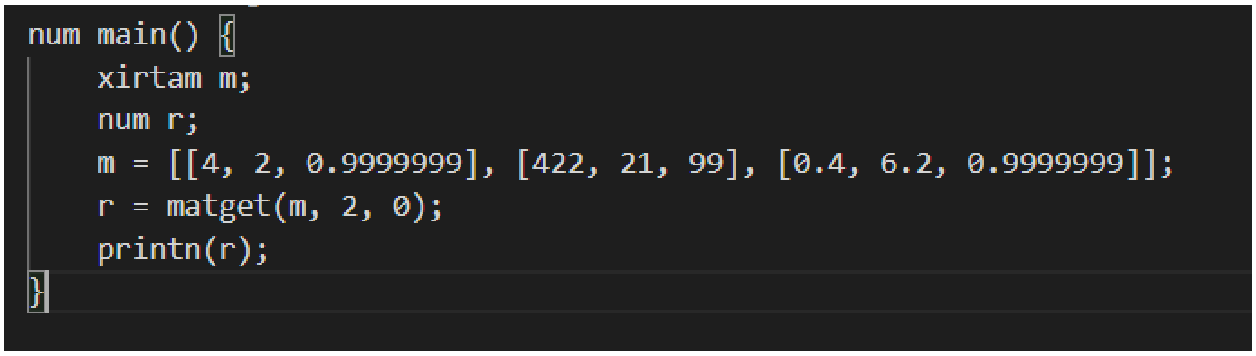
Please see appendix 9: XIRTAM COMPILER CODE, for specific credit on each file.

# 6. Test Plan

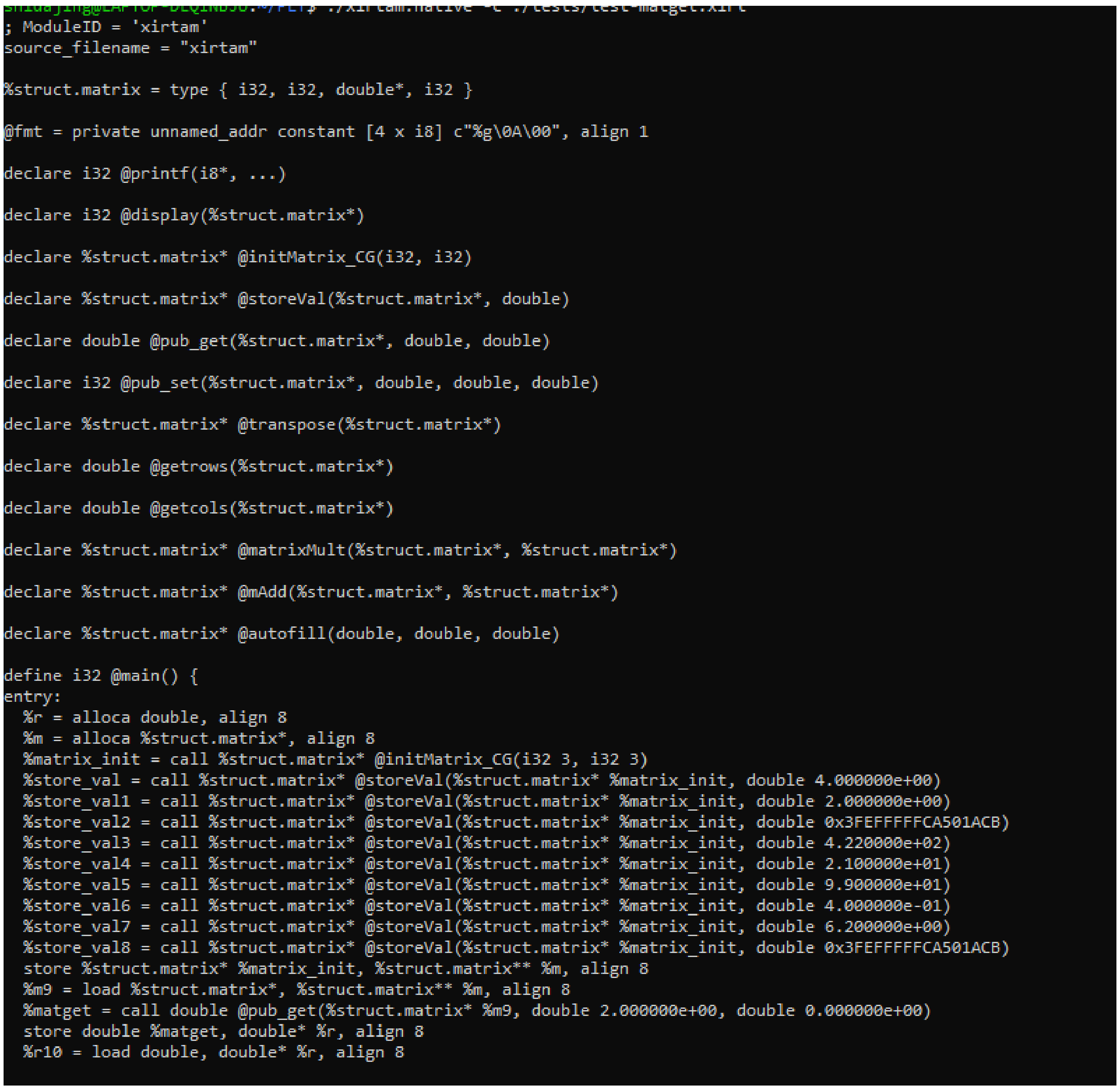
## 6.1. Two Representative Programs

Program one:
This is a simple program in which a matrix, **m**, is instantiated with both integer and decimal values. On the backend, these are converted to **'num'** types.

The built-in function **matget** returns the specific value (type is **num**) at the x,y coordinate given. Here, it will return the value located at 2,0 of matrix m.



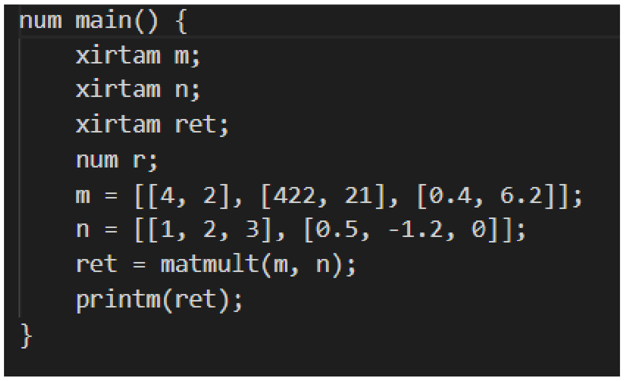
```
num main() {
    xirtam m;
    num r;
    m = [[4, 2, 0.9999999], [422, 21, 99], [0.4, 6.2, 0.9999999]];
    r = matget(m, 2, 0);
    printn(r);
}
```

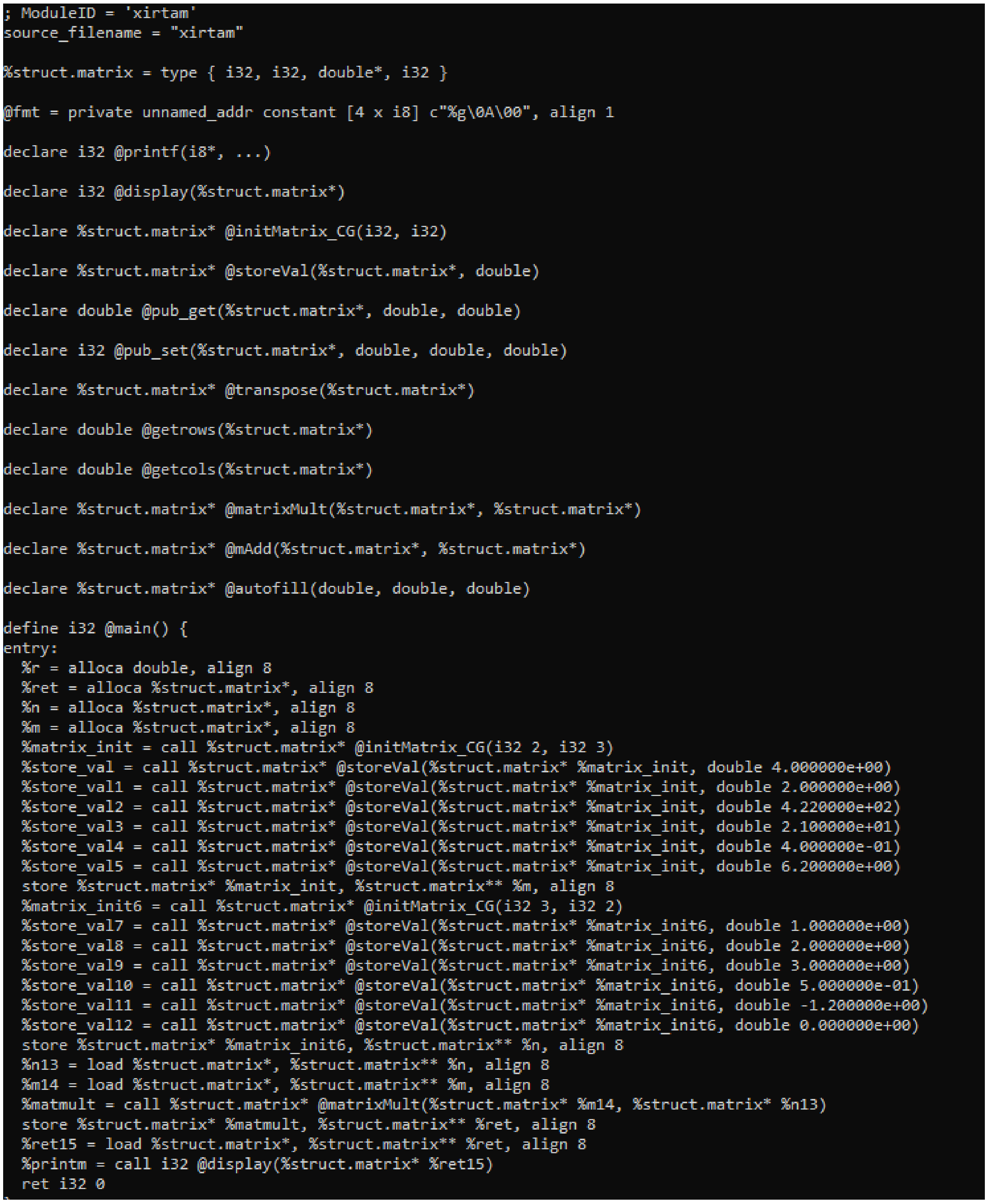


```
; ModuleID = 'xirtam'
source_filename = "xirtam"

%struct.matrix = type { i32, i32, double*, i32 }

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

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

declare i32 @display(%struct.matrix*)

declare %struct.matrix* @initMatrix_CG(i32, i32)

declare %struct.matrix* @storeVal(%struct.matrix*, double)

declare double @pub_get(%struct.matrix*, double, double)

declare i32 @pub_set(%struct.matrix*, double, double, double)

declare %struct.matrix* @transpose(%struct.matrix*)

declare double @getrows(%struct.matrix*)

declare double @getcols(%struct.matrix*)

declare %struct.matrix* @matrixMult(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @mAdd(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @autofill(double, double, double)

define i32 @main() {
entry:
  %r = alloca double, align 8
  %m = alloca %struct.matrix*, align 8
  %matrix_init = call %struct.matrix* @initMatrix_CG(i32 3, i32 3)
  %store_val = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 4.000000e+00)
  %store_val1 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 2.000000e+00)
  %store_val2 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 0x3FEFFFFFCA501ACB)
  %store_val3 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 4.220000e+02)
  %store_val4 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 2.100000e+01)
  %store_val5 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 9.900000e+01)
  %store_val6 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 4.000000e-01)
  %store_val7 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 6.200000e+00)
  %store_val8 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 0x3FEFFFFFCA501ACB)
  store %struct.matrix* %matrix_init, %struct.matrix** %m, align 8
  %m9 = load %struct.matrix*, %struct.matrix** %m, align 8
  %matget = call double @pub_get(%struct.matrix* %m9, double 2.000000e+00, double 0.000000e+00)
  store double %matget, double* %r, align 8
  %r10 = load double, double* %r, align 8
```

Program two:

This program illustrates the **matmult** built in function. Here, two matrices, **m** and **n**, are initialized. Note that it is not necessary to pre-define the size of the matrices before initialization. **Matmult** will multiply the two matrices, m and n. **Matmult** returns a matrix "**ret**" representing the resultant matrix. Finally, **printm** prints the resultant matrix (**'ret'**) in a human readable format.


```
num main() {
    xirtam m;
    xirtam n;
    xirtam ret;
    num r;
    m = [[4, 2], [422, 21], [0.4, 6.2]];
    n = [[1, 2, 3], [0.5, -1.2, 0]];
    ret = matmult(m, n);
    printm(ret);
}
```



```
; ModuleID = 'xirtam'
source_filename = "xirtam"

%struct.matrix = type { i32, i32, double*, i32 }

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

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

declare i32 @display(%struct.matrix*)

declare %struct.matrix* @initMatrix_CG(i32, i32)

declare %struct.matrix* @storeVal(%struct.matrix*, double)

declare double @pub_get(%struct.matrix*, double, double)

declare i32 @pub_set(%struct.matrix*, double, double, double)

declare %struct.matrix* @transpose(%struct.matrix*)

declare double @getrows(%struct.matrix*)

declare double @getcols(%struct.matrix*)

declare %struct.matrix* @matrixMult(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @mAdd(%struct.matrix*, %struct.matrix*)

declare %struct.matrix* @autofill(double, double, double)

define i32 @main() {
entry:
  %r = alloca double, align 8
  %ret = alloca %struct.matrix*, align 8
  %n = alloca %struct.matrix*, align 8
  %m = alloca %struct.matrix*, align 8
  %matrix_init = call %struct.matrix* @initMatrix_CG(i32 2, i32 3)
  %store_val = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 4.000000e+00)
  %store_val1 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 2.000000e+00)
  %store_val2 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 4.220000e+02)
  %store_val3 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 2.100000e+01)
  %store_val4 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 4.000000e-01)
  %store_val5 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init, double 6.200000e+00)
  store %struct.matrix* %matrix_init, %struct.matrix** %m, align 8
  %matrix_init6 = call %struct.matrix* @initMatrix_CG(i32 3, i32 2)
  %store_val7 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init6, double 1.000000e+00)
  %store_val8 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init6, double 2.000000e+00)
  %store_val9 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init6, double 3.000000e+00)
  %store_val10 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init6, double 5.000000e-01)
  %store_val11 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init6, double -1.200000e+00)
  %store_val12 = call %struct.matrix* @storeVal(%struct.matrix* %matrix_init6, double 0.000000e+00)
  store %struct.matrix* %matrix_init6, %struct.matrix** %n, align 8
  %n13 = load %struct.matrix*, %struct.matrix** %n, align 8
  %m14 = load %struct.matrix*, %struct.matrix** %m, align 8
  %matmult = call %struct.matrix* @matrixMult(%struct.matrix* %m14, %struct.matrix* %n13)
  store %struct.matrix* %matmult, %struct.matrix** %ret, align 8
  %ret15 = load %struct.matrix*, %struct.matrix** %ret, align 8
  %printm = call i32 @display(%struct.matrix* %ret15)
  ret i32 0
```

## 6.2. Test Plan

<u>Chronologically</u>

1. Hello World executable

    a. Test an simplistic program to establish end to end functionality
2. Test suite
    a. Basic type invocations
    b. Printing
    c. Matrix creation and operations
    d. Arithmetic operations on numbers
    e. Error handling

With each modification to the compiler, the test suite was run. The following shows the required output in order to succeed at all the tests:

```
./testall.sh
test-arithmetic-op...OK
test-assignment...OK
test-autofill...OK
test-cool-program-2...OK
test-cool-program...OK
test-determinant...OK
test-emptymain...OK
test-for-c...OK
test-func-dec...OK
test-getcols...OK
test-getrows...OK
test-global-variable...OK
test-if...OK
test-matadd-2...OK
test-matadd-3...OK
test-matadd...OK
test-matget...OK
test-matmult-2...OK
test-matmult...OK
test-matset...OK
test-print-matrix1...OK
test-print-matrix2...OK
test-printn...OK
test-temp...OK
test-transpose...OK
test-unary-and-bool-op...OK
test-while-c...OK
test-while...OK
```

```
fail-add1...OK
fail-add2...OK
fail-arithmetic-op...OK
fail-assign1...OK
fail-assign2...OK
fail-emptymat...OK
fail-func-arg...OK
fail-func-arg2...OK
fail-func-dec...OK
fail-getcols...OK
fail-getrows...OK
fail-global-variable...OK
fail-if...OK
fail-intmain...OK
fail-mult1...OK
fail-mult2...OK
fail-uninit-var-usage...OK
fail-while...OK
fail-while2...OK
```

## 6.3. Automated Testing

Created bash file called **testall.sh** to run test suite automatically (see Appendix for the actual file)

## 6.4. Testing Credit

The test suite was written by Andrew Gorovoy, Annie Wang and Lior Attias. The test cases, both test and fail, were chosen to demonstrate the proper functionality as well as condition for failure for every added feature of our language. Annie Wang and Andrew Gorovoy set up the testing script and wrote the initial tests. Shida Jing, Bailey Hwa, and Lior Attias added other test files throughout the semester.

## 6.5. Test suite -- see Appendix 8

**Actual tests code are in the Appendix 8: TEST SUITE**

# 7. Lessons Learned and Advice for Future Students

## 7.1. Lior Attias

Overall, this project was an interesting challenge. While it may seem that we only built a compiler, I believe it was a project that actually helped us think about the many facets of software engineering. To succeed in this project, we could not only make a test suite and start coding. With every decision, we closely considered the design of the overall architecture, the relationships between each component, the data flow we wanted to ensure, and the usability we wanted to provide through our language.

Overall, I enjoyed the entire process. I think one of my favorite aspects was the large scale nature of the compiler, in that it encompassed multiple pieces each of which had a very specific role. Because of this, we were able to really create a mini version of a complete large scale compiler. Through this hands-on learning, I was really able to understand the purpose of each component.

One tip that I think is important for groups to take away from this project is that one of the most important steps in the process is the design of how the user will interact with the language. This is one design decision that helps inform many decisions you will make about every portion of the compiler down the line.

## 7.2. Andrew Peter Yevsey Gorovoy

This project provided an opportunity of course to learn the ins and outs of compiler design. The Ocaml language is at first confusing, but its advantages and strong suites quickly became apparent. The language itself had many small nuances and required a good amount of time to figure out all the bugs.

Regarding the project, a huge lesson was understanding how to properly build and expand upon existing code. At first, we started off trying to build from scratch and then transferred to expanding upon the microc code. The web development practice of testing one incremental step at a time was very helpful in this case as it allowed us to focus and isolate on certain issues.

In all honesty, the greatest learning experience from this project was understanding how to work on a complex project as a team. This was a long term project with complex subject matter. Each person had to do different things and this required a lot of communication and collaboration. This team experience has been very valuable.

## 7.3. Bailey Nozomu Hwa

This project taught me--a Python fanatic-- that functional languages such as Ocaml are actually cool. I feel that people should actually give OCaml a shot. Ocaml is akin to the Twizzler's candy, it's nasty at first, but once you eat enough of it, it starts to grow on you. The project itself is quite nifty and feels great once you have finished things.

I recommend future PLT students to start early, make sure people get things done on time, and ensure that the development environment is squared away first. If this is done, things will run smoothly. Never assume that code is good until everyone has tested it and searched every nook and cranny for errors. Maintain a centralized workflow and never get complacent until everything is finished from start to end.

### 7.4. Shida Jing

Technically, I have learned and become more comfortable with functional language. Particularly, I have learned good methods to use when debugging a functional language program, or just reading a functional language program to try to understand it.

Practically, I have learned that when doing a complicated project, starting with a thin thread of things that work, using test suites, and gradually expanding the width of the thread is the best way to go. The whole process becomes more manageable and trackable. At every point in time, you are very aware of what you have accomplished, and what is next to come.

I’ve also learned that starting early is always the best way to go. Divide and conquer when the task is too big, and never be afraid to ask for help when you need it.

### 7.5. Annie Wang

The most important learning I had was having tight communication with the team by setting up meetings as frequently as needed so everyone is on the same page. It’s easy when one group member is creating certain functionalities in the code to not try to understand how that code is written. Not doing so causes more time inputted when there’s a bug and the person who isn’t writing the code goes and tries to fix it. I definitely felt bad asking my group members to explain some things several times but realized that it’s a group effort and personal ego will only get in the way.

My advice is to really ask the TA’s and professors tons of questions! Pay attention to class, especially when the professor is going over the MicroC code. Ocaml seems scary but the more you look at it, tinker and ask questions, you’ll get to understand Ocaml a bit more. Celebrate every win you make during the process!

# 8. Appendix

Citation: project skeleton inspired by Professor Edwards's MicroC and a previous project called Matrx. You can find Matrx here: http://www.cs.columbia.edu/~sedwards/classes/2018/4115-fall/

This section and the next section contains a complete listing of the test suites and a complete listing of our code, in that order.

## 8.1. Test Suite

`Add-fail1.xirt`

This test checks a failure catching condition in addition of two matrices.

```

num min(num a, num b) {
    if (a >= b) {
        return b;
    } else {
        return a;
    }
}

num min_half(num a, num b) {
    num min;
    min = min(a, b);
    return min/2;
}

num main() {
    xirtam m;
    xirtam m2;
    xirtam r;
    m = [[1],[1,2]];
    m2 = [[1]];
    r = matadd(m, m2);
    printm(r);


}
```

Expected output:

`Add-fail1.err`

```
Fatal error: exception Failure("No staggered Matrices allowed, rows must be same size")
```

`Add-fail2.xirt`

This test checks a failure catching condition in addition of two matrices.

```

num min(num a, num b) {
    if (a >= b) {
        return b;
    } else {
        return a;
    }
}

num min_half(num a, num b) {
    num min;
    min = min(a, b);
    return min/2;
}

num main() {
    xirtam m;
    xirtam m2;
    xirtam r;

    m = [[]];
    m2 = [[-1]];
    r = matadd(m, m2);
    printm(r);


}
```

Expected output:

`Add-fail2.err`

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

`fail-arithmetic-op.xirt`

This test checks a failure catching improperly formatted arithmetic addition on num types.

```
num main() {
    num k;
    k = 1 + ;
}
```

Expected output:

`fail-arithmetic-op.err`

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

`fail-assign.xirt`

This test checks a failure catching improperly formatted assignment operation between strings, nums, and bools.

```
num main(){
    string s;
    num n;
    bool b;
    s=12;
    n=6;
    n=30.2;
    b=true;
    b = (6 == 8);
}
```

Expected output:

`fail-assign1.err`

```
Fatal error: exception Failure("illegal assignment string = num in s = 12.")
```

`fail-assign2.xirt`

This test checks a failure catching improperly formatted assignment operation between strings, nums, and bools.

```
num main(){
    string s;
    num n;
    bool b;
    s=12;
    n=6;
    n="hello";
    b=true;
    b = (6 == 8);
}
```

Expected output:

`fail-assign2.err`

```
Fatal error: exception Failure("illegal assignment num = string in n = hello")
```

`Fail-func-arg.xirt`

This test checks a failure catching bad passing of parameters to a user defined function; ie. fewer parameters in user-defined function call than are required.

```
num add(num a, num b) {
    num c;
    c = a + b;
    return c;
}

bool small_num(num input, num threshold) {
    return (input < threshold);
}

num main() {
    num k;
    num threshold;

    k = add(1, 2);
    printn(k);

    k = 0.34;
    threshold = 1;
    if (small_num(k)) {
        printn(k);
    } else {
        printn(100000);
    }
}
```

Expected output:

`fail-func-arg.err`

```
Fatal error: exception Failure("expecting 2 arguments in small_num(k)")
```

`Fail-func-arg2.xirt`

This test checks a failure catching bad passing of parameters to a user defined function; ie. more parameters in user-defined function call than are required.

```
num add(num a, num b) {
    num c;
    c = a + b;
    return c;
}

bool small_Num(num input, num threshold) {
    return (input < threshold);
}

num main() {
    num k;
    num threshold;

    k = add(1, 2, 3);
    printn(k);

    k = 0.34;
    threshold = 1;
    if (small_Num(k, threshold)) {
        printn(k);
    } else {
        printn(100000);
    }
}
```

Expected output:

`fail-func-arg2.err`

```
Fatal error: exception Failure("expecting 2 arguments in add(1., 2., 3.)")
```

`Fail-func-dec.xirt`

This test checks a failure catching bad declaration of a user-defined function; ie, capitalization of function call not matching function declaration

```
num add(num a, num b) {
    num c;
    c = a + b;
    return c;
}

bool small_Num(num input, num threshold) {
    return (input < threshold);
}

num main() {
    num k;
    num threshold;

    k = add(1, 2);
    printn(k);

    k = 0.34;
    threshold = 1;
    if (small_num(k, threshold)) {
        printn(k);
    } else {
        printn(100000);
    }
}
```

Expected output:

`Fail-func-dec.err`

```
Fatal error: exception Failure("unrecognized function small_num")
```

`fail-get-cols.xirt`

This test checks a failure catching of a poorly defined call to a built-in function; ie, getcols requires a matrix input, not a num input.

```
num main() {
    num test;
    num result;

    test = 12;

    result = getcols(test);
}
```

Expected output:

`fail-get-cols.err`

```
Fatal error: exception Failure("illegal argument found num expected xirtam in test")
```

`fail-get-rows.xirt`

This test checks a failure catching of a poorly defined call to a built-in function; ie, getrows requires a matrix input, not a num input.

```
num main() {
    num test;
    num result;

    test = 12;

    result = getrows(test);
}
```

Expected output:

`fail-get-rows.err`

```
Fatal error: exception Failure("illegal argument found num expected xirtam in test")
```

`fail-global-variable.xirt`

This test checks a failure catching of a poorly defined call to a built-in function; ie, getrows requires a matrix input, not a num input.

```
num global;

global = 4;

num temp(num a) {
  global = 2;
  return a + 1;
}

num main()
{
  num i;

  global = 9;
  temp(4);

  i = 5;
  while (i > 0) {
    printn(i);
    i = i - 1;
  }
  printn(42);
  printn(global);
}
```

Expected output:

`Fail-global-variable.err`

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

`fail-if.xirt`

This test checks a failure of a badly defined if statement; ie, the if statement has no condition where one is expected.

```
num main() {
    num f;
    f = 8;

    if {

    }

}
```

Expected output:

`fail-if.err`

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

`fail-intmain.xirt`

This test checks a failure of a badly defined main statement; ie, a return type is specified in the function declaration, but nothing is returned.

```
int main () {
    num f;
    f = 4;
}
```

Expected output:

`fail-intmain.err`

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

`fail-mult1.xirt`

This test checks a failure of a badly called matrix multiplication call. This function ("matmul") is a built in function. It expects compatible sized matrices to be multiplied together. Here, badly sized matrices (incompatible for multiplication) are being passed to the function, which will throw an error as expected.

```
num main() {
    xirtam m;
    xirtam n;
    xirtam ret;
    num r;
    m = [[4, 2,1], [422, 21], [0.4, 6.2]];
    n = [[1, 2, 3], [0.5, -1.2, 0]];
    ret = matmult(m, n);
}
```

Expected output:

`fail-mutlt1.err`

```
Fatal error: exception Failure("No staggered Matrices allowed, rows must be same size")
```

`fail-mult2.xirt`

This test checks a failure of a badly called matrix multiplication call. This function ("matmul") is a built in function. Matmul built-in function expects to receive matrixes that can be multiplied together. Here, a matrix with an empty field is passed to matmul. Matmul will throw an error as expected.

```
num main() {
    xirtam m;
    xirtam n;
    xirtam ret;
    num r;

    m = [[1, 3, -1, -3]];
    n = [[1], [], [-1], [4]];
    ret = matmult(m,n);
    printm(ret);
}
```

Expected output:

`fail-mutlt2.err`

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

`fail-uninit-var-usage.xirt`

This test checks a failure of badly used num types. Here, a num type "m" is instantiated, but it is assigned to a type of "y" which has not yet been defined.

```
num main() {
    num m;
    m = y + 1;
}
```

Expected output:

`Fail-uninit-var-usage.err`

```
Fatal error: exception Failure("undeclared identifier y")
```

`fail-while.xirt`

This test checks a failure of badly defined while loop. Here, the while loop does not contain a condition which is required.

```
num main() {
    num i;
    i = 0;
    while() {
        if (i == 3) {
            printn(1);
        } else {
            printn(100);
        }
        i = i + 1;
    }
}
```

Expected output:

`fail-while.err`

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

`fail-while2.xirt`

This test checks a failure of badly defined while loop. Here, the while loop contains a badly formatted condition; ie. the condition does not return a boolean as expected.

```
num main() {
    num i;
    i = 1;
    while(i) {
        if (i == 3) {
            printn(1);
        } else {
            printn(100);
        }
        i = i + 1;
    }
}
```

Expected output:

`Fail-while2.err`

```
Fatal error: exception Failure("expected Boolean expression in i")
```

`test-arithmetic-op.xirt`

This test checks the success of all arithmetic operators. This includes division, multiplication, addition, and subtraction with both negative and positive numbers, and both integer and decimal numbers.

```
num main() {
    num k;

    k = 1 + 9;
    printn(k);

    k = 1 + -9;
    printn(k);

    k = 9 + -4;
    printn(k);

    k = -8 + 2;
    printn(k);

     k = 9 - 4;
    printn(k);

    k = 0 - 0;
    printn(k);

    k = 3 - 4;
    printn(k);

    k = 2 - 2;
    printn(k);

    k = 8 * 2;
    printn(k);

    k = -8 * 2;
    printn(k);

     k = 8.3 * 2;
    printn(k);

    k = 2 * 0;
    printn(k);


    k = -1.3 * 0.9;
    printn(k);

    k = 1.3 * -0.9;
    printn(k);

    k = 1 / 1;
    printn(k);

    k = 1 / 0;
    printn(k);

    k = 1/ 7;
    printn(k);

    k = -1 / -7;
    printn(k);
}
```

Expected output:

`Test-arithmetic-op.out`

```
10
-8
5
-6
5
0
-1
0
16
-16
16.6
0
-1.17
-1.17
1
inf
0.142857
0.142857
```

`test-assignment.xirt`

This test checks the success of proper assignment. Here, a boolean is assigned to “true”, and then re-assigned to an expression (6==8) which evaluates to false.

```
num main(){
    string s;
    num n;
    bool b;
    s="test";
    n=6;
    n=30.2;
    b=true;
    b = (6 == 8);
}
```

Expected output:

`Test-assignment.out`

Note: expected output is blank, with no errors or print statements.

`test-autofill.xirt`

This test checks the success of of auto-filling a matrix using the built-in autofill function. The built in auto-fill function is meant to instantiate a matrix of user provided rows and columns size, with the value the user provides. Here, the user requests a matrix of size 3 by 3, populated with value "1". This test also illustrates our num type, which automatically converts integers to floats on the backend.

```
num main() {
    xirtam m;

    m = autofill(3,3,1);
    printm(m);
}
```

Expected output:

`Test-autofill.out`

```
1.00 1.00 1.00
1.00 1.00 1.00
1.00 1.00 1.00
```

`test-cool-program-2.xirt`

This program takes in a matrix of any size, and replaces an element with a one if there is a one in its row or column. This large-scale test employs many built-ins of Xirtam. It shows:

- num and boolean types;
- control flow via for-loops;
- nested control flow via nested for-loops;
- Control flow of if statements
- Boolean evaluation of conditions within control flow structures
- Built in function: **matset** . This function sets a specific value in a given matrix.
- Built in function: **matget**. This function returns a specific value from a given matrix, given x,y coordinates.
- Nested evaluation of built in functions midget and matset.
- User defined functions declaration, and call of user-defined function.
- Built in function: **printm**, which prints a matrix in an easy to understand format

```
/* https://www.geeksforgeeks.org/a-boolean-matrix-question/ */

/* This function returns a matrix that's identical to s, except that
   an element is replaced with a 1 if there is a 1 in the col or row. */
xirtam populate_1(xirtam s, xirtam ret, num row, num col) {
    num i;
    num j;
    num temp_col;
    num temp_row;
    bool cur_col_has_1;
    bool cur_row_has_1;

    for (i = 0; i < row; i = i + 1) {
        for (j = 0; j < col; j = j + 1) {

            cur_row_has_1 = false;
            cur_col_has_1 = false;

             /* check if current row has a 1 */
             for (temp_col = 0; temp_col < col; temp_col = temp_col + 1) {
                 cur_row_has_1 = cur_row_has_1 || (matget(s, i, temp_col) == 1);
             }

             /* check if current col has a 1 */
             for (temp_row = 0; temp_row < row; temp_row = temp_row + 1) {
                 cur_col_has_1 = cur_col_has_1 || (matget(s, temp_row, j) == 1);
             }

            if ((cur_row_has_1 == true) || (cur_col_has_1 == true)) {
                matset(ret, i, j, 1);
            } else {
                matset(ret, i, j, matget(s, i, j));
            }
        }
    }

    return ret;
}

num main() {
    xirtam s;
    xirtam ret;
    s = [[2, 5, 6, 7], [4, 1, 5, 6]];
    ret = [[0,0,0,0], [0,0,0,0]];
    ret = populate_1(s, ret, 2, 4);
    printm(ret);
}
```

Expected output:

`Test-cool-program-2.out`

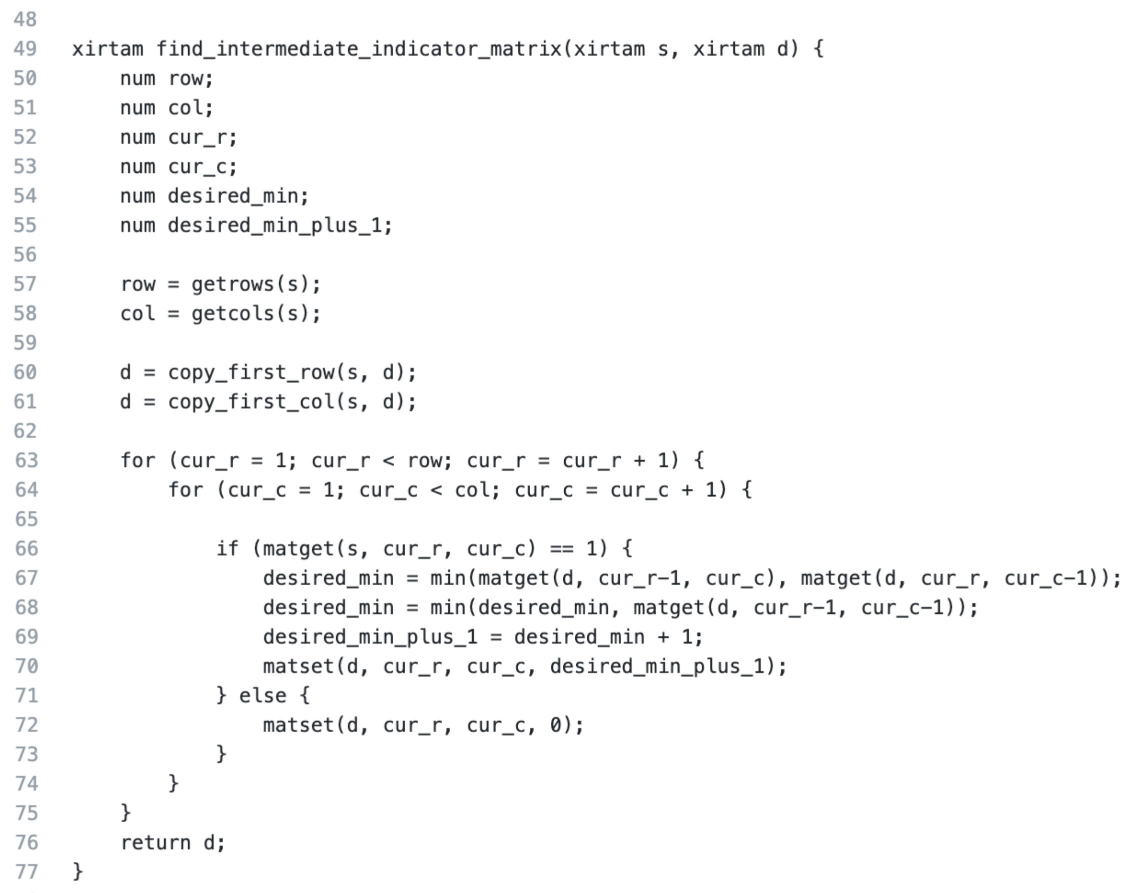
```
2.00 1.00 6.00 7.00
1.00 1.00 1.00 1.00
```

`test-cool-program.xirt`

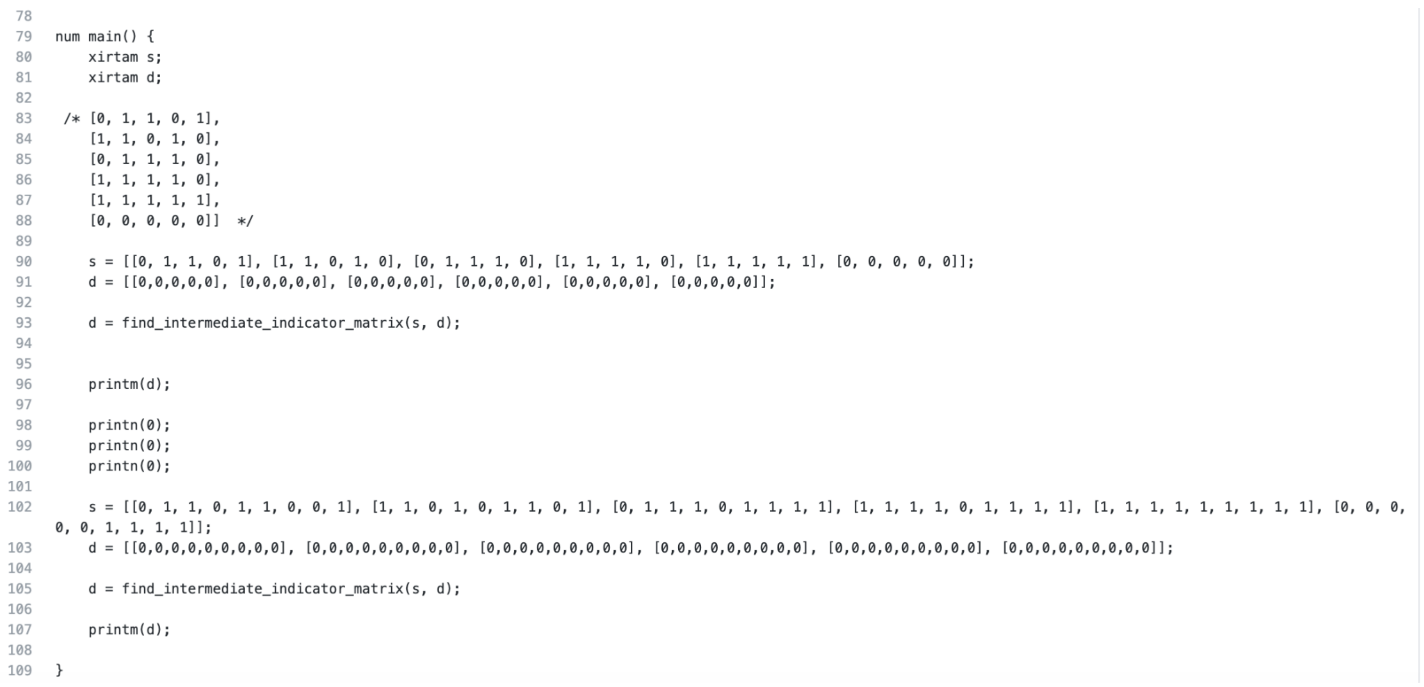
This is the demo program we showed in class. It takes in a binary matrix of any size, and an additional zero matrix of the same size. The output is a matrix whose element indicates the largest square submatrix that only has one's and whose bottom-most right-most element is at that position. This large-scale test employs many built-ins of Xirtam. It features:

- If/else control flow
- Boolean evaluation of conditions
- Built in function: **getcols**. This function returns the number of columns in a matrix.
- For loop control flow
- Built in function: **matset**. This function sets a specific x,y coordinate of a matrix with a value
- User defined function declaration and call
- Built in function: **getrows**. This function returns the number of rows in a matrix.
- Built in function: **matget** . This function gets a specific x,y coordinate of a function
- Built in function: **printn**. This function prints a num type.
- Built in function: **printm.** This function prints a matrix (xirtam) type.
- Direct assignment of matrix without requiring dimensions in advance.

```
/* https://www.geeksforgeeks.org/maximum-size-sub-matrix-with-all-1s-in-a-binary-matrix/ */


/* we only work with 6 by 5 matrices */
num min(num a, num b) {
    if (a <= b) {
        return a;
    } else {
        return b;
    }
}

xirtam copy_first_row(xirtam source, xirtam dest) {

    num val;
    num i;
    num col;

    col = getcols(source);

    for (i = 0 ; i < col ; i = i + 1) {
        val = matget(source, 0, i);
        matset(dest, 0, i, val);
    }

    return dest;

}




xirtam copy_first_col(xirtam source, xirtam dest) {

    num val;
    num i;
    num row;

    row = getrows(source);

    for (i = 0 ; i < row ; i = i + 1) {
        val = matget(source, i, 0);
        matset(dest, i, 0, val);
    }

    return dest;

}
```



```
xirtam find_intermediate_indicator_matrix(xirtam s, xirtam d) {
    num row;
    num col;
    num cur_r;
    num cur_c;
    num desired_min;
    num desired_min_plus_1;

    row = getrows(s);
    col = getcols(s);

    d = copy_first_row(s, d);
    d = copy_first_col(s, d);

    for (cur_r = 1; cur_r < row; cur_r = cur_r + 1) {
        for (cur_c = 1; cur_c < col; cur_c = cur_c + 1) {

            if (matget(s, cur_r, cur_c) == 1) {
                desired_min = min(matget(d, cur_r-1, cur_c), matget(d, cur_r, cur_c-1));
                desired_min = min(desired_min, matget(d, cur_r-1, cur_c-1));
                desired_min_plus_1 = desired_min + 1;
                matset(d, cur_r, cur_c, desired_min_plus_1);
            } else {
                matset(d, cur_r, cur_c, 0);
            }
        }
    }
    return d;
}
```



```

num main() {
    xirtam s;
    xirtam d;

 /* [0, 1, 1, 0, 1],
    [1, 1, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [1, 1, 1, 1, 0],
    [1, 1, 1, 1, 1],
    [0, 0, 0, 0, 0]]  */

    s = [[0, 1, 1, 0, 1], [1, 1, 0, 1, 0], [0, 1, 1, 1, 0], [1, 1, 1, 1, 0], [1, 1, 1, 1, 1], [0, 0, 0, 0, 0]];
    d = [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]];

    d = find_intermediate_indicator_matrix(s, d);


    printm(d);

    printn(0);
    printn(0);
    printn(0);

    s = [[0, 1, 1, 0, 1, 1, 0, 0, 1], [1, 1, 0, 1, 0, 1, 1, 0, 1], [0, 1, 1, 1, 0, 1, 1, 1, 1], [1, 1, 1, 1, 0, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [0, 0, 0,
0, 0, 1, 1, 1, 1]];
    d = [[0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0]];

    d = find_intermediate_indicator_matrix(s, d);

    printm(d);

}
```

Expected output:

`test-cool-program.out`

```
0.00 1.00 1.00 0.00 1.00
1.00 1.00 0.00 1.00 0.00
0.00 1.00 1.00 1.00 0.00
1.00 1.00 2.00 2.00 0.00
1.00 2.00 2.00 3.00 1.00
0.00 0.00 0.00 0.00 0.00
0
0
0
0.00 1.00 1.00 0.00 1.00 1.00 0.00 0.00 1.00
1.00 1.00 0.00 1.00 0.00 1.00 1.00 0.00 1.00
0.00 1.00 1.00 1.00 0.00 1.00 2.00 1.00 1.00
1.00 1.00 2.00 2.00 0.00 1.00 2.00 2.00 2.00
1.00 2.00 2.00 3.00 1.00 1.00 2.00 3.00 3.00
0.00 0.00 0.00 0.00 0.00 1.00 2.00 3.00 4.00
```

`test-determinant.xirt`

This large-scale test is able to find the determinant of a matrix using Xirtam built in functions and matrix types. It shows the usage of:

- Xirtam types
- Num types
- **matget** built in function
- Instantiation of matrix (xirtam type) without predefining size of matrix
- User defined function
- Control flow

The determ_two uses ba

```
num determ_two(xirtam s) {
    num a;
    num b;
    num c;
    num d;

    a = matget(s,0,0);
    b = matget(s,0,1);
    c = matget(s,1,0);
    d = matget(s,1,1);

    return (a*d - c*b);
}


num determ_three(xirtam s) {
    num a;
    num b;
    num c;
    num d;
    num e;
    num f;
    num g;
    num h;
    num i;
    num val;

    xirtam x;
    xirtam y;
    xirtam z;

    a = matget(s,0,0);
    b = matget(s,0,1);
    c = matget(s,0,2);
    d = matget(s,1,0);
    e = matget(s,1,1);
    f = matget(s,1,2);
    g = matget(s,2,0);
    h = matget(s,2,1);
    i = matget(s,2,2);

    x = [[e,f],[h,i]];
    y = [[d,f],[g,i]];
    z = [[d,e],[g,h]];

    val = (a * determ_two(x) - b * determ_two(y) + c * determ_two(z));

    return val;
}

num main() {
    xirtam t;
    xirtam thr;
    num determResult;

    t = [[ 10, 2 ], [ 3, 4 ]];
    determResult = determ_two(t);
    printn(determResult);

    thr = [[ 25, 28 ,35], [ 3, 4, 8 ], [12,10,15]];
    determResult = determ_three(thr);
    printn(determResult);
}
```

Expected output:

`Test-determinant.out`

```
34
298
```

`Test-emptymain.xirt`

This test success of declaring main, with no content inside main. This test should pass without errors.

```
num main(){}
```

Expected output:

`Test-emptymain.out`

Blank output, as no errors are thrown an no prints are utilized. This test should succeed, allowing for an empty main function.

---

`Test-for-c.xirt`

This tests declaring a c-like syntax for loop successfully. This test is expected to pass without errors.

```
num main()
{
  num i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    printn(i);
  }
  printn(42);
}
```

Expected output:

`Test-for-c.out`

```
0
1
2
3
4
42
```

`Test-func-dec.xirt`

This tests the successful creation of user-defined functions, and invocation of user-defined functions. This test should succeed without errors.

```
num add(num a, num b) {
    num c;
    c = a + b;
    return c;
}

bool small_num(num input, num threshold) {
    return (input < threshold);
}

num main() {
    num k;
    num threshold;

    k = add(1, 2);
    printn(k);

    k = 0.34;
    threshold = 1;
    if (small_num(k, threshold)) {
        printn(k);
    } else {
        printn(100000);
    }
}
```

Expected output:

`Test-func-dec.out`

```
3
0.34
```

`Test-getcols.xirt`

This tests the successful usage of a built in function, “getcols”. “Getcols” will return the number of columns in a matrix.

```
num main() {
    xirtam m;
    num result;

    m = [[1,2,1,3],[1,2,2,3],[1,2,2,3]];
    result = getcols(m);

    printn(result);
}
```

Expected output:

`Test-getcols.out`

```
4
```

`Test-getrows.xirt`

This tests the successful usage of the built in “getrows” function. “Getrows” will return the number of rows in a matrix.

```
num main() {
    xirtam m;
    num result;

    m = [[1,2,1,3],[1,2,2,3],[1,2,2,3]];
    result = getrows(m);

    printn(result);
}
```

Expected output:

`Test-getrows.out`

```
3
```

`Test-global-variable.xirt`

This tests the successful creation and usage of global variables.

```
num global;

num temp(num a) {
  global = 2;
  return a + 1;
}

num main()
{
  num i;

  global = 9;
  temp(4);

  i = 5;
  while (i > 0) {
    printn(i);
    i = i - 1;
  }
  printn(42);
  printn(global);
}
```

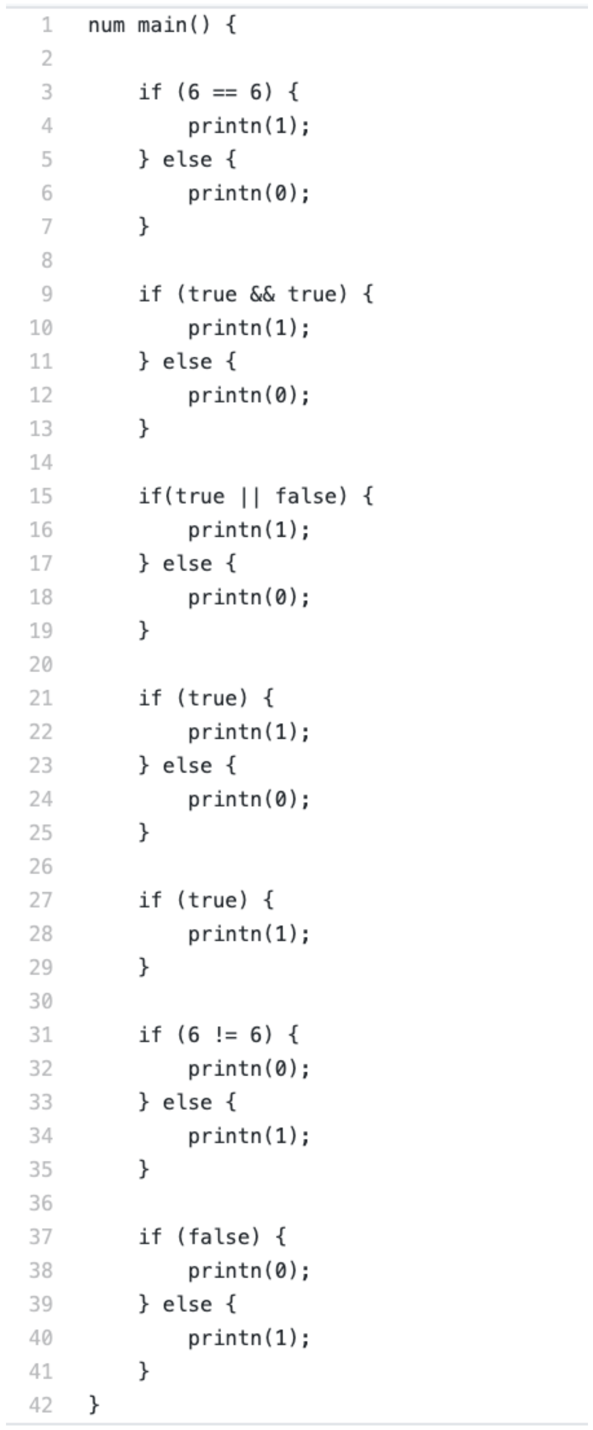
Expected output:

`Test-global-variable.out`

```
5
4
3
2
1
42
2
```

`Test-if.xirt`

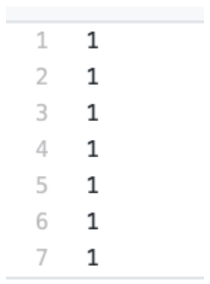
This tests the successful creation and usage of if statements with direct boolean conditionals, and conditionals that evaluate to booleans.



```
num main() {

    if (6 == 6) {
        printn(1);
    } else {
        printn(0);
    }

    if (true && true) {
        printn(1);
    } else {
        printn(0);
    }

    if(true || false) {
        printn(1);
    } else {
        printn(0);
    }

    if (true) {
        printn(1);
    } else {
        printn(0);
    }

    if (true) {
        printn(1);
    }

    if (6 != 6) {
        printn(0);
    } else {
        printn(1);
    }

    if (false) {
        printn(0);
    } else {
        printn(1);
    }
}
```

Expected output:

`Test-if.out`



```
1
1
1
1
1
1
1
```

`Test-matadd-2.xirt`

This tests the successful usage of the built in “matadd” function. This function adds two matrices together. This tests addition of 1x1 matrices with varying values.

```
num min(num a, num b) {
    if (a >= b) {
        return b;
    } else {
        return a;
    }
}

num min_half(num a, num b) {
    num min;
    min = min(a, b);
    return min/2;
}

num main() {
    xirtam m;
    xirtam m2;
    xirtam r;
    m = [[1]];
    m2 = [[1]];
    r = matadd(m, m2);
    printm(r);


    m = [[1]];
    m2 = [[-1]];
    r = matadd(m, m2);
    printm(r);


    m = [[(1 * -1)]];
    m2 = [[(-1 * -1)]];
    r = matadd(m, m2);
    printm(r);

    m = [[min(3, -1)]];
    m2 = [[min(-1, -5)]];
    r = matadd(m, m2);
    printm(r);

    m = [[min_half(3, -1)]];
    m2 = [[min_half(-1, -5)]];
    r = matadd(m, m2);
    printm(r);
}
```

Expected output:

`Test-matadd-2.out`

Please note that the output is representing a 5, 1x1 matrices, which is the result of the addition of the two matrix inputs.

```
2.00
0.00
0.00
-6.00
-3.00
```

`Test-matadd-3.xirt`

This tests the successful usage of the built in “matadd” function. This function adds two matrices together. This tests matrix addition of a matrix that contains integers, a matrix that contains floats, and a matrix that contains negative integers and floats. It should succeed without error.

```
num add_and_get(xirtam m1, xirtam m2, num r, num c) {
    xirtam m3;
    num ret;
    m3 = matadd(m1, m2);
    ret = matget(m3, r, c);
    return ret;
}

num main() {
    xirtam m;
    xirtam m2;
    num r1;
    num r2;
    /* m = [[1, 2, -9.00], [4, 2, -3]]; */
    m = [[-4, 2, 4], [422, 21, 2], [0.4, 6.2, -3]];
    m2 = [[1.01, 2, 0.91], [422, 21, -3], [0.4, 6.2, 32.74]];
    r1 = add_and_get(m, m2, 0, 0);
    r2 = add_and_get(m, m2, 2, 2);
    printn(r1);
    printn(r2);
}
```

Expected output:

`Test-matadd-3.out`

```
-2.99
29.74
```

`Test-matadd.xirt`

This tests the successful usage of the built in “matadd” function. This function adds two matrices together. This is the most basic matrix addition test.

```
num main() {
    xirtam m;
    m = [[1, 2], [3, 4], [5, 6]];
    printm(matadd(m,m));
}
```

Expected output:

`Test-matadd.out`

Please note that the output is representing one 3x2 matrix.

```
2.00 4.00
6.00 8.00
10.00 12.00
```

`Test-matget.xirt`

This tests the successful usage of the built in “matget” function. Matget returns the value at a specific x,y coordinate of a matrix.

```
num main() {
    xirtam m;
    num r;
    m = [[4, 2, 0.9999999], [422, 21, 99], [0.4, 6.2, 0.9999999]];
    r = matget(m, 2, 0);
    printn(r);
}
```

`Test-matget.out`

```
0.4
```

`Test-matmult-2.xirt`

This tests the successful usage of the built in “matmult” function. Matmult returns a new matrix that is the multiplication of two matrix inputs. Here, two matrices of different size with both positive and negative integers are multiplied.

```
num main() {
    xirtam m;
    xirtam n;
    xirtam ret;
    num r;
    m = [[-1]];
    n = [[32.1]];
    ret = matmult(m, n);
    printm(ret);

    m = [[1, 3, -1, -3]];
    n = [[1], [0], [-1], [4]];
    ret = matmult(m,n);
    printm(ret);

    m = [[0, 1]];
    n = [[1], [0]];
    ret = matmult(m, n);
    printm(ret);

    ret = matmult(n, m);
    printm(ret);
}
```

Expected output:

`Test-matmult-2.out`

```
-32.10
-10.00
0.00
0.00 1.00
0.00 0.00
```

`Test-matmult.xirt`

This tests the successful usage of the built in “matmult” function. Matmult returns a new matrix that is the multiplication of two matrix inputs. This is a basic matrix multiplication test.

```
num main() {
    xirtam m;
    xirtam n;
    xirtam ret;
    num r;
    m = [[4, 2], [422, 21], [0.4, 6.2]];
    n = [[1, 2, 3], [0.5, -1.2, 0]];
    ret = matmult(m, n);
    printm(ret);
}
```

Expected output:

`Test-matmult.out`

```
5.00 5.60 12.00
432.50 818.80 1266.00
3.50 -6.64 1.20
```

`Test-matset.xirt`

This tests the successful usage of the built in "matset" function. Matset sets a specific value at a given x,y coordinate in the matrix. It modifies the original matrix and does not return a new matrix.

```
num main() {
    xirtam m;
    num r;
    m = [[4, 2], [422, 21], [0.4, 6.2]];
    matset(m, 2, 0, -1.233);
    r = matget(m, 2, 0);
    printn(r);
}
```

Expected output:

`Test-matset.out`

```
-1.233
```

`Test-print-matrix1.xirt`

This tests the successful usage of the built in “printm” function. The “printm” built in function prints a matrix in a human readable way.

```
num main() {
    xirtam m;
    m = [[1, 2], [3,4], [5,6]];
    printm(m);
}
```

Expected output:

`Test-print-matrix1.out`

The following output represents a single matrix.

```
1.00 2.00
3.00 4.00
5.00 6.00
```

`Test-print-matrix2.xirt`

This tests the successful usage of the built in “printm” function. The “printm” built in function prints a matrix in a human readable way. This tests shows the printing of a matrix with both integer, negative and decimal values.

```
num main() {
    xirtam m;
    m = [[1], [-3], [52.4155]];
    printm(m);
}
```

Expected output:

`Test-print-matrix2.out`

The following output represents a single matrix.

```
1.00
-3.00
52.42
```

`Test-printn.xirt`

This tests the successful usage of the built in “printn” function. The “printn” built in function prints a num type. It prints out only 2 numbers to the right of the decimal point.

```
num main(){
    printn(7);
    printn(-3.14);
}
```

Expected output:

`Test-printn.out`

```
7
-3.14
```

`Test-temp.xirt`

This tests the successful creation of a **xirtam** type, which is a matrix. This test should succeed without errors.

```
num main() {
    xirtam m;
    m = [[91,2], [3,222], [5,6]];
}
```

Expected output:

`Test-temp.out`

Output is blank, as no errors should be thrown and no print statements are invoked.

`Test-transpose.xirt`

This tests the successful usage of the built in “**trans**” function, which transposes a matrix input, and returns the transposed matrix as output. Here, xirtam matrix “m” is being transposed.

```
num main() {
    xirtam m;
    m = [[1, 2], [3, 4]];

    printm(trans(m));
}
```

Expected output:

`Test-transpose.out`

The output is a single 2x2 matrix which represents the transpose of matrix m.

```
1.00 3.00
2.00 4.00
```

`test-unary-and-bool-op.xirt`

This tests the successful usage of both unary and binary operations. In this test, the following operations are invoked. Each return a boolean value internally. Each unary and binary operation is also tested with nested expressions.:

- Less than (<)
- Less than or equal to (<=)
- Greater than (>)
- Greater than or equal to (>=)
- And (&&)
- Equality (==)
- Or (||)

```
num main() {

    if (1 == 1) {
        printn(1);
    }


    if (9.9 > 2) {
        printn(1);
    } else {
        printn(0);
    }


    if (9.9 >= 2) {
        printn(1);
    } else {
        printn(0);
    }

    if (1 >= 1) {
        printn(1);
    } else {
        printn(0);
    }


    if (1 <= 1) {
        printn(1);
    } else {
        printn(0);
    }


    if (3 <= (5 * 6)) {
        printn(1);
    } else {
        printn(0);
    }

    if (!(1 == 3)) {
        printn(1);
    } else {
        printn(0);
    }

    if ((1 == 1) && (2 >= 2)) {
        printn(1);
    } else {
        printn(0);
    }

    if ((1 < 1) || (2 >= 2)) {
        printn(1);
    } else {
        printn(0);
    }


    /* Bad cases */


    if ((1 == 1) && (2 != 2)) {
        printn(0);
    } else {
        printn(1);
    }

    if ((1 < 1) || (2 > 2)) {
        printn(0);
    } else {
        printn(1);
    }


    if (1 > 2) {
        printn(0);
    } else {
        printn(1);
    }


    if (-1.4 > -(2.3 * -5)) {
        printn(0);
    } else {
        printn(1);
    }

    if (-1.4 < (2.3 * -5)) {
        printn(0);
    } else {
        printn(1);
    }

    if (-1.4 >= -(2.3 * -5)) {
        printn(0);
    } else {
        printn(1);
    }

    if (-1.4 <= (2.3 * -5)) {
        printn(0);
    } else {
        printn(1);
    }
}
```

Expected output:

`test-unary-and-bool-op.out`

In this output, 1 indicates a true statement evaluation. (0 would indicate a false statement evaluation)

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

`test-while-c.xirt`

This tests the successful usage of a c-style while loop.:

```
num main()
{
  num i;
  i = 5;
  while (i > 0) {
    printn(i);
    i = i - 1;
  }
  printn(42);
}
```

Expected output:

`test-while-c.out`

```
5
4
3
2
1
42
```

`test-while.xirt`

This tests the successful usage of a while loop.

```
num main() {
    num i;
    i = 0;
    while(i < 10) {
        if (i == 3) {
            printn(1);
        } else {
            printn(100);
        }
        i = i + 1;
    }
}
```

Expected output:

`Test-while.out`

```
100
100
100
1
100
100
100
100
100
100
```

## 8.2 XIRTAM Compiler Code

### Makefile

```
# Citation: microc and Matrx
# authored by: Annie, Lior and Shida

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

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

# "make all" builds the executable

.PHONY : all
all : xirtam.native matrix.o

# "make xirtam.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by including
# packages, enabling warnings
#
# See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

xirtam.native : matrix.bc
	opam config exec -- \
	ocamlbuild -use-ocamlfind xirtam.native -pkgs llvm,llvm.analysis,llvm.bitreader

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
	rm -rf testall.log ocamlllvm *.diff *.ll *.exe *.s *.o *.bc matrix

matrix : matrix.c
	cc -o matrix -DBUILD_TEST matrix.c

matrix.bc : matrix.c
	clang -emit-llvm -o matrix.bc -c matrix.c -Wno-varargs
```

## readme.md

```
![alt text](bin/xirtamLogo.png)

(Logo Designed by Bailey Hwa)

# XIRTAM Programming Language:



Bailey Nozomu Hwa( bnh2128)

Shida Jing( sj2670)

Andrew Peter Yevsey Gorovoy( apg2165)

Annie Wang ( aw3168)
Lior Attias (lra2135)

## About

Xirtam is a language meant to work with matrices, done for Prof. Edwards's 2021 Spring PLT class
```

## ast.ml

```
(* authored by: Bailey Hwa and Shida Jing
citation: microc compiler shown in class
*)


(*unary operations*)
type op_un = Not | Neg


type op_bin = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Great | Geq | And |
Or | Mod


type typ =
  Num
 | Bool
 | String
 | Void
 | Int
 | Xirtam

type expr =
       (*Primitives and expressions*)
       NumLit of float
       | StrLit of string
```

```
        | BoolLit of bool
        | Id of string
        | Unop of  op_un * expr
        | Binop of expr * op_bin * expr
        | Assign of string * expr
        | Call of string * expr list
        (*IMPLEMENT Xirtam specific below*)
  | XirtamLit of expr list
  | Empty


(* bind can be expression too*)
type bind = typ * string * expr


type stmt =
   Block of stmt list
 | Expr of expr
 | Return of expr
 | If of expr * stmt * stmt
 | For of expr * expr * expr * stmt
 | While of expr * stmt (*the while loop is back in business*)


type func_decl = {
   typ : typ;
   f_name : string;
   f_args : bind list; (* formals*)
   f_locals : bind list; (* adding this, making it akin to micro c makes life easier
*)
   f_statements : stmt list;
 }


 (* Pretty-printing functions below:*)


type program = bind list * func_decl list


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


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


let rec string_of_expr = function
 | NumLit(l) -> string_of_float l
 | BoolLit(true) -> "true"
 | BoolLit(false) -> "false"
 | XirtamLit(l) -> "\n[" ^ String.concat "," (List.map string_of_expr l) ^ "]\n"
 | Id(s) -> s
 | StrLit(s) -> s
 (*we use fun instead of function because fun can take in multiple arguments*)
 | Binop(e1, o, e2) ->string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr
e2
 | Unop(o, e) -> string_of_uop o ^ string_of_expr e
 | Assign(v, e) -> v ^ " = " ^ string_of_expr e
 | Call(f, e) -> f ^ "(" ^ String.concat ", " (List.map string_of_expr e) ^ ")"
 | Empty -> ""


let string_of_typ = function
   Num -> "num"
 | Bool -> "bool"
 | String -> "string"
 | Void -> "void"
 | Int -> "int"
 | Xirtam -> "xirtam"


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


let rec string_of_stmt = function
   Block(stmts) -> "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
 | Expr(expr) -> string_of_expr expr ^ ";\n";
```

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


let string_of_tuple x = "(" ^ (fst x) ^ " : " ^ string_of_typ (snd x) ^ ")"


(* variable name pretty print, some vdecl can have type, name, and optional assignment
to expression, hence, _typ,_name, _*)
let string_of_vdecl (_typ, _name, _) =
 string_of_typ _typ ^ " " ^ _name ^ ";\n"


(* Print out argument type and argument identifier *)
let string_of_fdecl fdecl =
 string_of_typ fdecl.typ ^ " " ^
 fdecl.f_name ^ "(" ^ String.concat ", " (List.map (fun (_, f_arg_name, _) ->
f_arg_name) fdecl.f_args) ^


 ")\n{\n"^
 String.concat "" (List.map string_of_vdecl fdecl.f_locals) ^
 String.concat "" (List.map string_of_stmt fdecl.f_statements) ^"}\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)
```

## codegen.ml

```
(* Authored by Bailey Hwa, Shida Jing, and Andrew Gorovoy.Co-debug with Lior Attias. *)



(* Citation: based on MicroC compiler. Citation for past
project Matrx, specifically for defining the matrix type and
defining and calling built-in matrix functions. *)



module L = Llvm
module A = Ast
open Sast


module StringMap = Map.Make(String)


(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
   let context    = L.global_context () in
   let llmem = L.MemoryBuffer.of_file "matrix.bc" in
   let llm = Llvm_bitreader.parse_bitcode context llmem in


   let the_module = L.create_module context "xirtam" in


   (* Get types from the context *)
   (* i1 is for Boolean *)
   let i1_t       = L.i1_type     context (*i1_t = context.i1_type *)
   and float_t    = L.double_type context
   and i32_t      = L.i32_type    context
   and i8_t       = L.i8_type     context
   in
   let char_point_t = L.pointer_type i8_t
   and void_t     = L.void_type   context
   and xirtam_t     = L.pointer_type (match L.type_by_name llm "struct.matrix" with
     None -> raise (Failure "Missing implementation for struct Matrix")
   | Some t -> t)
    in

   let ltype_of_typ = function
       A.Num  -> float_t
   |   A.Bool -> i1_t
   |   A.Void -> void_t
   |   A.String -> char_point_t
   |   A.Int    -> i32_t
   |   A.Xirtam -> xirtam_t
   in
   let global_vars : L.llvalue StringMap.t =
   (* ignore 3rd optional expr*)
   let global_var m (t, n, _) =
     let init = match t with
```

```
        A.Num -> L.const_float (ltype_of_typ t) 0.0
      (* | A.String -> *)
      | _ -> L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals in


  let printf_t : L.lltype =
  L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  (*print num aka a float*)
  let printf_func : L.llvalue =
    L.declare_function "printf" printf_t the_module in


  let printMatrix_t = L.function_type i32_t [| xirtam_t |] in
  let printMatrix_f = L.declare_function "display" printMatrix_t the_module in
  let matrix_init_t = L.function_type xirtam_t [|i32_t ; i32_t|] in
  let matrix_init_f = L.declare_function "initMatrix_CG" matrix_init_t the_module in
  let store_matrix_t = L.function_type xirtam_t [|xirtam_t ; float_t |] in
  let store_matrix_f = L.declare_function "storeVal" store_matrix_t the_module in


  (*get  matrix m*,float r, float c                  r and c get casted in private
calls*)
  let get_matrix_el_t = L.function_type float_t [| xirtam_t;float_t;float_t |] in
  let get_matrix_el_f = L.declare_function "pub_get" get_matrix_el_t the_module in
  (*set  matrix m* float r, float c, float v         r and c get casted in private
calls*)
  let set_matrix_el_t = L.function_type i32_t [| xirtam_t;float_t;float_t;float_t |]
in
  let set_matrix_el_f = L.declare_function "pub_set" set_matrix_el_t the_module in
  (*transpose: matrix  *)
  let transpose_t = L.function_type xirtam_t [| xirtam_t |] in
  let transpose_f = L.declare_function "transpose" transpose_t the_module in


  let getrows_t = L.function_type float_t [|xirtam_t|] in
  let getrows_f = L.declare_function "getrows" getrows_t the_module in


  let getcols_t = L.function_type float_t [|xirtam_t|] in
  let getcols_f = L.declare_function "getcols" getcols_t the_module in
  let mult_matrix_t = L.function_type xirtam_t [|xirtam_t; xirtam_t|] in
  let mult_matrix_f = L.declare_function "matrixMult" mult_matrix_t the_module in
  let add_matrix_t = L.function_type xirtam_t [|xirtam_t; xirtam_t|] in
  let add_matrix_f = L.declare_function "mAdd" add_matrix_t the_module in
  let autofill_t = L.function_type xirtam_t [|float_t; float_t;float_t|] in
  let autofill_f = L.declare_function "autofill" autofill_t the_module in


  (* Define each function (arguments and return type) so we can
      call it even before we've created its body *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
    let function_decl m fdecl =
      let name = fdecl.sf_name
      and formal_types = (* make this filter out the 2nd and 3rd arg*)
  Array.of_list (List.map (fun (t,_,_) -> ltype_of_typ t) fdecl.sf_args)


      in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
      StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions in


(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.sf_name function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in
  let float_format_str = L.build_global_stringptr "%g\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


    (* Allocate space for any locally declared variables and add the
     * resulting registers to our map *)
    and 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
    (*
    we should reshape these to get rid of the optional 3rd value that we don't care
about in the argument list
    when adding local vars or the func vars
    *)
    let filter_args_helper (_type, _var_id, _) = (_type, _var_id) in
    let formatted_args   = List.map filter_args_helper fdecl.sf_args in
    let formatted_locals = List.map filter_args_helper fdecl.sf_locals in
    let formals          = List.fold_left2 add_formal StringMap.empty formatted_args
        (Array.to_list (L.params the_function)) in
    List.fold_left add_local formals formatted_locals
  in


  (* Return the value for a variable or formal argument.
     Check local names first, then global names *)
  let lookup n = try StringMap.find n local_vars
                 with Not_found -> StringMap.find n global_vars
  in
      (* Construct code for an expression; return its value *)
  let rec expr builder ((_, e) : sexpr) = match e with
   SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
          | SNumLit l  -> L.const_float float_t l
    | SStrLit s ->  L.build_global_stringptr s "tmp" builder
    | SId id -> L.build_load (lookup id) id builder
    | SXirtamLit (contents, rows, cols) ->
            let rec expr_list = function
              [] -> []
              | hd::tl -> expr builder hd::expr_list tl
            in
            let contents' = expr_list contents
            in
            let m = L.build_call matrix_init_f [| L.const_int i32_t cols; L.const_int
i32_t rows |] "matrix_init" builder
            in
            ignore(List.map (fun v -> L.build_call store_matrix_f [| m ; v |]
"store_val" builder) contents'); m
```

```
     | SUnop(op, ((t, _) as e)) ->
         let e' = expr builder e in
         (match op with
                 A.Neg when t = A.Num -> L.build_fneg
               | A.Neg                  -> L.build_neg
         | A.Not                  -> L.build_not) e' "tmp" builder
     (*In fashion of microc have typechecking for *)
     | SEmpty -> L.const_int i32_t 0
     | SBinop (e1, op, e2) ->
       let (_typ, _ ) = e1 in (*get type of first expression, semant should have
checked that both types of e1 and e2 should be same*)
       let e1' = expr builder e1 in
       let e2' = expr builder e2 in (match _typ with
       (*this looks much cleaner*)
       (*binary bool operations!*)
         A.Bool -> (match op with
           A.And     -> L.build_and
           | A.Or      -> L.build_or
           | A.Equal   -> L.build_icmp L.Icmp.Eq
           | A.Neq     -> L.build_icmp L.Icmp.Ne
           | _         -> raise (Failure "internal error: semant should have rejected
and/or on float")
         ) e1' e2' "tmp" builder
         (* num operations*)
         | A.Int | A.String | A.Void | A.Num -> (match op with
           A.Add     -> L.build_fadd
             | A.Sub     -> L.build_fsub
             | A.Mult    -> L.build_fmul
             | A.Div     -> L.build_fdiv
             | A.Mod     -> L.build_frem
             (* | A.Exp     -> L.build_call exp_func [| e1'; e2'|] "exp" builder double
check this *)
             | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
             | A.Neq     -> L.build_fcmp L.Fcmp.One
             | A.Less    -> L.build_fcmp L.Fcmp.Olt
             | A.Leq     -> L.build_fcmp L.Fcmp.Ole
             | A.Great -> L.build_fcmp L.Fcmp.Ogt
             | A.Geq     -> L.build_fcmp L.Fcmp.Oge
             | A.And
             | A.Or      -> raise (Failure "internal error: semant should have rejected
and/or on float")
             ) e1' e2' "tmp" builder
       | A.Xirtam -> raise (Failure "cannot use binop on matrices")
       )
     | SAssign (s, e) -> let e' = expr builder e in
                         ignore(L.build_store e' (lookup s) builder); e'
     | SCall ("printn", [e]) ->
         L.build_call printf_func [| float_format_str ; (expr builder e) |]
           "printf" builder
     | SCall ("printm", [e]) ->
       L.build_call printMatrix_f [| (expr builder e) |] "printm" builder
     | SCall ("matmult", [e1; e2]) ->
       L.build_call mult_matrix_f [| (expr builder e1); (expr builder e2) |] "matmult"
builder
     | SCall ("matadd", [e1; e2]) ->
       L.build_call add_matrix_f [| (expr builder e1); (expr builder e2) |] "matadd"
builder
       (* acces and get *)
     | SCall ("matget", [e1; e2; e3;]) ->
       L.build_call get_matrix_el_f [| (expr builder e1); (expr builder e2);(expr
builder e3) |] "matget" builder
     | SCall ("matset", [e1; e2; e3; e4;]) ->
       L.build_call set_matrix_el_f [| (expr builder e1); (expr builder e2);(expr
builder e3);(expr builder e4) |] "matset" builder
     (* transpose *)
     | SCall ("trans", [e]) ->
       L.build_call transpose_f [| (expr builder e) |] "trans" builder
     | SCall ("getrows", [e]) ->
         L.build_call getrows_f [| (expr builder e) |] "getrows" builder
     | SCall ("getcols", [e]) ->
         L.build_call getcols_f [| (expr builder e) |] "getcols" builder
     | SCall ("autofill", [e1; e2; e3;]) ->
           L.build_call autofill_f [| (expr builder e1); (expr builder e2);(expr
builder e3) |] "autofill" builder


     | SCall (f, args) ->
        let (fdef, fdecl) = StringMap.find f function_decls in
        let llargs = List.rev (List.map (expr builder) (List.rev args)) in
        let result = (match fdecl.styp with
                       A.Void -> ""
                     | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list llargs) result builder
   in
   (* *)
   (* LLVM insists each basic block end with exactly one "terminator"
      instruction that transfers control.  This function runs "instr builder"
      if the current block does not already have a terminator.  Used,
      e.g., to handle the "fall off the end of the function" case. *)
   let add_terminal builder instr =
     match L.block_terminator (L.insertion_block builder) with
          Some _ -> ()
     | None -> ignore (instr builder) in
(* statements and control flow*)
let rec stmt builder = function
 SBlock sl -> List.fold_left stmt builder sl
     | SExpr e -> ignore(expr builder e); builder
     | SReturn e -> ignore(match fdecl.styp with
                            (* Special "return nothing" instr *)
                            A.Void -> L.build_ret_void builder
                            (* Build return statement *)
                          | _ -> L.build_ret (expr builder e) builder );
                    builder
     (* are we adding continue?*)
     | SIf (predicate, then_stmt, else_stmt) ->
        let bool_val = expr builder predicate in
  let merge_bb = L.append_block context "merge" the_function in
        let build_br_merge = L.build_br merge_bb in (* partial function *)


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


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


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


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

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


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


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


      (* Implement for loops as while loops *)
      | SFor (e1, e2, e3, body) -> stmt builder
      ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
    in
    (* Build the code for each statement in the function *)
    let builder = stmt builder (SBlock fdecl.sf_statements) in


    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.styp with
        A.Void -> L.build_ret_void
      (* | A.Int -> L.build_ret (L.const_int i32_t 0) *)
      | A.Num -> L.build_ret (L.const_float float_t 0.0)
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in
  List.iter build_function_body functions;
  the_module
```

## matrix.c

```
#include <stdlib.h>

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



// Authored by Bailey Hwa, Shida Jing, and Andrew Gorovoy
// Referenced the past project Matrx. However, we noticed
// that the past project code did not work, and we
// made heavy modifications. At this point, this file is pretty
// much original, except for general skeleton stuff and helper functions.


static void die(const char *message)
{
   perror(message);
   exit(1);
}

struct matrix {
 int num_rows;
 int num_cols;
 double* matrixAddr; // accessed [row][col]
 int buildPosition;
};
typedef struct matrix matrix;


int debug = 0;


double get(struct matrix* m, int r,int c){
 //get m[r][c]
 int kill = 0;
 if (r>((m->num_rows)-1)){
   perror("row index out of range when setting matrix ");
   kill = 1;
 }
 if (c>((m->num_cols)-1)){
   perror("col index out of range when setting matrix ");
   kill = 1;
 }
 if(kill==1){
   die("");
 }
 int idx = c + (r * (m->num_cols));
 return m->matrixAddr[idx];
}


void set( struct matrix* m,int r,int c,double v){
 }

 //set m[r][c] to v
 int kill = 0;
 if (r>((m->num_rows)-1)){
   perror("row index out of range when setting matrix ");
   kill = 1;
 }
 if (c>((m->num_cols)-1)){
   perror("col index out of range when setting matrix ");
   kill = 1;
 }
 if(kill==1){
   die("");
 }
 int idx = c + (r * (m->num_cols));
 m->matrixAddr[idx]=v;
}


double pub_get(struct matrix* m, double r,double c){


 if (r < 0) {
   perror("Row value is less than 0");
   exit(1);
 }
 if (c < 0) {
   perror("Column value is less than 0");
   exit(1);
 }
 return get(m,(int)r,(int)c);
}


void pub_set( struct matrix* m, double r,double c, double v){
 if (r < 0) {
   perror("Row value is less than 0");
   exit(1);
 }
 if (c < 0) {
   perror("Column value is less than 0");
   exit(1);
 }
 set(m,(int)r,(int)c,v);
```

```
}


double getrows(matrix* m) {
 return (double) m->num_rows;
}


double getcols(matrix* m) {
 return (double) m->num_cols;
}


matrix* autofill(double num_cols, double num_rows, double value) {


   if  (   ((int) num_cols < 1 )|| ((int) num_rows < 1)    ) {
     perror("Number of columns or number of rows is not valid.\nRows and columns must be
a positive number.");
     exit(1);
   }


   double* matrixValues = malloc( (int) num_rows * (int) num_cols * sizeof(double*));


   for(int r = 0; r < num_rows; r++) {
     for(int c = 0; c < num_cols; c++) {
       // matrixValues[r + (c * num_rows)]=0;
       matrixValues[c + (r * (int)num_cols)]=(int) value;
     }
   }

 //return a pointer to matrix struct
 matrix* result = malloc(sizeof(struct matrix));
 result->num_cols = num_cols;
 result->num_rows = num_rows;
 result->matrixAddr = matrixValues;
 result->buildPosition = 0;
 return result;

}
```

```
matrix* storeVal(matrix* target, double value) {
   int position = target->buildPosition;
   int curr_row = position / target->num_cols;
   int curr_col = position % target->num_cols;


   if(debug == 1) {
       printf("Storing: %f\n", value);
       printf("in row: %d\n", curr_row);
       printf("in col: %d\n", curr_col);
   }
   target->matrixAddr[position] = value;
   target->buildPosition = target->buildPosition + 1;
   return target;
}


matrix* initMatrix(double* listOfValues, int num_cols, int num_rows) {
 double* matrixValues = malloc(num_rows * num_cols * sizeof(double*));
 if(debug == 1) {
     printf("Building matrix:\n");
     printf("num_rows: %d\n", num_rows);
     printf("num_cols: %d\n", num_cols);
 }


 //set all values in matrix to 0 if list of values is NULL
 if (listOfValues == NULL) {
   for(int r = 0; r < num_rows; r++) {
     for(int c = 0; c < num_cols; c++) {
       matrixValues[c + (r * num_cols)]=0;
     }
   }
 }

 //load values from a list of values
 else {
   for(int r = 0; r < num_rows; r++) {
     for(int c = 0; c < num_cols; c++) {
       int idx = c + (r * num_cols);
       matrixValues[idx]=listOfValues[idx];
     }
   }
 }


 //return a pointer to matrix struct
 matrix* result = malloc(sizeof(struct matrix));
 result->num_cols = num_cols;
 result->num_rows = num_rows;
 result->matrixAddr = matrixValues;
 result->buildPosition = 0;
 return result;
}


matrix* initMatrix_CG( int num_cols, int num_rows) {
   return initMatrix(NULL, num_cols, num_rows);
}


matrix* mAdd(matrix* lhs, matrix* rhs) {
 //check dimensions
 if (lhs->num_rows != rhs->num_rows || lhs->num_cols != rhs->num_cols) {
   perror("Addition size mismatch.");
   perror("Add");
   exit(1);
 }
 int rows = lhs->num_rows;
 int cols= lhs->num_cols;
 matrix *result = initMatrix(NULL, cols, rows);
 for(int i=0; i < rows; i++) {
   for(int j=0; j < cols; j++) {
       double sum = get(lhs,i,j)+get(rhs,i,j);
       set(result,i,j,sum);
   }
 }


 return result;
}
matrix* matrixMult(matrix* lhs, matrix* rhs) {
 //check dimensions//our original code xirtam
 if (lhs->num_cols != rhs->num_rows) {
   die("matrix multiplication dimensions mismatch, must have (AxM)*(MxB)");
 }
 int rows = lhs->num_rows;
 int cols= rhs->num_cols;//(r1xc1)*(r2xc2)
 matrix *result = initMatrix(NULL, cols, rows);
 for(int i=0; i<rows; i++) {
   for(int j=0; j<cols; j++) {
     for (int k=0; k < rhs->num_rows; k++){
       set(result,i,j,get(result,i,j)+(get(lhs,i,k)*get(rhs,k,j)));
     }
   }
 }
 return result;
}
void display(matrix* input) {
   int row = input->num_rows;
   int col = input->num_cols;
   for(int i = 0; i<row; i++) {
       for(int j=0; j<col; j++) {
         if (j == 0) {
           printf("%.2f", get(input,i,j));
         } else {
         printf(" %.2f", get(input,i,j));
         }
       }
       printf("\n");
   }
}


matrix* transpose(matrix* input) {
 //switch rows and cols, get empty(i.e., zeroed matrix of transposed size, then fill)
 int rows = input->num_rows;
 int cols = input->num_cols;
 matrix *result = initMatrix(NULL, rows, cols);
 for(int i=0; i<rows; i++) {
   for(int j=0; j<cols; j++) {
       set(result, j,i, get(input,i,j));
   }
 }
 return result;
}
```

```
#ifdef BUILD_TEST
int main(int argc,char** argv) {
 //run tests of each function
 //initMatrix and display of empty matrix
 printf("\n===========testing empty init========\n");
 matrix *null_matrix=initMatrix(NULL, 2, 2);


printf("\n===========testing list init========\n");
 //initMatrix and display of 2x2 matrix
 double vals1[] = {91, 2, 3, 222, 7, 6};
 double *list1 = vals1;
 matrix *m = initMatrix(list1, 3, 2);


 display(m);
   for( int i = 0; i < 4; i++) {
     //fill first 4 values, i,e., first row as wel as first element of second row
     m = storeVal(m, 5);
     printf("Storing 5: \n");
     display(m);
 }
 printf("\n===========testing public get and set ========\n");
 double r = 1.1; double c = 0;double setval = 151.99;
 printf("get 1,0 of above matrix: %.2f\n", pub_get(m,r,c));
 pub_set(m,r,c,setval);
 printf("set 1,0 of above matrix: \n");
 display(m);


 // //add 2 of the same matrix
 printf("\n===========testing addition========\n");
 double vals1a[] = {1,2,3,4,5,6};
 double *list1a = vals1a;
 matrix *ma = initMatrix(list1a, 2, 3);
 matrix *result_sum = mAdd(ma, ma);
 display(result_sum);//


 // //multiply two matrices
 printf("\n===========testing multiplication========\n");
 double v1[] = {1,2,3,4,5,6};
 double v2[] = {10,11,20,21,30,31};
 matrix *m1 = initMatrix(v1, 3, 2);
 matrix *m2 = initMatrix(v2, 2, 3);
 matrix *result_product = matrixMult(m1, m2);
 // Should yield
 // 140.00 146.00
 // 320.00 335.00
 display(result_product);


 printf("\n===========testing transpose========\n");
 double v1t[] = {1,2,3,4,5,6};
 matrix *m1t = initMatrix(v1t, 2, 3);
 display(transpose(m1t));


 printf("\n Below are Shida testing on weird cases.\n");
 double k1[] = {-4, 2, 4, 422, 21, 2, 0.4, 6.2, -3};
 double k2[] = {1.01, 2, 0.91, 422, 21, -3, 0.4, 6.2, 32.74};
 matrix *n1 = initMatrix(k1, 3, 3);
 matrix *n2 = initMatrix(k2, 3, 3);
 matrix *result_product2 = mAdd(n2, n1);
 // Should yield
 // 140.00 146.00
 // 320.00 335.00
 display(result_product2);


 // Below is Shida testing demo program

 // double source[] = {0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
1, 1, 1, 0, 0, 0, 0, 0};
 // double dest[] = {0,0,0,0,0,0, 0,0,0,0,0,0, 0,0,0,0,0,0, 0,0,0,0,0,0, 0,0,0,0,0,0,
0,0,0,0,0,0};
 // matrix* d = initMatrix(dest, 5, 6);
 // matrix* s = initMatrix(source, 5, 6);



 // int cur_r;
 //   int cur_c;
 //   double desired_min;
 //   double desired_min_plus_1;
 //   d = copy_first_row(s, d);
 //   d = copy_first_col(s, d);


 //   for (cur_r = 1; cur_r < 6; cur_r = cur_r + 1) {
 //       for (cur_c = 1; cur_c < 5; cur_c = cur_c + 1) {

 //           if (get(s, cur_r, cur_c) == 1) {
 //               desired_min = min(get(d, cur_r-1, cur_c), get(d, cur_r, cur_c-1),
get(d, cur_r-1, cur_c-1));
 //               desired_min_plus_1 = desired_min + 1;
 //               set(d, cur_r, cur_c, desired_min_plus_1);


 //               if (cur_r == 4 && cur_c == 3) {
 //                 printf("WE ARE HERE \n");
 //                 printf("%f", desired_min);
 //               }
 //           } else {
 //               set(d, cur_r, cur_c, 0);
 //           }
 //       }
 //   }


 //   display(d);
}
#endif
```

## parser.mly

```
/*
XIRTAM Parser
Citation: microC processor code
*/

%{ open Ast %}
/* Authored  by Bailey Hwa, help by Shida Jing */



/* Tokens: syntax */
```

```
%token PAREN_L PAREN_R CURLY_L CURLY_R SQUARE_L SQUARE_R SEMICOL COMMA
/* Tokens: Operators & literals */
%token ADD SUB TIMES ASSIGN NOT EQ NEQ GT LT LEQ GEQ PERIOD TRUE FALSE DIV MOD
/* Tokens: program flow */
%token AND OR IF ELSE FOR WHILE RETURN NEW DEL NULL
/* Tokens: matrix functions */
%token MAT_FILL MAT_TRANSPOSE MAT_ROWS MAT_COLS MAT_EQ MAT_ADD MAT_MULT_SCALAR MAT_MULT
/* Tokens: Datatypes */
%token NUM BOOL STRING VOID XIRTAM


/*Literals*/
%token <float> NUMLIT
%token <bool> BOOLLIT
%token <string> ID
%token <string> STRLIT
%token EOF


/*Program*/
%start program
%type <Ast.program> program
%nonassoc HTELSE
%nonassoc ELSE
%left ASSIGN
%left COMMA
%left OR
%left AND
%left EQ NEQ
%left GT LT GEQ LEQ
%left ADD SUB
%left TIMES DIV MOD
%right NOT NEG /*boolean negation and negative*/


%%


program:
       decls EOF {$1}

decls:
  /* nothing */ { ([], [])               }
```

```
       | decls var_decl { (($2 :: fst $1), snd $1) }
       | decls func_decl { (fst $1, ($2 :: snd $1)) }


var_decl:
 typ ID SEMICOL               { ($1, $2, Empty) }




var_decl_list:
   /* nothing */    { [] }
 | var_decl_list var_decl { $2 :: $1 }


func_decl:
       typ ID PAREN_L f_args_opt PAREN_R CURLY_L var_decl_list stmt_list CURLY_R
    {{  typ = $1;
        f_name = $2; (*func name, use symboltables*)
        f_args = $4;(*args *)
        f_locals = $7;
        f_statements = List.rev $8  (*statements in function*)}} /*reverse list to
ensure proper ordering*/
f_args_opt:
  /* nothing */  { [] }
       | f_args_list   { List.rev $1 }


f_args_list: /* arg list with types for functinos */
   typ ID                  { [($1,$2,Empty)]     } /*added the Empty because we can
have assignment to expression, but we dont want to in this case*/
       | f_args_list COMMA typ ID { ($3,$4,Empty) :: $1 }


args_opt:
       { [] }
       | args_list   { List.rev $1 }


args_list:
     expr             {    [$1]    }
       | args_list COMMA expr { ($3 :: $1) }
```

```
/*datatypes*/
typ:
   NUM          {Num}
       | BOOL    {Bool}
       | STRING  {String}
       | VOID    {Void}


   /*matrix*/
   | XIRTAM {Xirtam}


stmt_list:
       {[]}
       | stmt_list stmt {$2 :: $1}


stmt: /*all statements must end with semicolon*/


 expr SEMICOL                                         { Expr $1             }
 | RETURN expr_opt SEMICOL                            { Return $2           }
 | CURLY_L stmt_list CURLY_R                          { Block(List.rev $2)   }
 | IF PAREN_L expr PAREN_R stmt %prec HTELSE { If($3, $5, Block([])) }
 | IF PAREN_L expr PAREN_R stmt ELSE stmt    { If($3, $5, $7)       }
 | FOR PAREN_L expr_opt SEMICOL expr SEMICOL expr_opt PAREN_R stmt { For($3, $5, $7,
$9)   }
 | WHILE PAREN_L expr PAREN_R stmt                    { While($3, $5)        }


expr_opt:
       {Empty} /*no expression or something */
       |  expr {$1}


expr:
       TRUE                 { BoolLit(true)          }
       | FALSE                { BoolLit(false)         }
       | STRLIT               { StrLit($1)             }
       | NUMLIT                 { NumLit($1)           }
       | ID                   { Id($1)                 }
   /*matrix*/
   | SQUARE_L   mat   SQUARE_R {XirtamLit($2)}


        | expr ADD            expr { Binop($1, Add,      $3) }
        | expr SUB            expr { Binop($1, Sub,      $3) }
        | expr TIMES      expr { Binop($1, Mult,      $3) }
        | expr DIV            expr { Binop($1, Div,      $3) }
   | expr MOD        expr  {Binop($1, Mod,      $3) }
        | expr EQ         expr { Binop($1, Equal,   $3) }
        | expr NEQ        expr { Binop($1, Neq,      $3) }
        | expr GT         expr { Binop($1, Great,     $3) }
        | expr LT         expr { Binop($1, Less,     $3) }
        | expr GEQ        expr { Binop($1, Geq,      $3) }
        | expr LEQ        expr { Binop($1, Leq,      $3) }
        | expr AND        expr { Binop($1, And,      $3) }
        | expr OR         expr { Binop($1, Or,       $3) }
        | SUB expr %prec  NEG  { Unop(Neg, $2)           } /*minus statement, adding
prec made it work,*/
        | NOT expr             { Unop(Not, $2)           } /*logical negation*/
        | ID ASSIGN expr       { Assign($1, $3)          }
        | ID PAREN_L args_opt PAREN_R { Call($1, $3)     }
        | PAREN_L expr PAREN_R   { $2                    } /*grouping
func1((a+b)(b+c))*/


/*Xirtam matrix*/
mat:
   SQUARE_L        args_list   SQUARE_R          {[XirtamLit(List.rev $2)]}
/*[[1,2,3]]*/
 | SQUARE_L      args_list SQUARE_R COMMA mat  {XirtamLit(List.rev $2)::$5}
/*[[1,2,3], [1,2,3], MAT]*/
```

## sast.ml

```
Open Ast

(* authored by: Bailey Hwa and Shida Jing *)
(* Citation: based on MicroC code *)
```

```
type sexpr = typ * sx
and sx =
       (*Primitives and expressions*)
       SNumLit of float
       | SStrLit of string
       | SBoolLit of bool
 | SXirtamLit of sexpr list * int * int
       | SId of string
       | SUnop of  op_un * sexpr
       | SBinop of sexpr * op_bin * sexpr
       | SAssign of string * sexpr
       | SCall of string * sexpr list
 | SEmpty


type sbind = typ * string * sexpr


type sstmt =
   SBlock of sstmt list
 | SExpr of sexpr
 | SReturn of sexpr
 | SIf of sexpr * sstmt * sstmt
 | SFor of sexpr * sexpr * sexpr * sstmt
 | SWhile of sexpr * sstmt (* adding while loop back*)


(*should make sbind because these should be semantically checked!*)
type sfunc_decl = {
   styp          : typ;
   sf_name       : string;
   sf_args       : sbind list; (* formals*)
   sf_locals     : sbind list; (*add local variables *)
   sf_statements : sstmt list;
 }
(*add this so we can check if variables are initialized or not!*)
type inited = {
   v_type          : typ;
   v_id            : string;
   mutable v_init  : bool;
 }
(* Pretty-printing functions below:*)
```

```
type sprogram = bind list * sfunc_decl list


let rec string_of_sexpr (t, e) =
 "(" ^ string_of_typ t ^ " : " ^ (match e with
 | SNumLit(l) -> string_of_float l
 | SBoolLit(true) -> "true"
 | SBoolLit(false) -> "false"
 | SId(s) -> s
 | SStrLit(s) -> s
 | SXirtamLit(x, r, c) -> "(rows: " ^ string_of_int r ^ ", cols: " ^ string_of_int c ^
") : [" ^ String.concat ", " (List.map string_of_sexpr x) ^ "]"
 (*we use fun instead of function because fun can take in multiple arguments*)
 | SBinop(e1, o, e2) -> string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^
string_of_sexpr e2
 | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
 | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
 | SCall(f, e) -> f ^ "(" ^ String.concat ", " (List.map string_of_sexpr e) ^ ")"
 | SEmpty -> ""
 ) ^ ")"


let rec string_of_sstmt = function
   SBlock(stmts) -> "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
 | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
 | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
 | SIf(e, s, SBlock([])) -> "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
 | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
     string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
 | SFor(e1, e2, e3, s) ->
     "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
 | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s (* we
should implement this I think*)


(* Print out argument type and argument identifier *)
let string_of_sfdecl fdecl =
 string_of_typ fdecl.styp ^ " " ^
 fdecl.sf_name ^ "(" ^String.concat ", " (List.map (fun (_, f_arg_name, _) ->
f_arg_name) fdecl.sf_args) ^
 ")\n{\n" ^
 String.concat "" (List.map string_of_vdecl fdecl.sf_locals) ^
 String.concat "" (List.map string_of_sstmt fdecl.sf_statements) ^
 "}\n"

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

## scanner.mll

```
{ open Parser }
(* authored by: Bailey Hwa, help by Shida Jing *)
(* Citation: MicroC scanner *)


(*digits*)
let digit = ['0'-'9']
let digits = digit+


rule tokenize = parse
 [' ' '\t' '\r' '\n'] { tokenize lexbuf } (* Whitespace/filler*)
(* ---------- COMMENTS ----------- *)
| "/*"     { comment lexbuf }           (* comments *)
(*  Basic syntax  *)
| '('  {PAREN_L}
| ')'  {PAREN_R}
| '{'  {CURLY_L}
| '}'  {CURLY_R}
| '['  {SQUARE_L}
| ']'  {SQUARE_R}
| ';'  {SEMICOL}
| ','  {COMMA}
(*  Operators, both unary and binary  *)
| '+'  {ADD}
| '-'  {SUB}
| '*'  {TIMES}
| '/'  {DIV}
| '='  {ASSIGN }
| "!"  {NOT}
| "==" {EQ}
| "!=" {NEQ}
| '>'  {GT}
| '<'  {LT}
| "<=" {LEQ}
| ">=" {GEQ}
| '.'  {PERIOD}


(* Program flow  *)
| "&&" {AND}
| "||" {OR}
| "if"     {IF}
| "else"   {ELSE}
| "for"    {FOR}
| "while" {WHILE}
| "return" {RETURN}
| "new" {NEW}
| "del" {DEL}
| "NULL" {NULL}


(* Primitive data & function types *)
| "num"    {NUM}
| "bool"   {BOOL}
| "string" {STRING}
| "void"   {VOID}
| "xirtam"   {XIRTAM}


(*  Literals*)
| "true"   {TRUE}
| "false"  {FALSE}
| digits as lex { NUMLIT(float_of_string lex) } (*convert all numbers to float (num
datatype)*)
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lex {NUMLIT(float_of_string
lex) } (*accept floating point numbers with signs*)
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lex {ID(lex)} (*Variable IDS
string and number _*)
| '"' ([^ '"']* as lex) '"' { STRLIT(lex) } (*double quotes with lookahead*)
(*  Xirtam module functions*)
| eof { EOF }
| _ as char { raise (Failure("invalid character detectred: " ^ Char.escaped char)) }(*
raise error *)
```

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

## semant.ml

```
(* authored by: Bailey Hwa and Shida Jing, co-debug with Lior Attias *)
(* Citation: Semantic checking for the MicroC compiler.
 Also referenced past project Matrx, but eventually modified
 everything, so it's pretty much original at this point. *)


open Ast
open Sast


module StringMap = Map.Make(String)
(* Semantic checking of the AST. Returns an SAST if successful,
  throws an exception if something is wrong.
  Check each global variable, then check each function *)


(* global semant functions*)
(* make error msg*)
let make_err er = raise (Failure er) ;;
(*if func is main, make it hidden int type, if it is somehow int and not main, return
error*)
let typ_helper _nm _tp = (match _nm with
     "main" -> Int
     | _ -> (match _tp with
         Int -> make_err ("function "^_nm ^" is of illegal type int")
         |_   -> _tp));;


let check (globals, functions) =
 (* function binding checks, shouldnt have duplicated of that*)
 let check_binds (kind : string) (binds : bind list) =
   List.iter (function
 (Void, b, _) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
     | _ -> ()) binds;
   let rec dups = function
       [] -> ()
     | ((_,n1,_) :: (_,n2,_) :: _) when n1 = n2 ->
   raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
     | _ :: t -> dups t
   in dups (List.sort (fun (_,a,_) (_,b,_) -> compare a b) binds)
 in
 (*  Check global variables *)
 check_binds "global" globals;


(*function declaration, add funcs and appropriate field data to the StringMap, Struct
below matches that of function in ast *)
let built_in_decls =
   let add_bind map_in (_name, _argtype, _ret_type) =
   (* add entry for function name into the string map *)
   StringMap.add _name
     {
       typ = _ret_type; (*type *)
       f_name = _name;
       f_args = (* create list of args *)
         (
           let rec bind_funcs = (function
             [] -> []
             | fst::snd -> (fst, "x", Empty)::(bind_funcs snd))
           in
           bind_funcs _argtype
         );
       f_locals = [];
       f_statements = []
     } map_in
   in List.fold_left add_bind StringMap.empty [
     (*build in functions:  _name, [_argument_types], return types*)
     ("printn", [Num], Void);
     ("printm", [Xirtam], Void);
     ("matmult", [Xirtam; Xirtam], Xirtam);
     ("matadd", [Xirtam; Xirtam], Xirtam);
     (*get and access*)
     ("matget",[Xirtam;Num;Num], Num);
     ("matset",[Xirtam;Num;Num;Num], Void);
     (*transpose*)
     ("trans",[Xirtam], Xirtam);
     ("getrows",[Xirtam], Num);
     ("getcols",[Xirtam], Num);
     ("autofill",[Num;Num;Num], Xirtam);
```

```
    ]
  in
  (* Add function name to symbol table *)
  let add_func map fd =
    let built_in_err = "function " ^ fd.f_name ^ " may not be defined"
    and dup_err = "duplicate function " ^ fd.f_name
    and n = fd.f_name (* Name of the function *)
    in
    (* ensure again that func main is int as well as return type*)
    let _ret = {
      typ = typ_helper n fd.typ;
      f_name = fd.f_name;
      f_args = fd.f_args;
      f_locals = fd.f_locals;
      f_statements = fd.f_statements;
    }
    in
    match fd with (* No duplicate functions or redefinitions of built-ins *)
         _ when StringMap.mem n built_in_decls -> make_err built_in_err
       | _ when StringMap.mem n map -> make_err dup_err
       (* add to prevent user from creating functions with same name as built-in
functions? *)
       | _ when n = "printn"
              || n = "printm"
              || n = "matmult"
              || n = "matadd"
              || n = "matget"
              || n = "matset"
              || n = "trans"
              || n = "getrows"
              || n = "getcols"
              || n = "autofill"
           -> make_err dup_err
       | _ ->  StringMap.add n _ret map
  in
  (* Collect all function names into one symbol table *)
  let function_decls = List.fold_left add_func built_in_decls functions
  in
    (* Return a function from our symbol table *)
  let find_func s =
    try StringMap.find s function_decls
    with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in
```

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


 let check_function func =
 (*check local vars for duplicates!!!! check for arg duplicates and local var
duplicates*)
 check_binds "function argument" func.f_args;
 check_binds "local variable" func.f_locals;
 let check_assign lvaluet rvaluet err =
      if lvaluet = rvaluet then lvaluet else raise (Failure err)
   in
   (*make symboltable have information about type, name, and whether the variable is
initialized or not*)
   let symbols =
     List.fold_left
       (fun _val (_type, _name, _) -> (StringMap.add _name {
         v_type = _type;
         v_id = _name;
         v_init  = false;
         } _val)
       )
       StringMap.empty (globals @ func.f_args @ func.f_locals )
   in
   (* function args only, use to avoid program seeing function args as uninitialized
*)
   let func_arg_symbols =
     List.fold_left
       (fun _val (_type, _name, _) -> (StringMap.add _name {
         v_type = _type;
         v_id = _name;
         v_init  = false;
         } _val)
       )
       StringMap.empty (func.f_args )
     in


    let type_of_identifier s =
     try StringMap.find s symbols
     with Not_found -> raise (Failure ("undeclared identifier " ^ s))
   in
   (* check if vars within expressions where relevant are initialized, return e if ok
     maybe do type conversion here?? idk
     maybe we want to auto set variables to default value if they r not initialized?
*)
   let expr_init_check e_in =
     (*fix the error print*)
     let init_err _i = ("cannot use unitialized variable "^ _i ^" in expression "^
(string_of_expr e_in)) in
     let rec init_check_helper e = match e with
         NumLit  _   -> true
       | BoolLit _   -> true
       | StrLit _ -> true
       | Empty       -> true
       | XirtamLit _ -> true (*double check *)
       | Id i ->
           let var_dat = type_of_identifier i in
           let _ = var_dat.v_init <- true  in
           if (var_dat.v_init = false ) && not (StringMap.mem i func_arg_symbols)
then
               make_err (init_err i)
           else
               true
      | Call(_, args) ->   List.iter (fun _ex -> ignore (init_check_helper _ex)) args;
true
      | Unop (_, ex) -> init_check_helper ex
      | Binop (e1, _, e2)  -> (init_check_helper e1) && ( init_check_helper e2)
      | Assign (id, _) as _exp ->
          let var_dat = type_of_identifier id in
          (*set variable as initialized! we need to have let _ = or it won't work*)
          let _ = var_dat.v_init <- true
          in true
     in
     ignore(init_check_helper e_in); e_in
   in
   (* Return a semantically-checked expression, i.e., with a type *)
   let rec expr = function
        NumLit  l   -> (Num, SNumLit l)
      | BoolLit l   -> (Bool, SBoolLit l)
      | StrLit l -> (String, SStrLit l)
      | Empty       -> (Void, SEmpty)
      | XirtamLit l ->
```

```
        (*get list of row lengths in matrix*)
        let rec mat_length_list _mat_in =  match _mat_in with
          XirtamLit x -> List.length x :: mat_length_list (List.hd x)
          | _ -> []
        in
        (*given list of matrix elements, check type and return error if not*)
        let check_mat_val_type _mat_val=
          let (_typ,_e) = expr _mat_val in
          (match _typ with
            String -> make_err("no strings allowed in matrices!")
            | Bool -> make_err("no booleans allowed in matrices!")
            | Xirtam -> make_err("Xirtam Literals are only allowed in matrices!")
            |  _ -> expr (expr_init_check _mat_val)
          )
        in
        (*turn matrix into flattened single array while checking fo staggered matrix,
i.e., all row must have same col length*)
        let rec check_stagger test_col = function
          XirtamLit hd::tl ->
            let row_len   = List.hd test_col in (*same column we compare it to*)
            let row_check = List.length hd in (*row we need to check*)
            if row_len != row_check then
              make_err ("No staggered Matrices allowed, rows must be same size")
            else
              (check_stagger (List.tl test_col) hd) @ (check_stagger test_col tl)
          (*for individual row, which is list, map expr to each matrix element*)
          | _mat_row -> List.map check_mat_val_type _mat_row
        in
        (*get list containing length of matrix rows *)
        let mat_rc = mat_length_list (XirtamLit l) in
        let _cols_check = List.tl mat_rc in (*get the rest of the cols*)
        let _rows = List.hd mat_rc in (*rows*)
        let _cols = List.hd _cols_check in (*cols*)
        (*map expr to each of the matrix elements*)
          (Xirtam,
            SXirtamLit (check_stagger _cols_check l, _rows, _cols)
          )
      | Id s       ->
          let var_dat = type_of_identifier s in
            (var_dat.v_type, SId s)
      | Call(fname, args) as call ->
          let call = expr_init_check  call in
          let fd = find_func fname in
```

```
         let param_length = List.length fd.f_args in
         if List.length args != param_length then
           raise (Failure ("expecting " ^ string_of_int param_length ^
                           " arguments in " ^ string_of_expr call))
         else let check_call (ft, _) e =
           let (et, e') = expr e in
           let err = "illegal argument found " ^ string_of_typ et ^
             " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
           in (check_assign ft et err, e')
         in
         (*call func*)
         let _fd_func_args = List.map (fun (_type, f_arg_name, _) -> (_type,
f_arg_name) ) fd.f_args in
         let args' = List.map2 check_call _fd_func_args args
         in (fd.typ, SCall(fname, args'))
     | Unop (op, l) as ex ->
       let (t, l') = expr (expr_init_check l) in
         let ty = match op with
           Neg when t = Num -> t
         | Not when t = Bool -> Bool
         | _ -> raise (Failure ("illegal unary operator " ^
                                string_of_uop op ^ string_of_typ t ^
                                " applied to " ^ string_of_expr ex))
       in (ty, SUnop(op, (t, l')))
     | Binop (e1, op, e2) as e ->
         let (t1, e1') = expr (expr_init_check e1)
         and (t2, e2') = expr (expr_init_check e2) in
         (* Based on the MicroC, all binary operators require operands of the same
type,
         However, should we allow type casting between bool and num?
         Someone look into this please
          *)
         let same = t1 = t2 in
         (* Determine expression type based on operator and operand types *)
         let ty = match op with
           Add | Sub | Mult | Div | Mod when same && t1 = Num   -> Num
         | Equal | Neq            when same               -> Bool
         | Less | Leq | Great | Geq
                    when same && (t1 = Num) -> Bool (*castable to bool, should we
like python have string "" = false and "asdfasdf"  be true?*)
         | And | Or when same && t1 = Bool -> Bool
         | _ -> raise (
           Failure ("illegal binary operator " ^
                      string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                      string_of_typ t2 ^ " in " ^ string_of_expr e))
         in (ty, SBinop((t1, e1'), op, (t2, e2')))
     | Assign (id, v) as _exp ->
         let var_dat = type_of_identifier id in
         let _left = var_dat.v_type  in
         let (_right, val') = expr (expr_init_check v) in
         let err =
           "illegal assignment " ^ string_of_typ _left ^ " = " ^ string_of_typ _right
^ " in " ^ string_of_expr _exp
         in
         (*set variable as initialized! we need to have let _ = or it won't work*)
         let _ = var_dat.v_init <- true ;
         in (check_assign _left _right err, SAssign(id, (_right, val')))
   in
   (* check boolean statement*)
   let check_bool_expr e =
     let (t', e') = expr e
     and err = "expected Boolean expression in " ^ string_of_expr e
     in if t' != Bool then raise (Failure err) else (t', e')
   in
   (* Return a semantically-checked statement i.e. containing sexprs *)
   let rec check_stmt = function
       Expr e -> SExpr (expr e)
     | If(p, b1, b2) -> SIf(check_bool_expr (expr_init_check p), check_stmt b1,
check_stmt b2)
     | For(e1, e2, e3, st) ->
   SFor(expr (expr_init_check e1), check_bool_expr (expr_init_check e2), expr
(expr_init_check e3), check_stmt st)
     | While(p, s) -> SWhile(check_bool_expr (expr_init_check p), check_stmt s)
     | Return e -> let (t, e') = expr (expr_init_check e) in
     (match func.f_name with
       (*The user should not give main a return value*)
       "main" -> make_err ("function main should not have a return value!")
       | _ ->
         if t = func.typ then
           SReturn (t, e')
         else
         make_err("return gives " ^ string_of_typ t ^ " expected " ^
      string_of_typ func.typ ^ " in function" ^ string_of_expr e)
     )
     (* A block is correct if each statement is correct and nothing
        follows any Return statement.  Nested blocks are flattened. *)
     | Block sl ->
         let rec check_stmt_list = function
             [Return _ as s] -> [check_stmt s]
           | Return _ :: _   -> raise (Failure "nothing may follow a return")
           | Block sl :: ss  -> check_stmt_list (sl @ ss) (* Flatten blocks *)
           | s :: ss         -> check_stmt s :: check_stmt_list ss
           | []              -> []
         in SBlock(check_stmt_list sl)
   in (* body of check_function, get expression for assignment *)
   let arg_helper (_type,_name,_val) = (_type,_name, expr _val)
   in
   { styp = typ_helper func.f_name func.typ;
     sf_name = func.f_name;
     sf_args = List.map arg_helper func.f_args;
     sf_locals = List.map arg_helper func.f_locals;
     sf_statements = match check_stmt (Block func.f_statements) with
 SBlock(sl) -> sl
     | _ -> raise (Failure ("internal error: block didn't become a block?"))
   }
 in (globals, List.map check_function functions)
```

## testall.sh

```
#!/bin/sh

# Authored by Andrew Gorovoy, Annie Wang.
# Citation: based on MicroC test script.


# Regression testing script for Xirtam
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test
# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"
# Path to the LLVM compiler
LLC="llc"
# Path to the C compiler
CC="cc"
# Path to the xirtam compiler.  Usually "./xirtam.native"
# Try "_build/xirtam.native" if ocamlbuild was unable to create a symbolic link.
```

```
XIRTAM="./xirtam.native"
#XIRTAM="_build/xirtam.native"
# Set time limit for all operations
ulimit -t 30
globallog=testall.log
rm -f $globallog
error=0
globalerror=0
keep=0
Usage() {
   echo "Usage: testall.sh [options] [.xirt files]"
   echo "-k    Keep intermediate files"
   echo "-h    Print this help"
   exit 1
}
SignalError() {
   if [ $error -eq 0 ] ; then
   echo "FAILED"
   error=1
   fi
   echo "  $1"
}
# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
   generatedfiles="$generatedfiles $3"
   echo diff -b $1 $2 ">" $3 1>&2
   diff -b "$1" "$2" > "$3" 2>&1 || {
   SignalError "$1 differs"
   echo "FAILED $1 differs from $2" 1>&2
   }
}
# Run <args>
# Report the command, run it, and report any errors
Run() {
   echo $* 1>&2
   eval $* || {
   SignalError "$1 failed on $*"
   return 1
   }
}
# RunFail <args>
# Report the command, run it, and expect an error
```

```
RunFail() {
   echo $* 1>&2
   eval $* && {
   SignalError "failed: $* did not report an error"
   return 1
   }
   return 0
}
Check() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                             s/.xirt//'`
   reffile=`echo $1 | sed 's/.xirt$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
   echo -n "$basename..."
   echo 1>&2
   echo "###### Testing $basename" 1>&2
   generatedfiles=""
   generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out" &&
   Run "$XIRTAM" "$1" ">" "${basename}.ll" &&
   Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
   Run "$CC" "-o" "${basename}.exe" "${basename}.s" "matrix.c" &&
   Run "./${basename}.exe" > "${basename}.out" &&
   Compare ${basename}.out ${reffile}.out ${basename}.diff
   # Report the status and clean up the generated files
   if [ $error -eq 0 ] ; then
   if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
   fi
   echo "OK"
   echo "###### SUCCESS" 1>&2
   else
   echo "###### FAILED" 1>&2
   globalerror=$error
   fi
}
CheckFail() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                             s/.xirt//'`
   reffile=`echo $1 | sed 's/.xirt$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
   echo -n "$basename..."
   echo 1>&2
   echo "###### Testing $basename" 1>&2
   generatedfiles=""
   generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
   RunFail "$XIRTAM" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
   Compare ${basename}.err ${reffile}.err ${basename}.diff
   # Report the status and clean up the generated files
   if [ $error -eq 0 ] ; then
   if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
   fi
   echo "OK"
   echo "###### SUCCESS" 1>&2
   else
   echo "###### FAILED" 1>&2
   globalerror=$error
   fi
}
while getopts kdpsh c; do
   case $c in
   k) # Keep intermediate files
       keep=1
       ;;
   h) # Help
       Usage
       ;;
   esac
done
shift `expr $OPTIND - 1`
LLIFail() {
 echo "Could not find the LLVM interpreter \"$LLI\"."
 echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
 exit 1
}
which "$LLI" >> $globallog || LLIFail
# if [ ! -f printbig.o ]
# then
#     echo "Could not find printbig.o"
#     echo "Try \"make printbig.o\""
#     exit 1
# fi
if [ $# -ge 1 ]
```

```
then
   files=$@
else
   files="tests/test-*.xirt tests/fail-*.xirt"
fi
for file in $files
do
   case $file in
   *test-*)
       Check $file 2>> $globallog
       ;;
   *fail-*)
       CheckFail $file 2>> $globallog
       ;;
   *)
       echo "unknown file type $file"
       globalerror=1
       ;;
   esac
done
exit $globalerror
```

**xirtam.ml**

```ocaml
(* authored by: Bailey Hwa and Shida Jing *)

(* Citation: MicroC *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./xirtam.native [-a|-s|-l|-c] [file.txt]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

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