Numnum Language Final Report

Programming Languages and Translators

COMS 4115 W Section 1

Prof. Edwards

December 20, 2017

Sharon Chen	syc2138	Tester
Kaustubh Gopal Chiplunkar	kc3148	Language Guru
Paul Czopowik	pc2550	Manager
David Tofu	dat2149	Tester
Art Zuks	az2487	System Architect

1. Introduction

Numnum is a programming language which is based on C and Python languages. It is designed to be a domain specific matrix and array manipulation language. Numnum differs in syntax and encapsulates the best of C and Python and some other common languages to deliver a fun and easier programming experience for a user.

The purpose of the language is to provide a native way to manipulate matrices and arrays. To make matrix manipulation easy, the language features simple syntax to allow basic matrix arithmetic, and includes built in functions for matrix element arithmetic.

An example of a program that can be created in Numnum is one that can manipulate images. For example a program could be written to blur images or remove or adjust color information. Images are made of numbers arranged in matrices, which are multi-dimensional arrays of numbers. Because our language offers a native matrix interface it simplifies implementing libraries that would allow for image manipulation.

2. Language Tutorial

2.1 The Setup

Numnum requires the installation of the OCaml llvm library. Use Ubuntu 16.04 LTS for ease of use. Then download the following packages.

```
sudo apt-get install -y ocaml m4 llvm opam
opam init
opam install llvm.3.6 ocamlfind
eval `opam config env
```

The compiler is called upon by using the numnum.native command and streaming in a file of .num format.

```
./numnum.native < hello_world.num
```

2.2 Code Walkthrough

In this section we will go through a basic code which reads in a colored image and converts it to black and white.

```
int main()
{
  string path;
  string path2;
  int i;
  int j;
  float sum;
  float temp;
  float w1;
  float w2;
  float w3;
  byte[3][600][400] a;
  w1 = 0.2126;
  w2 = 0.7152;
  w3 = 0.0722;
  path = "./cat-stripped.ppm";
  path2 = "./cat-check-bw.ppm";
```

Every numnum program must have a main function. Variables are declared first and then assigned. Arrays are declared with the variables with their type, followed by number of dimensions each enclosed in square brackets. The strings `path` and `path2` are the locations of the the image to be written to and from where to read.

```
read(path, a);
```

The command `read`, reads in the values in the file specified by the string path, and reads them into the variable `a`. The command always tries to read in data of size of `a`, so there can be no out of bound errors.

```
for (i = 0 ; i < 400 ; i = i + 1) {
  for (j = 0; j < 600 ; j = j + 1 ) {
    sum = 0;
    temp = 0;
    temp = w1 * a[0][j][i];
    sum = sum + temp;
    temp = w2 * a[1][j][i];
    sum = sum + temp;
    sum = w3 * a[2][j][i];
    sum = sum + temp;
    sum = sum/3;
    a[0][j][i] = sum;
    a[1][j][i] = sum;
    a[2][j][i] = sum;
  }
}
```

These two for loops iterate over the image and pickup every pixel. Then we perform the weighted sum of the RGB values for the pixel to convert it to grayscale. There are many implicit type conversions which must be understood here.
First, in the line

```
temp = w1 * a[0][j][i];
```

`a` is an array of `byte`s, however it is multiplied by a `float`, hence it is implicitly converted to a `float` and their multiplication is assigned to another `float` temp.

In,

```
sum = sum / 3;
```

The `3` is converted to a `float` again and then assigned to `float sum`.
In the line,

```
a[0][j][i] = sum;
```

`Float  sum` is assigned to a `byte` array, hence `sum` is implicitly casted to a `byte`.

Thus, iterating through the array, we convert the RGB pixels to a grayscale using a weighted conversion.

```
write(path2, a);
  return 0;
}
```

In the end, we write the matrix **`a`** back to the path and complete the conversion. The `write` function is similar to the `read` function, in the sense that it will write all of the size of the array to the specified path.

Also, we return `0`, matching with the function return type.

Something to watch out for while writing code in numnum are the implicit type conversions, even if the compiler won't complain about syntactical errors, you may not actually mean some of those automatic conversions.

3. Language Reference Manual

3.1 Lexical Conventions

3.1.1 White space

White space is used to separate tokens in the language and is otherwise ignored. The programmer is free to use space, tab or newline characters to make code more readable.

3.1.2 Comments

The character `/*` marks the start of a string and the character `*/` marks its end.

3.1.3 Identifiers for Functions and Variables

An identifier is a sequence of letters and digits and the first character must be alphabetic. The underscore _ counts as alphabetic. Upper and lower case letters are considered different.
Declared more formally as : `['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_'  ]*`

3.1.4 Keywords:

- `int`
- `float`
- `string`
- `Byte`
- `void`
- `while`
- `for`
- `if`
- `elif`
- `else`
- `print` (int)
- `printfl` (float)
- `printstrn` (string no `\n`)
- `printstr` (string)
- `printbyte` (byte)
- `printb` (bool)
- `open`
- `write`
- `dim` (# dimensions)
- `return`

3.1.5 Constants

The language contains the following constants:

- integer
- floating point number

- string
- boolean

3.1.5.1 Integer Constants

An integer constant consists of a sequence of digits. The language recognizes decimal numbers only and does not recognize binary, octal, hexadecimal or other number systems. Integer constants are signed by default. To represent a negative integer, the minus sign is used. Leading zeros are ignored.

Example:

```
int a = 456
int b = -12
```

3.1.5.2 Floating Point Constants

Floating point constants consist of the integral part in form of a sequence of digits, a period and a fractional part which is also a sequence of digits. The language recognizes decimal numbers only and does not recognize binary, octal, hexadecimal or other number systems. For the integral part, leading zeros are ignored and the number can be signed with a minus sign.

Example:

```
float a = 456.789
float b = -12.0
```

3.1.5.3 String Constants

A string constant is a sequence of characters enclosed by double quotes `""` and terminated by a null byte `\0` to indicate the end of the string. Strings are not parsed for comments and The backslash `\` is used for escaping characters in the string.

Escape Characters:

- `\` - Escape Character
- `\n` - newline Character
- `\t` - Tab Character
- `\\` - Backslash
- `\"` - Quote

Example:

```
str name = "John Doe";
str x = "10 \t 20 \"Inch\"";
str example = "example string /* this is not a comment */ \"
still in the string"
```

3.2 Syntax

The semicolon ; is a statement terminator.

```
print ("Hello, world!");
```

3.2.1 Code Blocks

Code blocks are enclosed by curly braces { }

3.2.2 Functions

Function has a return type and has arguments. A function cannot return a matrix but can return other data types. Matrices can only be passed by reference in a function.

Syntax:

```
/* Function Declaration */
type name (list of parameters) {
    variable declaration list;
    statement list;
    return statement;
}

/* Function Call */
name (list of parameters);
```

Example:

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

3.2.3 Control Flow

Control flow is achieved by loops and conditional statements.

3.2.3.1 Loops

There is are two ways to implement loops, a `for` loop and a `while` loop:

For Loop Syntax:

```
for (expression; condition expression; increment expression) {
      Statement list;
}
```

While Loop Syntax:

```
while (condition expression) {
      Statement list;
}
```

3.2.3.2 Conditional Statements

Conditional statements are handled by using `if`, `elif` and `else`.

Syntax:

```
if (expression) {
      expression;
} elif (expression) {
      expression;
} else {
      expression;
}
```

3.2.4 Operators

3.2.4.1 Binary Operators

+	Subtraction of two 32-bit int/ 64-bit floats/8 bit byte. Right side gets cast to left type.
-	Subtraction of two 32-bit int/ 64-bit floats/8 bit byte. Right side gets cast to left type.
/	Subtraction of two 32-bit int/ 64-bit floats/8 bit byte. Right side gets cast to left type.
*	Subtraction of two 32-bit int/ 64-bit floats/8 bit byte. Right side gets cast to left type.
==	Equality Check
!=	Inequality Check
>	Greater Than Operator
<	Less Than Operator
>=	Greater Than or Equal Operator
<=	Less Than or Equal Operator
&&	Logical And
||	Logical Or

3.2.4.2 Unary Operators

-	Written before in int/float to make it negative
!	Logical Not

3.2.4.3 Assignment Operators

=	Assigns the right hand value to the variable on the left

3.2.5 Operator Precedence

[ ] {}	Highest
!	
* / %	
+ -	
> < <= >=	
== !=	
&&	
||	
=	Lowest

3.2.6 File IO

There are two functions open and write that control interaction with files.

```
int open(string path,*[] matrix_ptr)
```

Takes in a string to the path of the file and any integer matrix type of any dimension. Internally will open a file descriptor and attempt to read the maximum number of bytes that the matrix will be able to store.

```
int write(string path,*[])
```

Takes in a string to the path of the file and any integer matrix type of any dimension. Internally with call linux creat function to write the bytes of the passed in matrix into the file.

3.2.7 Matrices

Each matrix can have any number of dimensions. Allocation is done in a single contiguous block of memory.

Declaration:

```
int[dim1][dim2]... mat;
float[dim1][dim2]... mat1;
```

```
byte[dim1][dim2]... mat2;
```

3.2.8 Implicit Type Conversion

3.2.8.1 Assignment Casting

Converts the type on the right hand side of a assignment statement to the one it is being assigned to

```
type_1 = type_2; // Converts type2 to type1
```

3.2.8.2 Operator Casting

When binary operations have two different types on each side, numnum casts the type to the right of the operation into the type to the type of the left of the operation and returns the type on the left hand side

```
type_1*type_2; // Converts type_2 to type_1
```

3.3 Standard Matrix Library

Here are some built-in functions in the matrix library:

```
print(expression)
```

Prints the expression as a string to standard output. Accepts strings.

```
dim(matrix)
```

Returns an integer of the dimensions of the input expression.

```
el_add(a, b, c)
```

Element-wise matrix addition. Given matrices `a`, `b`, and `c`, each of the same data type and dimensions, the value of every element in `c` is set to be the sum of the element in `a` and the element in `b`, at the corresponding position in the matrix.

```
el_sub(a, b, c)
```

Element-wise matrix subtraction. Given matrices `a`, `b`, and `c`, each of the same data type and dimensions, the value of every element in `c` is set to be the difference of the element in `a` and the element in `b`, at the corresponding position in the matrix.

```
el_mul(a, b, c)
```

Element-wise matrix multiplication. Given matrices `a`, `b`, and `c`, each of the same data type and dimensions, the value of every element in `c` is set to be the sum of the element in `a` and the element in `b`, at the corresponding position in the matrix.

```
el_div(a, b, c)
```

Element-wise matrix division. Given matrices `a`, `b`, and `c`, each of the same data type and dimensions, the value of every element in `c` is set to be the quotient of the element in `a` and the element in `b`, at the corresponding position in the matrix.

```
bc_add(a, b, c)
```

Broadcasting matrix addition. Given matrices `a`, `b`, and `c`, each of the same data type, `a` having dimensions of `[1]`, and `b` and `c` having the same dimensions that might not be `[1]`, the value of every element in `c` is set to be the sum of that element in `a` and the element in `b` at the corresponding position in the matrix.

```
bc_sub(a, b, c)
```

Broadcasting matrix subtraction. Given matrices `a`, `b`, and `c`, each of the same data type, `a` having dimensions of `[1]`, and `b` and `c` having the same dimensions that might not be `[1]`, the value of every element in `c` is set to be the difference of that element in `a` and the element in `b` at the corresponding position in the matrix.

```
bc_mul(a, b, c)
```

Broadcasting matrix multiplication. Given matrices `a`, `b`, and `c`, each of the same data type, `a` having dimensions of `[1]`, and `b` and `c` having the same dimensions that might not be `[1]`, the value of every element in `c` is set to be the product of that element in `a` and the element in `b` at the corresponding position in the matrix.

```
bc_div(a, b, c)
```

Broadcasting matrix division. Given matrices `a`, `b`, and `c`, each of the same data type, `a` having dimensions of `[1]`, and `b` and `c` having the same dimensions that might not be `[1]`, the value of every element in `c` is set to be the quotient of that element in `a` and the element in `b` at the corresponding position in the matrix.

4. Project Plan

4.1 Processes

For project planning the team relied on a variety of tools to ensure that the project proceeded smoothly and deliverables were submitted on time. After evaluating a handful of web-based project management platforms, the manager has chosen to use freedcamp.com. This was primarily due to its licensing model, ease of use and availability of specific features such as milestones, subtasks, and scheduling, among others. Using freedcamp the manager was able to outline all tasks from the requirements and break them out into separate task groups. These tasks included due dates, priority, assignment to team members and allowed for progress tracking. In addition, the calendar was used to set up reminders for deadlines, homeworks and exams. Throughout the project freedcamp would email the team with progress updates and scheduling reminders.

We also used Google Docs extensively as the main collaboration platform. This was our primary documentation and collaboration tool so anything we discussed or worked on would be written in Google Docs. For example, during each meetings a team member would take meeting notes. This was very useful for review, to see what we agreed upon and for those that may have missed a meeting.

One of our first goals was to finish the “First three tasks” as outlined in the course. First, we discussed and assigned team roles, however these changed slightly in the early stages of the project. Each team member was also required to post their availability for this project along with basic contact information and a short bio as related to the project. Based on all of this information the manager was able to establish as weekly meeting schedule.

The team was also tasked to come up with a handful of ideas for our project before our meeting. Using questions such as “What is the purpose?” or “What are we trying to solve or accomplish?” helped us to establish goals. During our first few meetings we would discuss the ideas and try to narrow down the scope of the project. Once we agreed on our main project trajectory we were then able to narrow down the specifications of our language, which was captured in our team meeting notes.

In order to standardize development and testing and to save time with the setup of the tools the team used the same VM image as the development platform. The VM is an Ubuntu 16.04 (not 14 as it was mistakenly mentioned during the presentation) with all the required tools pre-installed. Members of the team would pick up tasks based on previous meetings and discussions. As development got underway, the team used “Issues” in GitHub to track items that needed to be worked on. As the project progressed more, we used Slack as a chat platform to ask questions or discuss issues during development.

4.2 Style Guide

The team did not implement a standard style guide. Development was done using common styling principles modeled after the style of the Micro-C compiler.

4.3 Timeline

Time	Task	Details
September 14	First 3 tasks	Formed team, Assigned team roles, Scheduled weekly meetings, came up with language idea, created project plan.
September 24	Project Proposal	Deliverable
October 10	Development environment	Setup Git repo, setup and share VM for VMware and VirtualBox
October 22-29	Development	Initial parser, floats, changed Python `def/func` to C style function declaration, `print` functions, Menhir test, test script, strings, hello world.
November 8	Deliverable	Hello World
November 5 - 29	Development	Work on AST, shift reduce errors, arrays, lookup tables, matrix declaration with any type, llvm test, parser complete, additional string testing
November 29 - December 7	Development	`Elif` added, semantic checks, debugging
December 13 - 15	Development	Reading binary data into arrays, added `Byte` datatype, debugging
December 15 - 16	Development	Progress on demo, image manipulation

December 17	Development	Casting and conversion, matrix input and output, more work on `elif` and `else`, work on demo for image manipulation (color, blur, reflections, flips, etc), demo of OCR
December 18	Development	Matrix element-wise operations - multiplication, addition, subtraction, division, including ints and floats, edge detection demo (image)
December 19	Development	Matrix broadcasting operations - multiplication, addition, subtraction, division, including ints and floats
December 20	Development	Project cleanup and final testing

4.4 Team roles and responsibilities

4.4.1 Art Zuks (az2487)

Systems Architect - responsible for compiler architecture, lead developer

4.4.2 Kaustubh Chiplunkar (kc3148)

Language Guru - responsible for language design

4.4.3 David Tofu (dat2149)

Tester - responsible for writing test suites

4.4.4 Paul Czopowik (pc2550)

Manager - responsible for project management, scheduling, deliverables, development environment setup, assisting where needed

4.4.5 Sharon Chen (syc2138)

Tester - responsible for writing test suites and automation, implementing language features, coordinating team efforts

4.5 Development Environment

The development environment was based on using Git for a source repository and a Linux Ubuntu 16.04 LTS Virtual Machine in VMware and VirtualBox format. The VM included all development tools required for the project. The tools in the VM included various compilers and languages including GCC and G++, Python, Ocaml suite with and related Ocaml tools like ocamlyacc and ocamllex, git, menhir, vim, and LLVM.

4.6 Project Log:

Below is the commit log from Git. Team members often collaborated in pairs and submitted as one.

```
90579da     Sharon      Wed Dec 20 16:32:41 2017      make sure every test
corresponds to an output
e1d1c9d     Sharon      Wed Dec 20 15:44:14 2017      cleaned up test script
again
c43c2e2     Sharon      Wed Dec 20 15:34:10 2017      reorganized tester
python script
2d52a97     Sharon      Wed Dec 20 15:15:14 2017      fixed semant: added in
one right parenthesis
11fd0f5     Sharon      Wed Dec 20 15:12:35 2017      Multiplication (#31)
f8f8088     Sharon      Tue Dec 19 11:32:25 2017      Merge pull request #30
from pc2550/multiplication
823f0a9     Sharon      Tue Dec 19 11:31:50 2017      Merge branch 'master'
into multiplication
fdddb9d     Sharon      Tue Dec 19 11:25:21 2017      beginning to add
element-wise logical operators
bbcf303     Sharon      Tue Dec 19 10:27:53 2017      added codegen and
semant for el_add
6f562b0     Sharon      Tue Dec 19 10:07:48 2017      cleaned up codegen for
el_mul
52fc134     Sharon      Tue Dec 19 01:10:07 2017      codegen for float
el_mul
cefac25     Sharon      Tue Dec 19 00:57:12 2017      done with el_mul
codegen
1505a8d     Art Zuks    Mon Dec 18 22:00:11 2017      updated semant
f03a361     Art Zuks    Mon Dec 18 21:42:36 2017      added demos
b983b38     Sharon      Mon Dec 18 18:21:23 2017      tried adding matrix
multiplication
6534175     Art Zuks    Mon Dec 18 16:09:54 2017      added edge detection
demo
9cd72fc     Sharon      Mon Dec 18 00:04:15 2017      finished semant for
el_mul
ed559eb     artzuks     Sun Dec 17 22:42:11 2017      Merge pull request #29
from pc2550/demo2
a20d297     Art Zuks    Sun Dec 17 22:41:15 2017      ocr working
e90faad     artzuks     Sun Dec 17 19:49:15 2017      Merge pull request #28
```

```
from pc2550/demo2
422c5f8     Art Zuks    Sun Dec 17 19:48:15 2017      dog demo
095c5b9     Sharon      Sun Dec 17 18:06:03 2017      Merge pull request #27
from pc2550/elif
844207c     Sharon      Sun Dec 17 17:53:07 2017      fixed elif parser for
no else
91523c2     Sharon      Sun Dec 17 15:44:54 2017      Merge branch 'master'
of https://github.com/pc2550/numnum into elif
100d4bc     Sharon      Sun Dec 17 15:43:12 2017      Merge branch 'master'
of https://github.com/pc2550/numnum into elif
b7c6bf2     Sharon      Sun Dec 17 15:41:44 2017      modifying codegen to
work without else
62955cb     Sharon      Sun Dec 17 15:30:48 2017      testing elif more
93d6f3a     Sharon      Sun Dec 17 13:57:25 2017      Merge pull request #26
from pc2550/elif
6912002     Sharon      Sun Dec 17 13:56:04 2017      Merge branch 'master'
into elif
d0e5df1     Sharon      Sun Dec 17 13:51:24 2017      add semantic checking
for elif
0157fa6     artzuks     Sun Dec 17 13:16:33 2017      Merge pull request #25
from pc2550/intcast
5dc46d6     Art Zuks    Sun Dec 17 13:16:04 2017      added matrix out
abc42a8     Art Zuks    Sun Dec 17 13:14:17 2017      conversion to lefthand
type in binop
a9a01d6     Art Zuks    Sun Dec 17 12:45:48 2017      casting from different
types
0733b44     artzuks     Sat Dec 16 23:01:43 2017      Merge pull request #24
from pc2550/demo1
f8f1105     Art Zuks    Sat Dec 16 18:13:32 2017      delete color from image
43539be     Art Zuks    Sat Dec 16 17:12:15 2017      some progress on demo
80970f8     Art Zuks    Fri Dec 15 23:03:15 2017      reading and adding 2
bytes from files
61d68ce     artzuks     Fri Dec 15 22:28:57 2017      fixed warnings (#23)
dd83392     artzuks     Fri Dec 15 22:15:55 2017      Merge pull request #20
from pc2550/string_tests
f1e592d     artzuks     Fri Dec 15 22:15:23 2017      Merge pull request #21
from pc2550/open
4ab251b     artzuks     Fri Dec 15 22:15:17 2017      Merge branch 'master'
into open
2b21435     artzuks     Fri Dec 15 22:13:50 2017      Merge pull request #22
from pc2550/chars
ef4d1d0     Art Zuks    Fri Dec 15 22:12:54 2017      added bytes
256aa2c     Art Zuks    Wed Dec 13 23:11:09 2017      removed ll file
```

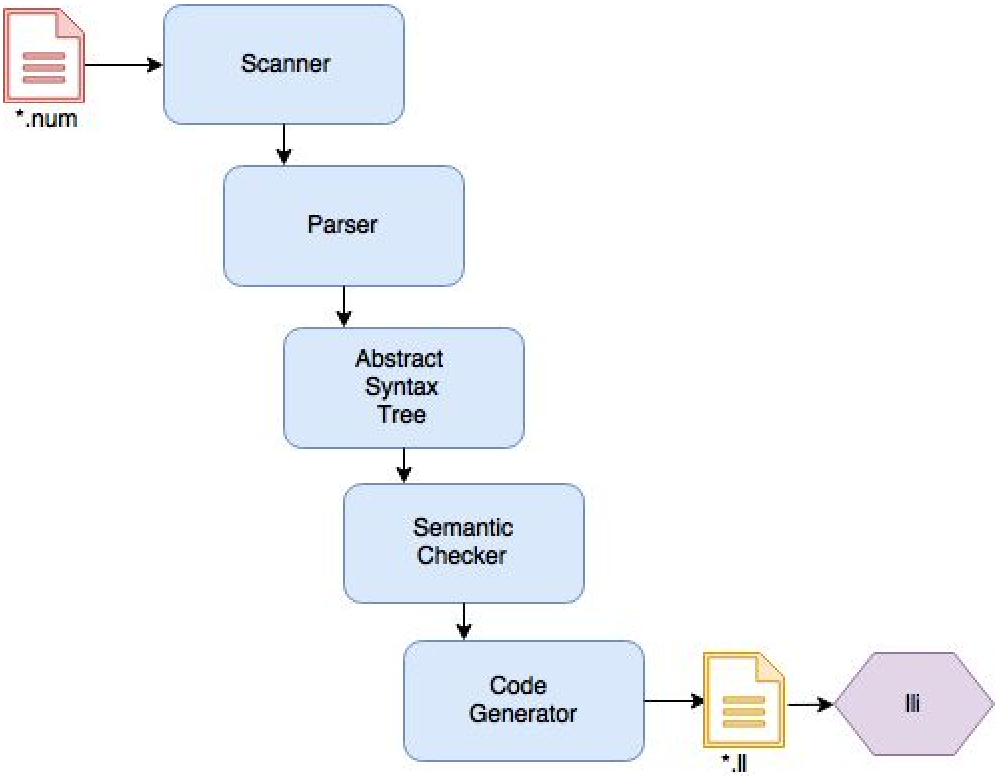
```
9dee9dc     Art Zuks    Wed Dec 13 23:09:14 2017      working reading binary
ints into an array
e90c109     Art Zuks    Thu Dec 7 23:09:16 2017 begin to read bytes from file
a5bf2e3     Sharon      Wed Dec 6 20:03:59 2017 add tests for elif, hello
world, and variables in main
ae39eea     Sharon      Wed Dec 6 19:07:24 2017 elif codegen base case
working with either no else or else stmt
d055a2a     Sharon      Wed Dec 6 18:55:57 2017 elif codegen works with else
base case
fa8d868     Sharon      Wed Dec 6 18:47:52 2017 modify elif codegen for base
case else statement
0067404     Art Zuks    Tue Dec 5 19:20:35 2017 Revert "add ast testers for
elif"
b43c745     artzuks     Tue Dec 5 16:32:08 2017 Merge pull request #18 from
pc2550/matrix
b559cf9     artzuks     Tue Dec 5 16:31:55 2017 Merge pull request #17 from
pc2550/dim_and_shape_of_matrix
29181ae     artzuks     Tue Dec 5 16:31:45 2017 Merge branch 'matrix' into
dim_and_shape_of_matrix
1576b49     DavidTofu   Mon Dec 4 17:30:58 2017 undo unnecssary changes
c33b30e     DavidTofu   Mon Dec 4 17:27:56 2017 Moved out our own tests,
modified testall.sh to run on our tests by default, and on all tests if
needed
0a31938     Sharon      Sun Dec 3 13:58:08 2017 Merge pull request #19 from
pc2550/elif
fd98fc8     Sharon      Sun Dec 3 13:54:17 2017 add ast testers for elif
3121557     Art Zuks    Sun Dec 3 13:05:38 2017 no more warnings
33bc18d     DavidTofu   Fri Dec 1 11:23:57 2017 Fix a warning
1be20fe     DavidTofu   Fri Dec 1 11:03:21 2017 Dim() function done
07204c8     DavidTofu   Fri Dec 1 10:58:11 2017 Pretty printer for matrix
69a030f     Sharon      Wed Nov 29 21:16:03 2017      fixed parser elif for
testing codegen
5a301b7     Sharon      Wed Nov 29 15:31:56 2017      Pretty print elif,
empty semantic check
3de3389     Sharon      Wed Nov 29 14:30:57 2017      Merge branch 'master'
of https://github.com/pc2550/numnum into elif
9d76401     Sharon      Wed Nov 29 14:28:09 2017      Merge pull request #16
from pc2550/string_tests
6855317     Sharon      Wed Nov 29 14:25:01 2017      changed .mc to .num for
running tests
262bc28     Sharon      Wed Nov 29 14:24:32 2017      checked how the testers
failed
361f86b     Sharon      Wed Nov 29 13:40:32 2017      rename extensions from
```

```
.mc to .num
bc66ed0     Sharon      Wed Nov 29 13:31:39 2017      Merge pull request #15
from pc2550/string_tests
dce4fec     DavidTofu   Wed Nov 29 12:55:48 2017      Some more string tests
c4ce06f     Art Zuks    Mon Nov 27 21:12:55 2017      took out foo
2727ff9     Art Zuks    Sun Nov 26 13:50:58 2017      parser done
707ad40     Art Zuks    Sun Nov 26 13:00:12 2017      llvm tests
7936951     Sharon      Tue Nov 21 20:32:28 2017      Merge pull request #14
from pc2550/master
f3dcf32     Sharon      Tue Nov 21 20:28:12 2017      Merge pull request #13
from pc2550/string_tests
6967e7d     Art Zuks    Tue Nov 21 19:15:23 2017      static arrays are done
1f5e754     Art Zuks    Sun Nov 19 15:01:57 2017      access might be
complete
e791c74     Art Zuks    Sun Nov 12 12:49:54 2017      store ast type in
lookup table to get dims for matrix
069c26e     Art Zuks    Sun Nov 12 12:21:19 2017      matrix deceleration
with any type
a2e39a4     Art Zuks    Sun Nov 5 17:57:31 2017 working ast
68af421     Art Zuks    Sun Nov 5 17:12:53 2017 fixed shift reduce
6a4990c     Art Zuks    Sun Nov 5 14:49:59 2017 shift reduce on [
a5afcbd     Art Zuks    Sun Nov 5 11:46:59 2017 formated files
418ac5e     Art Zuks    Sun Nov 5 11:29:37 2017 removed microx from repo
06d4d19     Sharon      Thu Nov 2 12:17:59 2017 added string testers
0a7b71f     Sharon      Tue Oct 31 21:32:12 2017      Merge pull request #4
from pc2550/strings
22c969e     Art Zuks    Sun Oct 29 15:43:11 2017      hello world
b43374c     kaustubh    Sun Oct 29 15:29:59 2017      a
6ad8793     kaustubh    Sun Oct 29 15:27:45 2017      2
d9263cb     kaustubh    Sun Oct 29 14:53:45 2017      a
acb35a3     kaustubh    Sun Oct 29 14:50:17 2017      strings
8e64927     artzuks     Sun Oct 29 13:38:28 2017      Merge pull request #3
from pc2550/test_script
e20796b     kaustubh    Sun Oct 29 13:29:56 2017      test script
9954b35     artzuks     Sun Oct 29 13:17:13 2017      Merge pull request #1
from pc2550/floats
8cfb183     kaustubh    Sun Oct 29 13:15:49 2017      Menhir Test for parser
828ff53     Art Zuks    Sat Oct 28 16:06:14 2017      added operations for
floats
0c52dd9     Art Zuks    Sat Oct 28 12:24:31 2017      make print function and
added some tests
43733cd     Art Zuks    Sat Oct 28 11:03:10 2017      took out func for now
and fixed tests
```

```
c58386b     Art Zuks    Sun Oct 22 13:55:12 2017      floats done
0952fa0     Art Zuks    Sun Oct 22 13:26:13 2017      float stuff
4e27244     Art Zuks    Sun Oct 22 12:35:24 2017      initial parser
bd19343     Paul Czopowik     Mon Oct 9 19:10:39 2017 adding microc-llvm
3ef6f85     Paweł Czopowik    Sun Oct 8 12:43:49 2017 Initial commit
```

5. Architectural Design

5.1 Compiler Diagram



5.2 Scanner

Worked on by Art and Chip.

The scanner is responsible for taking in the input of a program and generating the tokens which will be read in the parser. During this phase, all of the white spaces are taken out and tokens are generated for anything that has syntactic meaning in the language. This includes all of the variable names, any braces or brackets as well as the string,integer and float literals. Everything that is within a comment block (uses regular c-style syntax `/**/`) is discarded at this step.

5.3 Parser

Matrix and types worked on by Art and Chip.
Elif flow control worked on by Art and Sharon.
Matrix arithmetic worked on by Sharon.

The parsers job is to receive the stream of tokens out of the scanner, and construct an abstract syntax tree out of the stream. Most of the overall design remains the same as MicroC compiler. The program is a series of declarations which can be variable declarations (globals) or function declarations. Function declarations are as you would expect in C with the additional caveat that variable declarations and statements must appear separately, one before the other.

5.4 Semantic Checking

Team wide effort.
Matrix and type checking worked on by Art and Chip.
Elif, Matrix Arithmetic done by Sharon.
Element-wise matrix multiplication done by Sharon and David.

The semantic checker is responsible for walking through the AST that was generated by the parser and make sure that the input file isn't violating any syntactic rules. Where the parser was able to complain when it found a missing bracket or brace, the semantic checker is able to tell the user when they are doing something not supported by the user such as assigning a `string` literal into a `int` type. It is also responsible for a table of variable names and functions (symbol table) so that it can complain if a program is trying to access an undeclared variable or function. It also contains a list of all predefined functions in the language and will complain when the parameters don't match in a function call.

5.5 Code Generation

Matrix access/assignment and types by Chip and Art.
File IO and implicit type casting by Art.
Elif control flow and matrix arithmetic by Sharon.
Element-wise matrix multiplication done by Sharon and David.

The code generator walks the freshly checked AST from the semantic checker and tries to translate the nodes into llvm. It is responsible for making sure that the generated llvm code is valid. For instance when doing binary operations between two unevenly sized numbers (32 bit integer and 64 bit float), it makes sure to convert the left hand side to the proper size before doing the binary operation. Also for file IO, it makes sure to check the type of matrix to know how many bytes to read from a file. When processing `Elif` statements, the code generator actually creates new AST nodes that it processes to make the condition statements properly.

6. Test Plan

6.1 Example Test Programs

This section starts off with three representative Numnum programs, along with their generated LLVM code. Right below each program is the expected/actual output of the programs.

This first program checks to see if the first `elif` condition that evaluates to true is run and the later `elif` statements are skipped.

Input: tests/test-elif17.num

```
int cond(bool b)
{
  int x;
  if (false)
    x = 42;
  elif (b) /* because this is an if statement whose condition evaluates to true, the below elif statement is skipped */
    x = 95;
  elif (b)
    x = 423;
  elif (b)
    x = 500;
  else
    x = 600;
  return x;
}

int main()
{
 print(cond(true));
 return 0;
}
```

LLVM code: tests/test-elif17.ll

```
; ModuleID = 'NumNum'

@errno = available_externally global i32 0
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%x\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.4 = private unnamed_addr constant [3 x i8] c"%s\00"
@fmt.5 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%x\0A\00"
@fmt.7 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.9 = private unnamed_addr constant [3 x i8] c"%s\00"

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

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

declare i32 @read(i32, i32*, i32, ...)

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

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

declare i32 @close(i32, ...)

define i32 @main() {
entry:
  %cond_result = call i32 @cond(i1 true)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8]
  ret i32 0
}

define i32 @cond(i1 %b) {
entry:
  %b1 = alloca i1
  store i1 %b, i1* %b1
  %x = alloca i32
  br i1 false, label %then, label %else

merge:                                            ; preds = %merge3,
%then
  %x14 = load i32, i32* %x
  ret i32 %x14

then:                                             ; preds = %entry
  store i32 42, i32* %x
  br label %merge

else:                                             ; preds = %entry
  %b2 = load i1, i1* %b1
  br i1 %b2, label %then4, label %else5

merge3:                                           ; preds = %merge7,
%then4
  br label %merge

then4:                                            ; preds = %else
  store i32 95, i32* %x
  br label %merge3

else5:                                            ; preds = %else
  %b6 = load i1, i1* %b1
  br i1 %b6, label %then8, label %else9

merge7:                                           ; preds = %merge11,
%then8
  br label %merge3

then8:                                            ; preds = %else5
  store i32 423, i32* %x
  br label %merge7

else9:                                            ; preds = %else5
  %b10 = load i1, i1* %b1
  br i1 %b10, label %then12, label %else13

merge11:                                          ; preds = %else13,
%then12
  br label %merge7

then12:                                           ; preds = %else9
  store i32 500, i32* %x
  br label %merge11

else13:                                           ; preds = %else9
  store i32 600, i32* %x
  br label %merge11
}
```

Output: tests/test-elif17.out

```
95
```

The results of the `elif` test above confirmed that the first `elif` statement for which the condition is satisfied is the statement in which `x` is defined, and not any other statements.

The following tester is more comprehensive than the above test. This tester only passed after our language was capable of float matrix initialization, matrix assignment, matrix access, printing of floats, and the four different operations of element-wise arithmetic of matrices.

Input: tests/test-matrix6.num

```

int main(){
    float [2][1] a;
    float [2][1] b;
    float [2][1] c;

    a[0][0] = 2.0;
    a[1][0] = 4.0;
    b[0][0] = 3.0;
    b[1][0] = 3.0;
    c[0][0] = 1.0;
    c[1][0] = 1.0;

    el_sub(a, b, c);

    printfl(c[0][0]);
    printfl(c[1][0]);

    el_add(a, b, c);

    printfl(c[0][0]);
    printfl(c[1][0]);

    el_mul(a, b, c);

    printfl(c[0][0]);
    printfl(c[1][0]);

    el_div(a, b, c);

    printfl(c[0][0]);
    printfl(c[1][0]);

    return 0;
}
```

LLVM code: tests/test-matrix6.ll

```
; ModuleID = 'NumNum'

@errno = available_externally global i32 0
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%x\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.4 = private unnamed_addr constant [3 x i8] c"%s\00"

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

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

declare i32 @read(i32, i32*, i32, ...)

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

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

declare i32 @close(i32, ...)

define i32 @main() {
entry:
  %a = alloca [2 x double]
  %b = alloca [2 x double]
  %c = alloca [2 x double]
  %tmp = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 0
  store double 2.000000e+00, double* %tmp
  %tmp1 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 1
  store double 4.000000e+00, double* %tmp1
  %tmp2 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 0
  store double 3.000000e+00, double* %tmp2
  %tmp3 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 1
  store double 3.000000e+00, double* %tmp3
  %tmp4 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
  store double 1.000000e+00, double* %tmp4
  %tmp5 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
  store double 1.000000e+00, double* %tmp5
  %tmp6 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 0
  %tmp7 = load double, double* %tmp6
  %tmp8 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 0
  %tmp9 = load double, double* %tmp8
```

```
%tmp10 = fsub double %tmp7, %tmp9
%tmp11 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
store double %tmp10, double* %tmp11
%tmp12 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 1
%tmp13 = load double, double* %tmp12
%tmp14 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 1
%tmp15 = load double, double* %tmp14
%tmp16 = fsub double %tmp13, %tmp15
%tmp17 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
store double %tmp16, double* %tmp17
%tmp18 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
%tmp19 = load double, double* %tmp18
%printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8]
%tmp20 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
%tmp21 = load double, double* %tmp20
%printf22 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i
%tmp23 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 0
%tmp24 = load double, double* %tmp23
%tmp25 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 0
%tmp26 = load double, double* %tmp25
%tmp27 = fadd double %tmp24, %tmp26
%tmp28 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
store double %tmp27, double* %tmp28
%tmp29 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 1
%tmp30 = load double, double* %tmp29
%tmp31 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 1
%tmp32 = load double, double* %tmp31
%tmp33 = fadd double %tmp30, %tmp32
%tmp34 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
store double %tmp33, double* %tmp34
%tmp35 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
%tmp36 = load double, double* %tmp35
%printf37 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i
%tmp38 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
%tmp39 = load double, double* %tmp38
%printf40 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i
%tmp41 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 0
%tmp42 = load double, double* %tmp41
```

```
  %tmp43 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 0
  %tmp44 = load double, double* %tmp43
  %tmp45 = fmul double %tmp42, %tmp44
  %tmp46 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
  store double %tmp45, double* %tmp46
  %tmp47 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 1
  %tmp48 = load double, double* %tmp47
  %tmp49 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 1
  %tmp50 = load double, double* %tmp49
  %tmp51 = fmul double %tmp48, %tmp50
  %tmp52 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
  store double %tmp51, double* %tmp52
  %tmp53 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
  %tmp54 = load double, double* %tmp53
  %printf55 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i
  %tmp56 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
  %tmp57 = load double, double* %tmp56
  %printf58 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i
  %tmp59 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 0
  %tmp60 = load double, double* %tmp59
  %tmp61 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 0
  %tmp62 = load double, double* %tmp61
  %tmp63 = fdiv double %tmp60, %tmp62
  %tmp64 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
  store double %tmp63, double* %tmp64
  %tmp65 = getelementptr [2 x double], [2 x double]* %a, i32 0, i32 1
  %tmp66 = load double, double* %tmp65
  %tmp67 = getelementptr [2 x double], [2 x double]* %b, i32 0, i32 1
  %tmp68 = load double, double* %tmp67
  %tmp69 = fdiv double %tmp66, %tmp68
  %tmp70 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
  store double %tmp69, double* %tmp70
  %tmp71 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 0
  %tmp72 = load double, double* %tmp71
  %printf73 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i
  %tmp74 = getelementptr [2 x double], [2 x double]* %c, i32 0, i32 1
  %tmp75 = load double, double* %tmp74
  %printf76 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i
  ret i32 0
}
```

Output: tests/test-matrix6.out

```
1.000000
1.000000
-1.000000
1.000000
5.000000
7.000000
5.000000
7.000000
6.000000
12.000000
0.666667
1.333333
```

The next test was one of the earliests tests written. It was an extension of a Micro-C test, which allowed us to check that our new `string` type could indeed be a global variable.

Input: tests/test-cast1.num

```
int main()
{
  byte a;
  byte c;
  int b;
  b = 3;
  a = b;
  c = 5;
  printbyte(a);
  printbyte(c);
  return 0;
}
```

LLVM code: tests/test-cast1.ll

```
; ModuleID = 'NumNum'

@errno = available_externally global i32 0
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%x\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%f\0A\00"
@fmt.3 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@fmt.4 = private unnamed_addr constant [3 x i8] c"%s\00"

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

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

declare i32 @read(i32, i32*, i32, ...)

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

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

declare i32 @close(i32, ...)

define i32 @main() {
entry:
  %a = alloca i8
  %c = alloca i8
  %b = alloca i32
  store i32 3, i32* %b
  %b1 = load i32, i32* %b
  %conv = trunc i32 %b1 to i8
  store i8 %conv, i8* %a
  store i8 5, i8* %c
  %a2 = load i8, i8* %a
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8]
  %c3 = load i8, i8* %c
  %printf4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4
x i8
  ret i32 0
}
```

Output: tests/fail-global1.err

```
Fatal error: exception Failure("illegal void global a")
```

6.2 Test Suites

For each new feature we implemented, we created multiple test programs. Here are possible program extensions and what they signify:

1. .num: the source language program which may be good or faulty
2. .out: the expected printed output of a Numnum program
3. .err: the expected error message of a faulty Numnum program

These test cases were placed into test suites. There are four large test suites for our translator, each of which contains tests for various Numnum language features:

1. menhir_tests: preliminary tests for checking the abstract syntax tree
2. old_tests: older tests that retained from the Micro-C test suite
3. tests: tests specifically for the Numnum language
4. future_tests: a directory of tests for future use, tests for future Numnum feature implementations

Now, we will explore the details of the main test suite, tests. First off, it contains tests for each of the following language features that we implemented:

1. Types
   - Integers
   - Floats
   - Booleans
   - Bytes
   - Strings
2. Variables
   - Assignment
   - Scope
3. Control Flow
   - For and while loops
   - If, Elif, Else, and Else if
4. Matrices
   - Assignment
   - Access
   - Arithmetic operations

Finally, this test suite also contains the tests for our extensive image processing and optical character recognition demonstrations. As input the image manipulation programs, ppm files are also in the test suite.

6.2.1 Reasoning Behind the Test Cases

We tested throughout the development of the language, across all stages of the compiler pipeline. There were tests specifically written for each stage independent of other stages of the pipeline, i.e. tests for the codegen, tests for the semantic checker, tests for the AST, tests for the scanner, and tests for the parser. The parts that were implemented first were the ones that were tested first. In addition, we took advantage of the fact that by passing in the different compiler flags available, we were also able to test the ast pretty-printing without interfering with the other stages.

Our approach to testing involved thinking about edge cases and ensuring that everything worked as expected.

6.2.2 Test Automation

Because of our detail-oriented approach, we ended up writing numerous tests for even small features. There were too many test programs for each feature that we wanted to test, so we resorted to test automation. We now have two python scripts for automatic testing:

1. tester.py: A python script was written and executed to run all the tests for specific features. Every execution of the script resulted in long and detailed messages printed on the console, which displayed the code that was run, the output of the code, and the expected output of the code.
2. demo-tester.py: A second python script was written for testing demos.

tester.py

```
from subprocess import call
import glob
import sys
import os, errno

"""
Sharon Chen
December 20, 2017
tester.py
This program tests numnum features that have tests in the tests directory.

usage: python tester.py <feature> <show_code>
"""

```

```python

feature = sys.argv[1]
show_code = sys.argv[2].lower() == "true"


def main():

    test_sources = glob.glob("tests/*" + feature + "*")
    try:
        os.makedirs("tests/" + feature)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise

    tests = [test.split(".")[0].split("/")[1] for test in test_sources if ".num" in test]
    tests.sort()
    want_passes = []
    want_fails = []
    for test in tests:
        if "test" in test:
            want_passes.append("tests/" + test)
        elif "fail":
            want_fails.append("tests/" + test)

    print "=========================================="
    print "We are now testing this feature: " + feature
    print "------------------------------------------"


    print "=========================================="
    print "Here are the tests that should be passing: "
    print "------------------------------------------"
    print want_passes

    for test in want_passes:
        try:
            run_test(test, True)
        except:
            continue

    print "=========================================="
    print "Here are the tests that should be failing: "
    print "------------------------------------------"
    print want_fails

    for test in want_fails:
        try:
            run_test(test, False)
        except:
            continue


def run_test(test, want_pass):
    """Run this one test, which either should pass or fail."""

    print "__________________________________________"
    print test
    print "``````````````````````````````````````````"

    if show_code:
        print "Here is the code: "
        call(["cat", test + ".num"])

    in_f = open(test + ".num", "r")
    out_f = open(test + ".ll", "w")
    call(["./numnum"], stdin=in_f, stdout=out_f)
    print ""
    print "Running: " + test + ".num"
    call(["lli", test + ".ll"])
    print ""
    print "Expected output: " + test + ".out"

    if want_pass:
        ext = ".out"
    else:
        ext = ".err"

    call(["cat", test + ext])
    print ""
    print "End of test for " + test
    in_f.close()
    out_f.close()

```

```
    os.remove(test + ".ll")

main()
```

Representative Example Snippet of Output:

```
==========================================
We are now testing this feature: elif
------------------------------------------
==========================================
Here are the tests that should be passing:
------------------------------------------
['tests/test-elif1', 'tests/test-elif13', 'tests/test-elif14',
'tests/test-elif16', 'tests/test-elif17', 'tests/test-elif2',
'tests/test-elif3', 'tests/test-elif4', 'tests/test-elif6',
'tests/test-elif8']

...
__________________________________________
tests/test-elif3
``````````````````````````````````````````

Running: tests/test-elif3.num
42
17

Expected output: tests/test-elif3.out
42
17

End of test for tests/test-elif3

…
==
Here are the tests that should be failing:

['tests/fail-elif1', 'tests/fail-elif2', 'tests/fail-elif3']
____
...
```

demo_tester.py
```

```
import sys
from subprocess import call

"""
usage: python demo-tester.py <effect>
"""
filepath = 'cat.ppm'
output = open("cat-stripped.ppm","w")
fileFormat = ""
dims = ""
maxVal = ""
with open(filepath) as fp:
        fileFormat = fp.readline()
        dims = fp.readline()
        maxVal = fp.readline()
        line = fp.readline()
        while line:
                output.write(line)
                line = fp.readline()
effect = sys.argv[1]
call(['sh', './testall.sh', './tests/demo-' + effect + '.num'])
with open('cat-check-' + effect + '.ppm', 'r') as original: data = original.read()
with open('cat-check-' + effect + '.ppm', 'w') as modified: modified.write(fileFormat + dims + maxVal + data)

```

6.2.3 Who Did What

Sharon kicked off the creation of the test suite. After that, the work on testing began to become more based on who was developing what parts of the language, so everyone was involved in creating the extensive test suite. Sharon, David, Art, Chip, and Paul all wrote varying numbers of tests, depending on how many features they implemented in the language and how rigorous and detail-oriented they were in their language implementation approach. Eventually, Art and Chip created and tested demos by writing up several specific scripts in Numnum, python, and C++. Finally, Sharon created the ultimate automation and organization of the test suite for both the feature test cases and the demo tests.

7. Lessons Learned

7.1 Art

Some main takeaways from the project is OCaml and LLVM IR code. Even though I had a slow start with OCaml, it eventually beat me into submission and once stockholm syndrome kicked in, I really began to like the language. It lead to me finding out one of my ex-colleagues had written a OCaml compiler which omits javascript called Bucklescript. Seeing the typing problems that plague web development it now seems very natural that OCaml would prevent you from making some really silly mistakes. Before starting the project I was slightly familiar with LLVM and clang but this has certainly given me a new appreciation for the IR. It was very interesting being able to code in C++, generate llvm ir and linking it with our code. The after seeing the power of infinite registers, I hope to never have to see assembly code ever again.

7.2 Chip

Writing a compiler in a completely unknown language is a daunting task. OCaml definitely has a steep learning curve. The biggest takeaway for me was the understanding this project gave me about the low level workings of modern day compilers, from memory allocation to stack function calls etc. This helped me in uncovering some bugs in one of my other projects, which I never would have if I didn't know what was going on under the C compiler. My advice for future students is to peer code at least once a week. Peer coding keeps the errors down and helps in keeping everyone on the same page.

7.3 David

What I learnt the most about from this project is probably Ocaml. I have never programmed in a functional programming language before, and this was a good introduction, especially because of the already written codebase. Learning about LLVM was also very interesting. Most of all I learnt the ins and outs of compiler writing. And for me that was the most interesting part. That a compiler can be so cleanly decomposed into a few files of Ocaml was something I was never exposed to before. Also interesting was the process of making design decisions. For example, to implement the dim() function, we had to override the semantic checking because dim takes in an array of any size, but the way the semantic checker was written required a fill matrix type specification with the dimensions.

My advice to future teams would be to take advantage of existing codebases. I learnt a lot simply by reading previously written code for MicroC, and I think I could have learnt more if I read code from previous years' projects' files

7.4 Paul

Although I've written programs many times before, this was my first time doing a large group programming project. As a team manager I learned that planning such a project has its own challenges. Scheduling a group of five people with different schedules and commitments and keeping track of tasks was more difficult that I expected. It was also important for me to make sure I listened to everyone's input equally and did not leave any team member's opinions out. Besides the project management aspect, I learned a lot about the complete compiler pipeline, mostly on the front end, but also about IR and optimization. The favorite thing I learned about was LLVM which is a brilliant solution to the multiple languages and target architectures. Providing a middle layer between the two allows language developers to target a single virtual assembly language which can be targeted to any architecture, provided that conversion is written for that specific architecture. Much like hardware virtualization we have a middle abstraction layer that decouples hardware from software. Additionally, I now have a much better understanding of how compilers work which demystified something that seemed very complicated and seemingly beyond grasp. Particularly, I learned about how the semantic checking produces warning and errors in the compiler, one of the most useful features when programming.

My advice to future team members is to keep the scope of the language narrow and to work on tasks as soon as possible since the latter half of the semester has a heavier workload. Additionally, the language reference manual and final report should be updated throughout the entire timeline of the project.

7.5 Sharon

Of course, while working on the project, I learned how to create my own programming language, how assembly code is written, and how there are no limits to how a programming language can be designed. Before the project, I had never heard of LLVM and was very intimidated by the project because everyone else seemed to already know what an LLVM was and what assembly code was, and others on my team were using terms I did not recognize. However, actually doing the project has made me excited about extending the project or creating my own unique language. I feel that all projects I have started out feeling too incompetent to work on end up being fun, fulfilling, and rewarding when I do end up working on them and putting in all my effort on them. But I always seem to forget that and still feel the impostor syndrome every time.

All in all, I have learned a lot about working on a software engineering team project. I learned how to use branches on github, how to use a virtual machine, how to communicate on Slack, and how to divide responsibilities among a group. Most importantly, I learned that just like writing essays, programming a compiler is much easier when the work is split up into many days. Every day, you get to see what you have written or attempted with fresh eyes, from a different perspective. Also, I must say that partner programming is much more effective than individual programming, because more than one head is better than one. That would be my advice for future teams. To set specific goals, and assign each goal to a pair of members on a team. Also, for others in the team to understand and be able to extend the code that you write, it is much more efficient to have every code block be commented and for commit messages on github to be descriptive.

8. Appendix

parser.mly

```
/* Ocamlyacc parser for MicroC */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL VOID
%token RBRACK LBRACK ELIF BREAK FLOAT STRING BYTE
%token SHAPE DIMS FUNC
%token <int> LITERAL
%token <float> FLITERAL
%token <string> ID SLITERAL
%token EOF

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

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

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

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

```

```

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

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

typ:
    INT { Int }
  | BOOL { Bool }
  | VOID { Void }
  | FLOAT { Float }
  | STRING { String }
  | BYTE   { Byte }
  | typ matrix_params  %prec NOLBRACK { Matrix($1, List.rev $2) }

matrix_params:
    matrix_decl %prec NOLBRACK {[$1]}
  | matrix_params matrix_decl {$2 :: $1}

matrix_decl:
  LBRACK LITERAL RBRACK {$2}

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

vdecl:
   typ ID SEMI { ($1, $2 ) }

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

stmt:
    expr SEMI { Expr $1 }
  | RETURN SEMI { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | IF LPAREN expr RPAREN stmt elif_list %prec NOELSE { Elif(($3 :: (List.rev(fst $6))), (List.rev((Block([]) :: (List.rev ($5 :: (List.rev (snd $6)))))))) }
  | IF LPAREN expr RPAREN stmt elif_list ELSE stmt { Elif(($3 :: (List.rev(fst $6))), (List.rev(($8 :: (List.rev ($5 :: (List.rev (snd $6)))))))) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }

elif_list:
    elif {[fst $1],[snd $1]}
  | elif_list elif {(fst $2 :: fst $1 ), (snd $2 :: snd $1 )}

elif:
   ELIF LPAREN expr RPAREN stmt {$3,$5}

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

expr:
    LITERAL          { Literal($1) }
  | FLITERAL         { FLiteral($1) }
  | SLITERAL             { SLiteral($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID               { Id($1) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | ID matrix_accs { MatrixAccess($1, List.rev $2) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | ID matrix_accs ASSIGN expr { MatrixAssign($1, List.rev $2, $4) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }

matrix_accs:
    matrix_acc %prec NOLBRACK {[$1]}
  | matrix_accs matrix_acc {$2 :: $1}

matrix_acc:
  LBRACK expr RBRACK {$2}

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

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

scanner.mll

```
(* Ocamllex scanner for MicroC *)

{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ']'      { RBRACK } (*numnum*)
| '['      { LBRACK } (*numnum*)
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "elif"   { ELIF } (*numnum*)
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "break"  { BREAK } (*numnum*)
| "int"    { INT }
| "bool"   { BOOL }
| "void"   { VOID }
| "byte"   { BYTE } (*numnum*)
| "float"  { FLOAT } (*numnum*)
| "string" { STRING } (*numnum*)
| "true"   { TRUE }
| "false"  { FALSE }
| "shape"  { SHAPE } (*numnum*)
| "dims"   { DIMS } (*numnum*)
| "func"   { FUNC } (*numnum*)
| ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
| ['0'-'9']*'.'['0'-'9']+ as lxm { FLITERAL(float_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| '"'(([^'"'])*  as lxm)'"' { SLITERAL(lxm)}
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

semant.ml

```
(* Semantic checking for the MicroC compiler *)
open Ast

module StringMap = Map.Make(String)


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

   Check each global variable, then check each function *)
```

```
let check (globals, functions) =
  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper =
      function
      | n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
    in helper (List.sort compare list) in
  (* Raise an exception if a given binding is to a void type *)
  let check_not_void exceptf =
    function | (Void, n) -> raise (Failure (exceptf n)) | _ -> () in
  (* Raise an exception of the given rvalue type cannot be assigned to
     the given lvalue type *)
  let is_int_type a = (match a with
    | Int|Byte|Float -> true
    | Matrix (t,_) -> (match t with
        Int|Byte|Float -> true
        | _ -> false )
    | _ -> false
  ) in
  let check_assign lvaluet rvaluet err =
    if lvaluet == rvaluet then lvaluet
    else if (is_int_type lvaluet) &&  (is_int_type rvaluet) then lvaluet
    else raise err
  in

    (**** Checking Global Variables ****)
    (**** Checking Functions ****)
    (List.iter (check_not_void (fun n -> "illegal void global " ^ n)) globals;
     report_duplicate (fun n -> "duplicate global " ^ n)
       (List.map snd globals);
     if List.mem "print" (List.map (fun fd -> fd.fname) functions)
     then raise (Failure "function print may not be defined")
     else ();
     report_duplicate (fun n -> "duplicate function " ^ n)
       (List.map (fun fd -> fd.fname) functions);
     (* Function declaration for a named function *)
     let built_in_decls =
        StringMap.add "dim"
         {
           typ = Int;
           fname = "dim";
           (* The arguments to Matrix
           don't matter, they are overridden in the checker below, but we need
           them here for this to compile *)
           formals = [ (Matrix(Int, [1]), "x") ];
           locals = [];
           body = [];
         }

       (StringMap.add "print"
         {
           typ = Void;
           fname = "print";
           formals = [ (Int, "x") ];
           locals = [];
           body = [];
         }
         (StringMap.add "open"
          {
            typ = Int;
            fname = "open";
            formals = [ (String, "x"); (Int,"y") ];
            locals = [];
            body = [];
          }
          (StringMap.add "read"
          {
            typ = Int;
            fname = "read";
            formals = [(String,"w"); ((Matrix( Byte , [])), "x") ];
            locals = [];
            body = [];
          }
          (StringMap.add "write"
          {
            typ = Int;
            fname = "write";
            formals = [(String,"w"); ((Matrix( Byte , [])), "x") ];
            locals = [];
            body = [];
          }
          (StringMap.add "printbyte"
          {
            typ = Void;
            fname = "printbyte";
            formals = [ (Byte, "x") ];
            locals = [];
            body = [];
          }
          (StringMap.add "printb"
             {
               typ = Void;
               fname = "printb";
               formals = [ (Bool, "x") ];
               locals = [];
               body = [];
             }
             (StringMap.add "printstrn"
             {
               typ = Void;
               fname = "printstrn";
               formals = [ (String, "x") ];
               locals = [];
               body = [];
             }
             (StringMap.add "printfl"
                {
                  typ = Void;
                  fname = "printfl";
                  formals = [ (Float, "x") ];
                  locals = [];
                  body = [];
                }
                (StringMap.singleton "printstr"
                   {
                     typ = Void;
                     fname = "printstr";
                     formals = [ (String, "x") ];
```

```
                    locals = [];
                    body = [];
                  }
        ))))))))
     in
     let built_in_decls =
        List.fold_left (fun m f ->
            StringMap.add f
            {
                typ = Void;
                fname = f;
                formals = [(Matrix(Int, [1]), "x"); (Matrix(Int, [1]), "y"); (Matrix(Int, [1]), "z") ];
                locals = [];
                body = [];
            }
            m
        ) built_in_decls ["el_add"; "el_sub"; "el_mul"; "el_div"]
    in
    (*
     let built_in_decls =
        List.fold_left (fun m f ->
            StringMap.add f
            {
                typ = Void;
                fname = f;
                formals = [(Matrix(Int, [1]), "x"); (Matrix(Int, [1]), "y"); (Matrix(Bool, [true]), "z") ];
                locals = [];
                body = [];
            }
            m
        ) built_in_decls ["el_and"; "el_or"; "el_eq"; "el_neq"; "el_less"; "el_leq"; "el_greater"; "el_geq"]
    in
    *)
     let built_in_decls =
        List.fold_left (fun m f ->
            StringMap.add f
            {
                typ = Void;
                fname = f;
                formals = [(Matrix(Int, [1]), "x"); (Matrix(Int, [1]), "y"); (Matrix(Int, [1]), "z") ];
                locals = [];
                body = [];
            }
            m
        ) built_in_decls ["bc_add"; "bc_sub"; "bc_mul"; "bc_div"]
    in
     let function_decls =
       List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
         built_in_decls functions in
     let function_decl s =
       try StringMap.find s function_decls
       with | Not_found -> raise (Failure ("unrecognized function " ^ s)) in
     let _ = function_decl "main" in (* Ensure "main" is defined *)
     let check_function func =
       (List.iter
          (check_not_void
             (fun n -> "illegal void formal " ^ (n ^ (" in " ^ func.fname))))
          func.formals;
        report_duplicate
          (fun n -> "duplicate formal " ^ (n ^ (" in " ^ func.fname)))
          (List.map snd func.formals);
        List.iter
          (check_not_void
             (fun n -> "illegal void local " ^ (n ^ (" in " ^ func.fname))))
          func.locals;
        report_duplicate
          (fun n -> "duplicate local " ^ (n ^ (" in " ^ func.fname)))
          (List.map snd func.locals);
        (* Type of each variable (global, formal, or local *)
        let symbols =
          List.fold_left (fun m (t, n) -> StringMap.add n t m) StringMap.
            empty (globals @ (func.formals @ func.locals)) in
        let type_of_identifier s =
          try StringMap.find s symbols
          with | Not_found -> raise (Failure ("undeclared identifier " ^ s)) in
        let type_of_matrix_identifier s =
          try let sym = StringMap.find s symbols in
            match sym with
               Matrix (t,_) -> t
               | _  -> raise (Failure ("identifier isn't a matrix " ^ s))
          with | Not_found -> raise (Failure ("undeclared identifier " ^ s)) in
        (* Return the type of an expression or throw an exception *)
        let rec expr =
          function
          | Literal _ -> Int
          | FLiteral _ -> Float
          | SLiteral _ -> String
          | BoolLit _ -> Bool
          | Id s -> type_of_identifier s
          | MatrixAccess (s, _) -> type_of_matrix_identifier s
          | (MatrixAssign (s,_,e) as ex) ->
              let lt = type_of_identifier s
              and rt = expr e
              in
                check_assign lt rt
                  (Failure
                     ("illegal assignment " ^
                        ((string_of_typ lt) ^
                           (" = " ^
                              ((string_of_typ rt) ^
                                 (" in " ^ (string_of_expr ex)))))))
         | (Binop (e1, op, e2) as e) ->
              let t1 = expr e1
              and t2 = expr e2
              in
                (match op with
                 | Add | Sub | Mult | Div when (t1 = Int) && (t2 = Int) -> Int
                 | Add | Sub | Mult | Div when (t1 = Float) && (t2 = Float) -> Float
                 | Add | Sub | Mult | Div when (t1 = Byte) && (t2 = Byte) -> Byte
                 | Add | Sub | Mult | Div when (t1 = Byte) && (t2 = Int) -> Byte
                 | Add | Sub | Mult | Div when (t1 = Byte) && (t2 = Float) -> Byte
                 | Add | Sub | Mult | Div when (t1 = Int) && (t2 = Byte) -> Int
                 | Add | Sub | Mult | Div when (t1 = Int) && (t2 = Float) -> Int
                 | Add | Sub | Mult | Div when (t1 = Float) && (t2 = Byte) -> Float
                 | Add | Sub | Mult | Div when (t1 = Float) && (t2 = Int) -> Float
                 | Equal | Neq when t1 = t2 -> Bool
```

```
                 | Equal | Neq when (t1 = Int) && (t2 = Byte) -> Bool
                 | Less | Leq | Greater | Geq when (t1 = Int) && (t2 = Int) -> Bool
                 | Less | Leq | Greater | Geq when (t1 = Int) && (t2 = Byte) -> Bool
                 | Less | Leq | Greater | Geq when (t1 = Byte) && (t2 = Int) -> Bool
                 | Less | Leq | Greater | Geq when (is_int_type t1) && (is_int_type t2) -> Bool
                 | And | Or when (t1 = Bool) && (t2 = Bool) -> Bool
                 | _ ->
                     raise
                       (Failure
                          ("illegal binary operator " ^
                             ((string_of_typ t1) ^
                                (" " ^
                                   ((string_of_op op) ^
                                      (" " ^
                                         ((string_of_typ t2) ^
                                            (" in " ^ (string_of_expr e))))))))))
          | (Unop (op, e) as ex) ->
              let t = expr e
              in
                (match op with
                 | Neg when t = Int -> Int
                 | Not when t = Bool -> Bool
                 | _ ->
                     raise
                       (Failure
                          ("illegal unary operator " ^
                             ((string_of_uop op) ^
                                ((string_of_typ t) ^
                                   (" in " ^ (string_of_expr ex)))))))
          | Noexpr -> Void
          | (Assign (var, e) as ex) ->
              let lt = type_of_identifier var
              and rt = expr e
              in
                check_assign lt rt
                  (Failure
                     ("illegal assignment " ^
                        ((string_of_typ lt) ^
                           (" = " ^
                              ((string_of_typ rt) ^
                                 (" in " ^ (string_of_expr ex)))))))
          | (Call (fname, actuals) as call) ->
              let fd = function_decl fname
              in
                (if ( != ) (List.length actuals) (List.length fd.formals)
                 then
                   raise
                     (Failure
                        ("expecting " ^
                           ((string_of_int (List.length fd.formals)) ^
                              (" arguments in " ^ (string_of_expr call)))))
                 else
                     if (fname = "dim") then
                         let e = List.hd actuals in
                         match (e) with
                         | Id(m) -> (match (type_of_identifier m) with
                             | Matrix(_,_) -> ()
                             | _ -> raise (Failure ("illegal argument to dim() found expected Matrix in " ^ (string_of_expr e))))
                         | _ -> raise (Failure ("illegal argument to dim() found  expected Matrix in " ^ (string_of_expr e)))
             else if (fname = "el_add" || fname = "el_sub" || fname = "el_mul" || fname = "el_div") then
                let e = List.hd actuals in
                (match(e) with
                    | Id(m) -> (match (type_of_identifier m) with
                        | Matrix(_, _) ->
                            let comp_matrix e1 e2 =
                            (match(e1, e2) with
                                | Id(m1), Id(m2) -> (match (type_of_identifier m1, type_of_identifier m2) with
                                    | Matrix(t1, l1), Matrix(t2, l2) ->
                                        let rec compareVs v1 v2 = match v1, v2 with
                                            | [], [] -> true
                                            | [], _
                                            | _, [] -> false
                                            | x::xs, y::ys -> x=y && compareVs xs ys
                                        in
                                        if (t1 != t2) then
                                            raise(Failure ("incompatibles types of matrices to " ^ fname))
                                        else if not (compareVs l1 l2) then
                                            raise(Failure ("incompatibles dimensions of matrices to " ^ fname))
                                        else
                                            e2
                                    | _, _ -> raise (Failure ("illegal argument to " ^ fname ^ " found expected Matrix in " ^ (string_of_expr e))))
                                | _, _ -> raise (Failure ("illegal argument to " ^ fname ^ " found  expected Matrix in " ^ (string_of_expr e))))
                                 (* checking to see if two matrices have same type and shape *)
                            in
                            ignore(List.fold_left comp_matrix e (List.tl actuals)); ()
                        | _ -> raise (Failure ("illegal argument to " ^ fname ^ " found  expected Matrix in " ^ (string_of_expr e))))
                    | _ -> raise(Failure ("illegal argument to " ^ fname ^ " found expected Matrix in "^ (string_of_expr e)))
                )
             else if (fname = "bc_add" || fname = "bc_sub" || fname = "bc_mul" || fname = "bc_div") then
                let e = List.hd actuals in
                (match(e) with
                    | Id(m) -> (match (type_of_identifier m) with
                        | Matrix(_, [1]) ->
                            let comp_matrix e1 e2 =
                            (match(e1, e2) with
                                | Id(m1), Id(m2) -> (match (type_of_identifier m1, type_of_identifier m2) with
                                    | Matrix(t1, l1), Matrix(t2, l2) ->
                                        let rec compareVs v1 v2 = match v1, v2 with
                                            | [], [] -> true
                                            | [], _
                                            | _, [] -> false
                                            | x::xs, y::ys -> x=y && compareVs xs ys
                                        in
                                        if (t1 != t2) then
                                            raise(Failure ("incompatibles types of matrices to " ^ fname))
```

```
                  else if not (compareVs l1 l2) then
                    raise(Failure ("incompatibles dimensions of matrices to " ^ fname))
                  else
                    e2
                | _, _ -> raise (Failure ("illegal argument to " ^ fname ^ " found expected Matrix in " ^ (string_of_expr e))))
              | _, _ -> raise (Failure ("illegal argument to " ^ fname ^ " found expected Matrix in " ^ (string_of_expr e))))
              (* checking to see if two matrices have same type and shape *)
            in
            ignore(List.fold_left comp_matrix (List.hd (List.tl actuals)) (List.tl actuals)); ()
          | _ -> raise (Failure ("illegal argument to " ^ fname ^ " found expected Matrix in " ^ (string_of_expr e))))
        | _ -> raise(Failure ("illegal argument to " ^ fname ^ " found expected Matrix in "^ (string_of_expr e)))
      )
    else

       List.iter2
        (fun (ft, _) e ->
          let et = expr e
          in
           ignore
            (check_assign ft et
             (Failure
              ("illegal actual argument found " ^
                ((string_of_typ et) ^
                 (" expected " ^
                   ((string_of_typ ft) ^
                    (" in " ^ (string_of_expr e))))))))
        fd.formals actuals;
      fd.typ) in
  let check_bool_expr e =
   if ( != ) (expr e) Bool
   then
    raise
     (Failure
      ("expected Boolean expression in " ^ (string_of_expr e)))
   else () in
  (* Verify a statement or throw an exception *)
  let rec stmt =
   function
   | Block sl ->
     let rec check_block =
      (function
       | [ (Return _ as s) ] -> stmt s
       | Return _ :: _ ->
         raise (Failure "nothing may follow a return")
       | Block sl :: ss -> check_block (sl @ ss)
       | s :: ss -> (stmt s; check_block ss)
       | [] -> ())
     in check_block sl
   | Expr e -> ignore (expr e)
   | Return e ->
     let t = expr e
     in
      if t = func.typ
```

```
                then ()
                else
                  raise
                    (Failure
                       ("return gives " ^
                          ((string_of_typ t) ^
                             (" expected " ^
                                ((string_of_typ func.typ) ^
                                   (" in " ^ (string_of_expr e)))))))
          | If (p, b1, b2) -> (check_bool_expr p; stmt b1; stmt b2)
          | Elif (exprs, stmts) ->
              (List.iter check_bool_expr exprs;
               List.iter stmt stmts)
          | For (e1, e2, e3, st) ->
              (ignore (expr e1);
               check_bool_expr e2;
               ignore (expr e3);
               stmt st)
          | While (p, s) -> (check_bool_expr p; stmt s)
        in stmt (Block func.body))
     in List.iter check_function functions)


```

ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or

type uop = Neg | Not

type typ = Int | Bool | Void
        | Float | String | Byte
        | Matrix of typ * int list

type bind = typ * string

type expr =
    Literal of int
  | FLiteral of float
  | BoolLit of bool
  | SLiteral of string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | MatrixAccess of string * expr list
  | MatrixAssign of string * expr list * expr
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | Elif of expr list * stmt list * stmt list
  | For of expr * expr * expr * stmt
  | While of expr * stmt

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



type program = bind list * func_decl list

(* Pretty-printing functions *)

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

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

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | FLiteral(l) -> string_of_float l
  | SLiteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | MatrixAccess (t,dims) -> t ^ (List.fold_left (fun acc el -> "[" ^ (string_of_expr el) ^ "]" ^ acc) "" dims)
  | MatrixAssign (t,dims,e) -> let r = string_of_expr e in
      t ^ (List.fold_left (fun acc el -> "[" ^ (string_of_expr el) ^ "]" ^ acc )"" dims) ^ " = " ^ r
  | Binop(e1, o, e2) ->
        let l = string_of_expr e1 and r = string_of_expr e2 in
            (l ^ " " ^ string_of_op o ^ " " ^ r)
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""

```

```
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
  | Elif(exprs, stmts) ->  "if (" ^ string_of_expr (List.hd exprs) ^ ")\n" ^
      string_of_stmt (List.hd stmts)
      ^ String.concat "" (List.map2 (fun e s -> "elif (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s) (List.tl exprs) (List.tl (List.rev (List.tl (List.rev stmts)))))
      ^ "else\n" ^ string_of_stmt (List.hd (List.rev stmts))
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Void -> "void"
  | Float -> "float"
  | String -> "string"
  | Byte -> "byte"
  | Matrix(t, l) -> (string_of_typ t) ^ (List.fold_left (fun acc el -> acc ^ "[" ^ (string_of_int el) ^ "]" ) "" l)

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

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

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

codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)
module L = Llvm

module A = Ast

module StringMap = Map.Make(String)


let translate (globals, functions) =
  let context = L.global_context () in

  let the_module = L.create_module context "NumNum"
  and i32_t = L.i32_type context
  and i8_t = L.i8_type context
  and i1_t = L.i1_type context
  and void_t = L.void_type context
  and float_t = L.double_type context
  and string_t = L.pointer_type (L.i8_type context)
  and array_t t dims = L.array_type t (List.fold_left (fun acc el -> acc*el) 1 dims) in
  let rec ltype_of_typ =
    function
    | A.Int -> i32_t
    | A.Bool -> i1_t
    | A.Void -> void_t
    | A.String -> string_t
    | A.Float -> float_t
    | A.Byte -> i8_t
    | A.Matrix (t, dims) -> array_t (ltype_of_typ t) dims in
  (* Declare each global variable; remember its value in a map *)
  let global_vars =
    let errno = (L.define_global "errno" (L.const_int i32_t 0) the_module,A.Int) in
    let () = L.set_linkage L.Linkage.Available_externally (fst errno) in
    let global_var m (t, n) =
      let init = L.const_int (ltype_of_typ t) 0
      in StringMap.add n ((L.define_global n init the_module),t) m
    in List.fold_left global_var (StringMap.singleton "errno" errno) globals in

  (* Declare linux functions numnum will call *)
  let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func = L.declare_function "printf" printf_t the_module in
  let open_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t;i32_t |] in
  let open_func = L.declare_function "open" open_t the_module in
  let read_t = L.var_arg_function_type i32_t [| i32_t; L.pointer_type i32_t; i32_t |] in
  let read_func = L.declare_function "read" read_t the_module in
  let readbyte_t = L.var_arg_function_type i32_t [| i32_t; L.pointer_type i8_t; i32_t |] in
  let readbyte_func = L.declare_function "read" readbyte_t the_module in
  let readfl_t = L.var_arg_function_type i32_t [| i32_t; L.pointer_type float_t; i32_t |] in
  let readfl_func = L.declare_function "read" readfl_t the_module in
  let creat_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t;i32_t |] in
  let creat_func = L.declare_function "creat" creat_t the_module in
  let write_t = L.var_arg_function_type i32_t [| i32_t; L.pointer_type i8_t; i32_t |] in
  let write_func = L.declare_function "write" write_t the_module in
  let close_t = L.var_arg_function_type i32_t [| i32_t |] in
  let close_func = L.declare_function "close" close_t the_module in
  (* Define each function (arguments and return type) so we can call it *)
```

```
let function_decls =
  let function_decl m fdecl =
    let name = fdecl.A.fname
    and formal_types =
      Array.of_list
        (List.map (fun (t, _) -> ltype_of_typ t) fdecl.A.formals) in
    let ftype = L.function_type (ltype_of_typ fdecl.A.typ) formal_types
    in
      StringMap.add name ((L.define_function name ftype the_module), fdecl)
        m
  in List.fold_left function_decl StringMap.empty functions in
(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.A.fname function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in
  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
  let byte_format_str = L.build_global_stringptr "%x\n" "fmt" builder in
  let float_format_str = L.build_global_stringptr "%f\n" "fmt" builder in
  let string_format_str = L.build_global_stringptr "%s\n" "fmt" builder in
  let stringn_format_str = L.build_global_stringptr "%s" "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,t) m)) in
    let add_local m (t, n) =
      let local_var = L.build_alloca (ltype_of_typ t) n builder
      in StringMap.add n (local_var,t) m in
    let formals =
      List.fold_left2 add_formal StringMap.empty fdecl.A.formals
        (Array.to_list (L.params the_function))
    in List.fold_left add_local formals fdecl.A.locals in
  (* Return the value for a variable or formal argument *)
  let lookup n =
    try match (StringMap.find n local_vars) with (lt,_) -> lt
    with | Not_found ->  match (StringMap.find n global_vars) with (lt,_) -> lt in
  (* Look up the dimensions for a matrix *)
  let lookup_dims n =
    let get_dims t = match t with
        A.Matrix (_,dims) -> dims
      | _ -> [] in
    try match (StringMap.find n local_vars) with (_,t) -> get_dims t
    with | Not_found ->  match (StringMap.find n global_vars) with (_,t) -> get_dims t in
  let lookup_type n =
    let get_type t = match t with
        A.Matrix (typ,_) -> typ
      | _ -> t in
    try match (StringMap.find n local_vars) with (_,typ) -> get_type typ
    with | Not_found ->  match (StringMap.find n global_vars) with (_,typ) -> get_type typ in
  let integer_conv_op lh rh builder =
      let rht =  (L.type_of rh) in
      let lht=  (L.type_of lh) in
      ( match lht with
          | _ when lht == i8_t ->  (
            match  rht with
```

```
                | _ when rht == i32_t -> (L.build_intcast rh i8_t "conv" builder)
                | _ when rht == float_t ->  (L.build_uitofp rh i8_t "conv" builder)
                | _ -> rh )
            | _ when lht == i32_t -> (
              match  rht with
                | _ when rht == i8_t -> (L.build_intcast rh i32_t "conv" builder)
                | _ when rht == float_t ->  (L.build_fptosi rh i32_t "conv" builder)
                | _ -> rh )
            | _ when lht == float_t ->  (
              match  rht with
                | _ when rht == float_t -> rh
                | _ ->  ( L.build_sitofp rh float_t "conv" builder) )
            | _ -> rh  ) in
    let integer_conversion lh rh builder =
        let rht = (L.type_of rh) in
          (match lh with
            | A.Byte -> (match rht with
                | _ when rht == i8_t -> rh
                | _ when rht == float_t  -> (L.build_fptosi rh i8_t "conv" builder)
                | _ -> ( L.build_intcast rh i8_t "conv" builder) )
            | A.Int -> (match rht with
                | _ when rht == i32_t -> rh
                | _ when rht == float_t -> (L.build_fptosi rh i32_t "conv" builder)
                | _  -> ( L.build_intcast rh i32_t "conv" builder) )
            | A.Float -> (match rht with
                | _ when rht == float_t -> rh
                | _ when rht == i8_t -> ( L.build_uitofp rh float_t "conv" builder)
                | _ -> ( L.build_sitofp rh float_t "conv" builder) )
            | _ -> rh) in
    (* Construct code for an expression; return its value *)
    let rec expr builder =
      function
      | A.Literal i -> L.const_int i32_t i
      | A.FLiteral i -> L.const_float float_t i
      | A.SLiteral l -> L.build_global_stringptr l "tmp" builder
      | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
      | A.Noexpr -> L.const_int i32_t 0
      | A.Id s -> L.build_load (lookup s) s builder
      | A.MatrixAccess ( s, params) ->
          let dims = lookup_dims s in
          let acc_params = List.map (fun el -> (expr builder el)) params in
          let get_pos = List.fold_right2
                          (fun p d acc -> (L.build_add p (L.build_mul (L.const_int i32_t d) acc "tmp" builder) "tmp" builder))
                          acc_params
                          dims
                          (L.const_int i32_t 0) in
          L.build_load (L.build_gep (lookup s) [|L.const_int i32_t 0;get_pos|] "tmp" builder) "tmp" builder
      | A.Binop (e1, op, e2) ->
          let e1' = expr builder e1 in
          let e2' = expr builder e2 in (*(print_int (L.integer_bitwidth (L.type_of e1')));*)
          let e2f = (integer_conv_op e1' e2' builder) in
          let etype = L.classify_type (L.type_of (expr builder e1))
          in
            (match etype with
             | L.TypeKind.Double ->
                 (match op with
                  | A.Add -> L.build_fadd
```

```
                      | A.Sub -> L.build_fsub
                      | A.Mult -> L.build_fmul
                      | A.Div -> L.build_fdiv
                      | A.And -> L.build_and
                      | A.Or -> L.build_or
                      | 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.Greater -> L.build_fcmp L.Fcmp.Ogt
                      | A.Geq -> L.build_fcmp L.Fcmp.Oge) e1' e2f "tmp" builder
                 | _ ->
                     (match op with
                      | A.Add -> L.build_add
                      | A.Sub -> L.build_sub
                      | A.Mult -> L.build_mul
                      | A.Div -> L.build_sdiv
                      | 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
                      | A.Less -> L.build_icmp L.Icmp.Slt
                      | A.Leq -> L.build_icmp L.Icmp.Sle
                      | A.Greater -> L.build_icmp L.Icmp.Sgt
                      | A.Geq -> L.build_icmp L.Icmp.Sge) e1' e2f "tmp" builder)
          | A.Unop (op, e) ->
              let e' = expr builder e
              in
                (match op with | A.Neg -> L.build_neg | A.Not -> L.build_not) e'
                  "tmp" builder
          | A.Assign (s, e) ->
              let e' = expr builder e in
              let s' = (lookup s) in
              let ef = (integer_conversion (lookup_type s) e' builder) in
                (ignore (L.build_store ef s' builder)); ef
          | A.MatrixAssign (s,dims_assign,e) ->
              let e' = expr builder e in
              let s' = (lookup s) in
              let ef = (integer_conversion (lookup_type s) e' builder) in
              let dims = lookup_dims s in
              let acc_params = List.map (fun el -> (expr builder el)) dims_assign in
              let get_pos = List.fold_right2
                              (fun p d acc -> (L.build_add p (L.build_mul (L.const_int i32_t d) acc "tmp" builder) "tmp" builder))
                              acc_params
                              dims
                              (L.const_int i32_t 0) in
              L.build_store  ef (L.build_gep s' [|L.const_int i32_t 0;get_pos|] "tmp" builder) builder
          | A.Call ("print", ([ e ])) | A.Call ("printb", ([ e ])) ->
              L.build_call printf_func [| int_format_str; expr builder e |]
                "printf" builder
          | A.Call ("printfl", ([ e ])) ->
              L.build_call printf_func [| float_format_str; expr builder e |]
                "printf" builder
          | A.Call ("printstr", ([ e ])) ->
              L.build_call printf_func [| string_format_str; expr builder e |]
                "printf" builder
          | A.Call ("printbyte", ([ e ])) ->
              L.build_call printf_func [| byte_format_str; expr builder e |]
                "printf" builder
          | A.Call ("printstrn", ([ e ])) ->
              L.build_call printf_func [| stringn_format_str; expr builder e |]
                "printf" builder
          | A.Call ("dim", ([ e ])) ->
                  ( match e with
                    | A.Id(t) ->
                      let d = L.build_alloca  i32_t "tmp" builder in
                      (ignore (L.build_store (L.const_int i32_t (List.length (lookup_dims t))) d  builder);
                      L.build_load d "tmp" builder)
                    | _ -> expr builder e)
          | A.Call (op, ([a; b; c])) ->
                ( match op with
                    | "el_add" | "el_sub" | "el_mul" | "el_div" ->
                        let el_op = op in
                        ( match a, b, c with
                            | A.Id(x), A.Id(y), A.Id(z) ->

                                (* Get a list of params lists *)
                                let dims = lookup_dims x in
                                let rec range i j = if i >= j then [] else A.Literal(i) :: (range (i+1) j) in
                                let dim2 = range 0 1 in
                                let dim1 = range 0 1 in
                                let tmp1 = List.concat (List.map (fun x -> List.map (fun y -> y::[x]) dim2) dim1) in
                                let tmp2 = List.fold_left (fun tmp dim -> (List.concat (List.map (fun x -> List.map (fun y -> y::x) (range 0 dim)) tmp))) tmp1 dims in
                                let all_pos = List.map List.rev (List.map List.rev (List.map List.tl (List.map List.tl (List.map List.rev tmp2)))) in

                                (* Do multiplication at each of the positions *)
                                let do_op = fun builder params ->
                                    let e1 = A.MatrixAccess(x, params) in
                                    let e2 = A.MatrixAccess(y, params) in
                                    let e1' = expr builder e1 in
                                    let e2' = expr builder e2 in
                                    let etype = L.classify_type (L.type_of e1') in
                                    let r = (match etype with
                                        | L.TypeKind.Double ->
                                            (match el_op with
                                                | "el_add" -> L.build_fadd
                                                | "el_sub" -> L.build_fsub
                                                | "el_mul" -> L.build_fmul
                                                | "el_div" -> L.build_fdiv
                                                (*
                                                | "el_and" -> L.build_and
                                                | "el_or" -> L.build_or
                                                | "el_eq" -> L.build_fcmp L.Fcmp.Oeq
                                                | "el_neq" -> L.build_fcmp L.Fcmp.One
                                                | "el_less" -> L.build_fcmp L.Fcmp.Olt
                                                | "el_leq" -> L.build_fcmp L.Fcmp.Ole
                                                | "el_greater" -> L.build_fcmp L.Fcmp.Ogt
                                                | "el_geq" -> L.build_fcmp L.Fcmp.Oge
                                                *)
                                                | _ -> raise (Failure ("Unable to do element-wise operation " ^ el_op ^ " on matrices"))
                                            )
                                        | _ ->
```

```
                                        (match el_op with
                                            | "el_add" -> L.build_add
                                            | "el_sub" -> L.build_sub
                                            | "el_mul" -> L.build_mul
                                            | "el_div" -> L.build_sdiv
                                            (*
                                            | "el_and" -> L.build_and
                                            | "el_or" -> L.build_or
                                            | "el_eq" -> L.build_icmp L.Icmp.Eq
                                            | "el_neq" -> L.build_icmp L.Icmp.Ne
                                            | "el_less" -> L.build_icmp L.Icmp.Slt
                                            | "el_leq" -> L.build_icmp L.Icmp.Sle
                                            | "el_greater" -> L.build_icmp L.Icmp.Sgt
                                            | "el_geq" -> L.build_icmp L.Icmp.Sge
                                            *)
                                            | _ -> raise (Failure ("Unable to do element-wise operation " ^ el_op ^ " on matrices"))
                                        )
                                    ) e1' e2' "tmp" builder
                                in
                                let z' = (lookup z) in
                                let ef = (integer_conversion (lookup_type z) r builder) in
                                let dims = lookup_dims z in
                                let acc_params = List.map (fun el -> (expr builder el)) params in
                                let get_pos = List.fold_right2
                                                  (fun p d acc -> (L.build_add p (L.build_mul (L.const_int i32_t d) acc "tmp" builder) "tmp" builder))
                                                  acc_params
                                                  dims
                                                  (L.const_int i32_t 0) in
                                ignore(L.build_store ef (L.build_gep z' [|L.const_int i32_t 0;get_pos|] "tmp" builder) builder); builder
                            in
                            ignore(List.fold_left do_op builder all_pos); L.const_int i32_t 0

                        | _, _, _ -> raise (Failure ("Unable to do element-wise operation " ^ el_op ^ " on matrices"))
                    )
                | "bc_add" | "bc_sub" | "bc_mul" | "bc_div" ->
                    let bc_op = op in
                    ( match a, b, c with
                        | A.Id(x), A.Id(y), A.Id(z) ->
                            (* Get a list of params lists *)
                            let dims = lookup_dims y in
                            let rec range i j = if i >= j then [] else A.Literal(i) :: (range (i+1) j) in
                            let dim2 = range 0 1 in
                            let dim1 = range 0 1 in
                            let tmp1 = List.concat (List.map (fun x -> List.map (fun y -> y::[x]) dim2) dim1) in
                            let tmp2 = List.fold_left (fun tmp dim -> (List.concat (List.map (fun x -> List.map (fun y -> y::x) (range 0 dim)) tmp))) tmp1 dims in
                            let all_pos = List.map List.rev (List.map List.rev (List.map List.tl (List.map List.tl (List.map List.rev tmp2)))) in

                            (* Do multiplication at each of the positions *)
                            let do_op = fun builder params ->
```

```
                                let e1 = A.MatrixAccess(x, [A.Literal(0)]) in
                                let e2 = A.MatrixAccess(y, params) in
                                let e1' = expr builder e1 in
                                let e2' = expr builder e2 in
                                let etype = L.classify_type (L.type_of e1') in
                                let r = (match etype with
                                    | L.TypeKind.Double ->
                                        (match bc_op with
                                            | "bc_add" -> L.build_fadd
                                            | "bc_sub" -> L.build_fsub
                                            | "bc_mul" -> L.build_fmul
                                            | "bc_div" -> L.build_fdiv
                                            | _ -> raise (Failure ("Unable to do broadcast operation " ^ bc_op ^ " on matrices"))
                                        )
                                    | _ ->
                                        (match bc_op with
                                            | "bc_add" -> L.build_add
                                            | "bc_sub" -> L.build_sub
                                            | "bc_mul" -> L.build_mul
                                            | "bc_div" -> L.build_sdiv
                                            | _ -> raise (Failure ("Unable to do broadcast operation " ^ bc_op ^ " on matrices"))
                                        )
                                    ) e1' e2' "tmp" builder
                                in
                                let z' = (lookup z) in
                                let ef = (integer_conversion (lookup_type z) r builder ) in
                                let dims = lookup_dims z in
                                let acc_params = List.map (fun el -> (expr builder el) ) params in
                                let get_pos = List.fold_right2
                                                  (fun p d acc -> (L.build_add p (L.build_mul (L.const_int i32_t d) acc "tmp" builder) "tmp" builder))
                                                  acc_params
                                                  dims
                                                  (L.const_int i32_t 0) in
                                ignore(L.build_store ef (L.build_gep z' [|L.const_int i32_t 0;get_pos|] "tmp" builder) builder); builder
                            in
                            ignore(List.fold_left do_op builder all_pos); L.const_int i32_t 0

                        | _ -> raise (Failure ("Unable to do broadcast operation " ^ bc_op ^ " on matrices"))
                    )
                 | _ -> raise (Failure ("Unable to do operation " ^ op ^ " on matrices"))
            )
      | A.Call ("open", ([ e ; e2 ])) ->
              (L.build_call open_func [| expr builder e;expr builder e2|] "open" builder)
      | A.Call ("read", ([ e ; e2 ])) ->
                let ev = expr builder e and
                 ev2 = A.string_of_expr e2 in
                let arrptr = (lookup ev2) in
                let arrtype = (lookup_type ev2) in
                let arrsize = (List.fold_left (fun acc el -> acc*el) 1 (lookup_dims ev2))  in
```

```
        let fd = (L.build_call open_func [| ev ; L.const_int i32_t 0|] "open" builder) in
        let ret = (match arrtype with
             A.Byte -> (L.build_call readbyte_func
                       [| fd ;
                        (L.build_gep arrptr [|L.const_int i32_t 0;L.const_int i32_t 0|] "tmp" builder);
                         L.const_int i32_t (arrsize)|] "read" builder)
             | A.Int -> (L.build_call read_func
                       [| fd ;
                        (L.build_gep arrptr [|L.const_int i32_t 0;L.const_int i32_t 0|] "tmp" builder);
                         L.const_int i32_t (arrsize*4)|] "read" builder)
             | A.Float -> (L.build_call readfl_func
                       [| fd ;
                        (L.build_gep arrptr [|L.const_int i32_t 0;L.const_int i32_t 0|] "tmp" builder);
                         L.const_int i32_t (arrsize*8)|] "read" builder)
             | _ -> raise (Failure ("Unable to read into matrix type " ^ (A.string_of_typ arrtype)))
        ) in
        (ignore (L.build_call close_func [| fd |] "close" builder));ret
   | A.Call ("write", ([e; e2])) ->
        let path = expr builder e and
        var_name =  A.string_of_expr e2 in
        let arrptr = (lookup var_name) in
        let arrsize = (List.fold_left (fun acc el -> acc*el) 1 (lookup_dims var_name)) in
        let fd = (L.build_call creat_func [| path ; L.const_int i32_t 438|] "creat" builder) in
        let ret = L.build_call write_func
                       [| fd ;
                        (L.build_gep arrptr [|L.const_int i32_t 0;L.const_int i32_t 0|] "tmp" builder);
                         L.const_int i32_t (arrsize)|] "write" builder
        in
        (ignore (L.build_call close_func [| fd |] "close" builder));ret
   | A.Call (f, act) ->
     let (fdef, fdecl) = StringMap.find f function_decls in
     let actuals = List.rev (List.map (expr builder) (List.rev act)) in
     let result =
       (match fdecl.A.typ with | A.Void -> "" | _ -> f ^ "_result")
     in L.build_call fdef (Array.of_list actuals) result builder in
  (* Invoke "f builder" if the current block doesn't already
    have a terminal (e.g., a branch). *)
  let add_terminal builder f =
   match L.block_terminator (L.insertion_block builder) with
   | Some _ -> ()
   | None -> ignore (f builder) in
  (* Build the code for the given statement; return the builder for
    the statement's successor *)
  let rec stmt builder =
   function
   | A.Block sl -> List.fold_left stmt builder sl
   | A.Expr e -> (ignore (expr builder e); builder)
   | A.Return e ->
```

```
        (ignore
         (match fdecl.A.typ with
          | A.Void -> L.build_ret_void builder
          | _ -> L.build_ret (expr builder e) builder);
        builder)
      | A.If (predicate, then_stmt, else_stmt) ->
        let bool_val = expr builder predicate in
        let merge_bb = L.append_block context "merge" the_function in
        let then_bb = L.append_block context "then" the_function
        in
         (add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
          (L.build_br merge_bb);
         let else_bb = L.append_block context "else" the_function
         in
          (add_terminal
            (stmt (L.builder_at_end context else_bb) else_stmt)
            (L.build_br merge_bb);
           ignore (L.build_cond_br bool_val then_bb else_bb builder);
           L.builder_at_end context merge_bb))
      | A.Elif (exprs, stmts) ->
         (match exprs with
           [] ->
             (match stmts with
               [] -> builder
               | h::_ ->
                 stmt builder (A.Block [ A.Block [(h)]])
             )
           | _ ->
             let bool_val = expr builder (List.hd exprs) in
             let merge_bb = L.append_block context "merge" the_function in
             let then_bb = L.append_block context "then" the_function
             in
             (add_terminal (stmt (L.builder_at_end context then_bb) (List.hd stmts))
               (L.build_br merge_bb);
               let else_bb = L.append_block context "else" the_function
               in
               (add_terminal
                 (stmt (L.builder_at_end context else_bb) (A.Elif (List.tl exprs, List.tl stmts)))
                 (L.build_br merge_bb);
               ignore (L.build_cond_br bool_val then_bb else_bb builder);
               L.builder_at_end context merge_bb))
         )
      | A.While (predicate, body) ->
        let pred_bb = L.append_block context "while" the_function
        in
         (ignore (L.build_br pred_bb builder);
         let body_bb = L.append_block context "while_body" the_function
         in
          (add_terminal (stmt (L.builder_at_end context body_bb) body)
            (L.build_br pred_bb);
           let pred_builder = L.builder_at_end context pred_bb in
           let bool_val = expr pred_builder predicate in
           let merge_bb = L.append_block context "merge" the_function
           in
            (ignore
             (L.build_cond_br bool_val body_bb merge_bb pred_builder);
            L.builder_at_end context merge_bb)))
      | A.For (e1, e2, e3, body) ->
```

```
          stmt builder
            (A.Block
               [ A.Expr e1; A.While (e2, (A.Block [ body; A.Expr e3 ])) ]) in
    (* Build the code for each statement in the function *)
    let builder = stmt builder (A.Block fdecl.A.body)
    in
      (* Add a return if the last block falls off the end *)
      add_terminal builder
        (match fdecl.A.typ with
         | A.Void -> L.build_ret_void
         | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in (List.iter build_function_body functions; the_module)
```

numnum.ml

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

type action = Ast | LLVM_IR | Compile

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

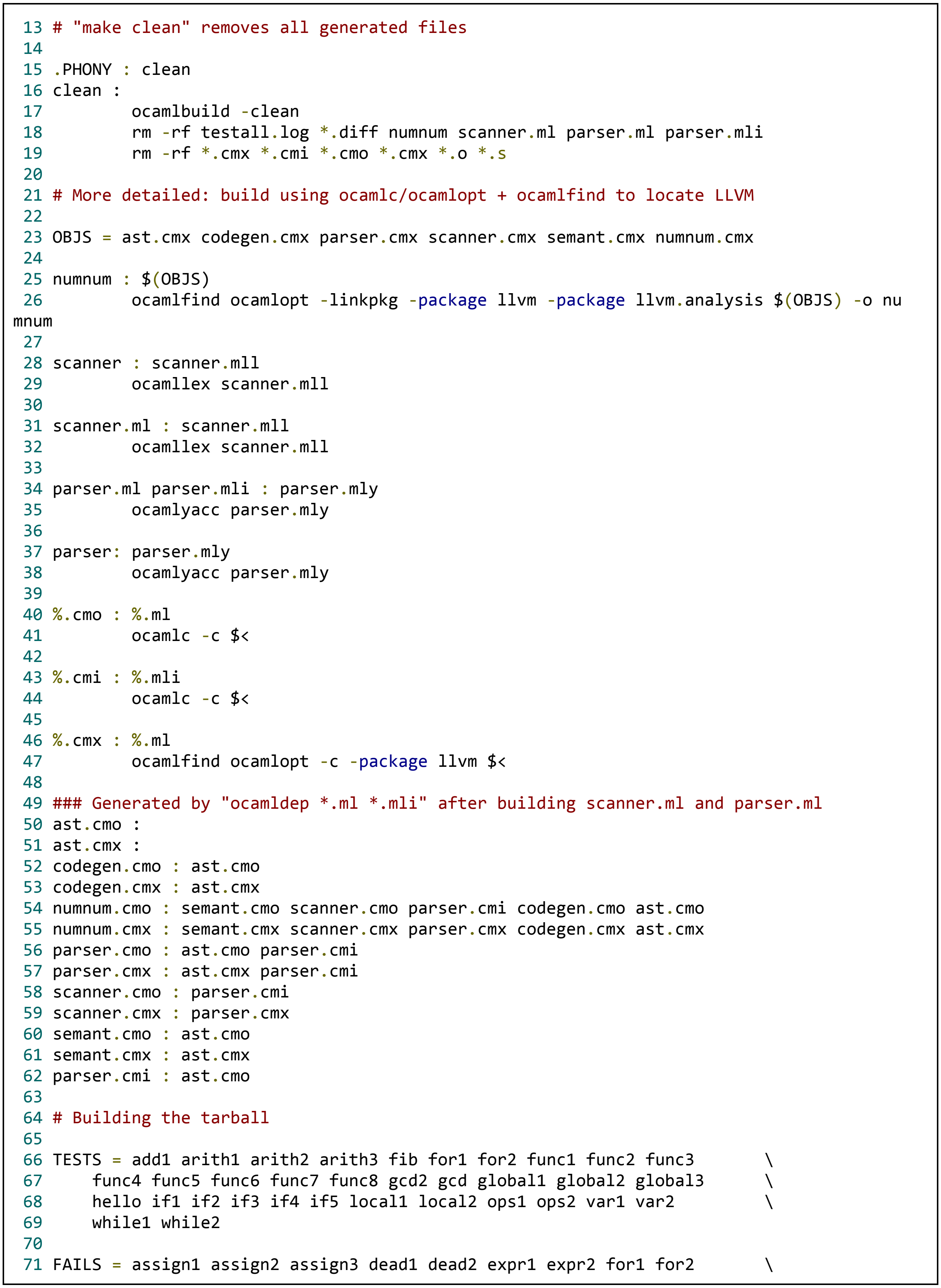
Makefile

```
# Make sure ocamlbuild can find opam-managed packages: first run
#
# eval `opam config env`

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

.PHONY : numnum.native

numnum.native :
        ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis,llvm.bitwriter,llvm.bitreader,llvm.linker,llvm.target -cflags -w,+a-4 \
                numnum.native
```



```
# "make clean" removes all generated files

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

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

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

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

scanner : scanner.mll
        ocamllex scanner.mll

scanner.ml : scanner.mll
        ocamllex scanner.mll

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

parser: parser.mly
        ocamlyacc parser.mly

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

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

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

### Generated by "ocamldep *.ml *.mli" after building scanner.ml and parser.ml
ast.cmo :
ast.cmx :
codegen.cmo : ast.cmo
codegen.cmx : ast.cmx
numnum.cmo : semant.cmo scanner.cmo parser.cmi codegen.cmo ast.cmo
numnum.cmx : semant.cmx scanner.cmx parser.cmx codegen.cmx ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
semant.cmo : ast.cmo
semant.cmx : ast.cmx
parser.cmi : ast.cmo

# Building the tarball

TESTS = add1 arith1 arith2 arith3 fib for1 for2 func1 func2 func3       \
    func4 func5 func6 func7 func8 gcd2 gcd global1 global2 global3      \
    hello if1 if2 if3 if4 if5 local1 local2 ops1 ops2 var1 var2         \
    while1 while2

FAILS = assign1 assign2 assign3 dead1 dead2 expr1 expr2 for1 for2       \
    for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 func8      \
    func9 global1 global2 if1 if2 if3 nomain return1 return2 while1     \
    while2

TESTFILES = $(TESTS:%=test-%.num) $(TESTS:%=test-%.out) \
            $(FAILS:%=fail-%.num) $(FAILS:%=fail-%.err)

TARFILES = ast.ml codegen.ml Makefile numnum.ml parser.mly README scanner.mll \
        semant.ml testall.sh $(TESTFILES:%=tests/%)

numnum-llvm.tar.gz : $(TARFILES)
        cd .. && tar czf numnum-llvm/numnum-llvm.tar.gz \
                $(TARFILES:%=numnum-llvm/%)
```