MPL

Jiangfeng Wang, David Rincon-Cruz, Wode ‘Nimo” Ni, Chi Zhang

Overview

Embracing the irrefutable correlation between novelty and quality
Essential
Shiny New Things
O RLY?
@ThePracticalDev

What is MPL?

LLVM

- MPL compiles to LLVM IR
- LLVM is flexible and works across multiple platforms

Motivation

- C/Java/Matlab - like Syntax
- Programmable Matrix Operations
- Lightweight and intuitive without math background

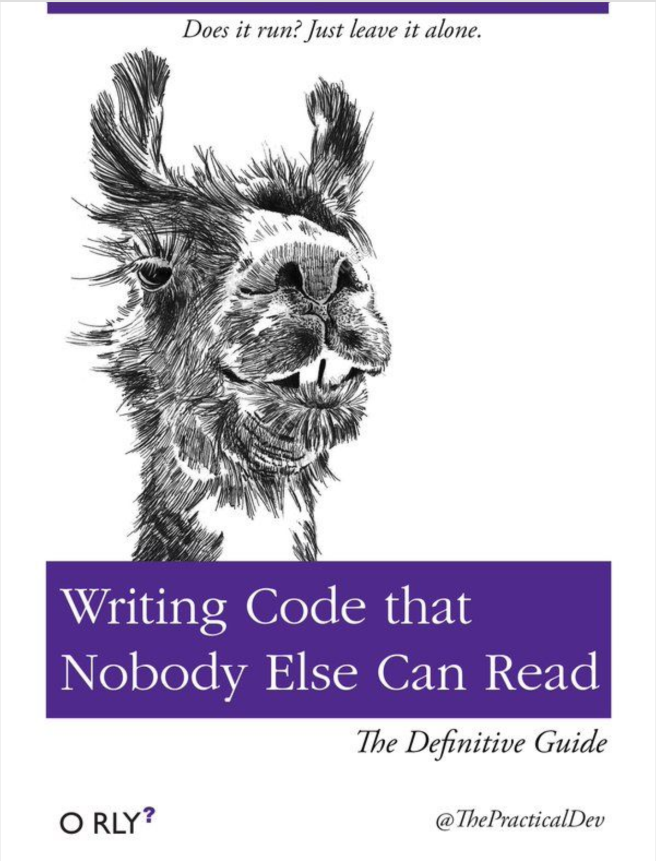
Matrices

- Matrix Arithmetic
- Apply Function

Images

- Reading in images
- Manipulating Pixels
- Writing images

Language Syntax


Does it run? Just leave it alone.
Writing Code that Nobody Else Can Read
The Definitive Guide
O RLY?
@ThePracticalDev

Programming in MPL

Comments

/* This is a comment*/

Primitives

int, float, bool, void, string, Mat

Control Flow

if, else, while, return

Arithmetic Operator

+ - * / = ++ --

Conditional Operator

== != > < >= <=

Logical Operator

!, &&, ||

Matrix

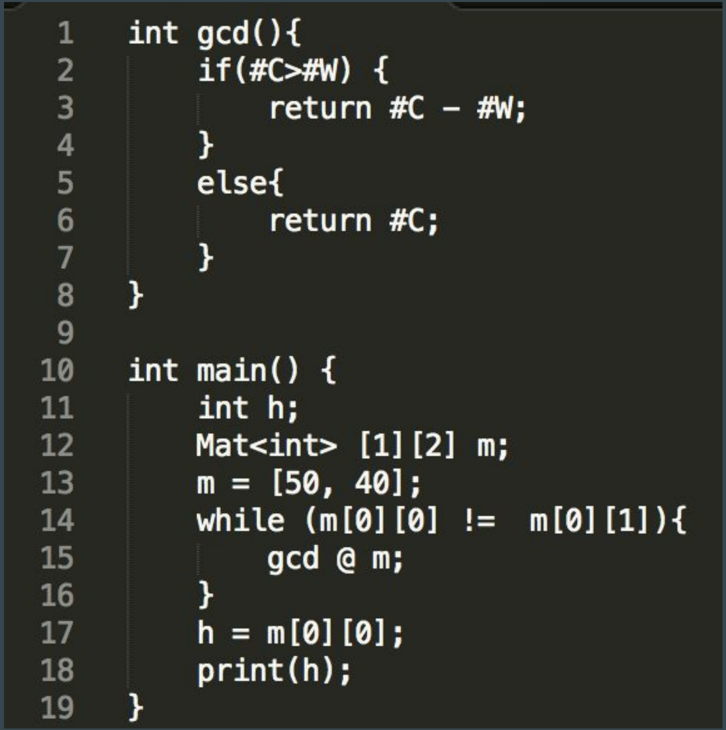
[1,2;3,4] [1.5,2.5;3.5,4.5]

Entry function

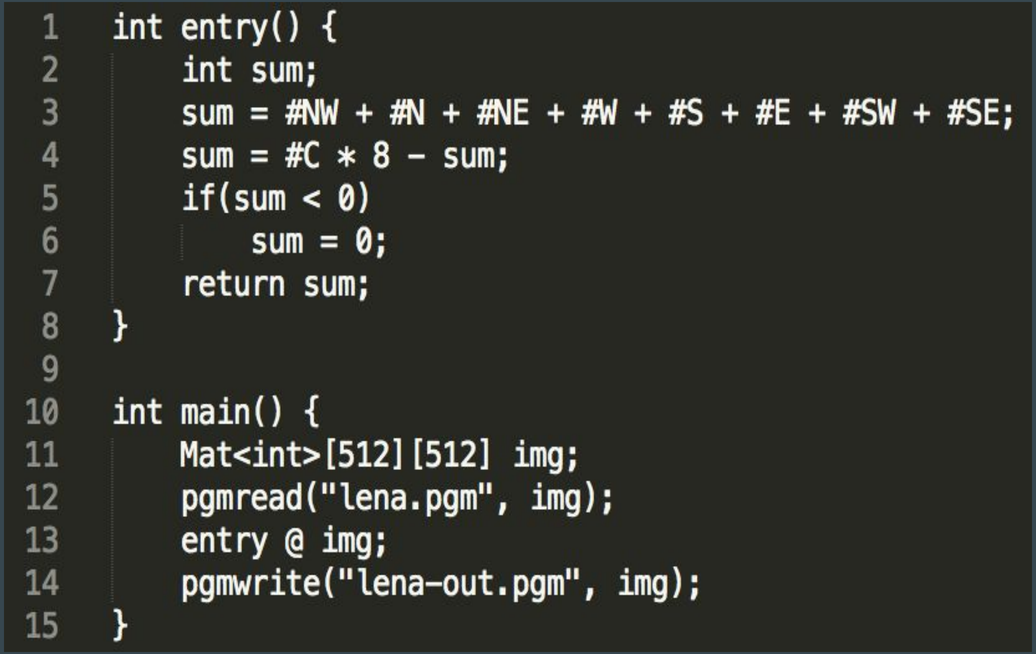
```
int entryf() {
        return 1;
}
```

Sample MPL program

Calculating GCD

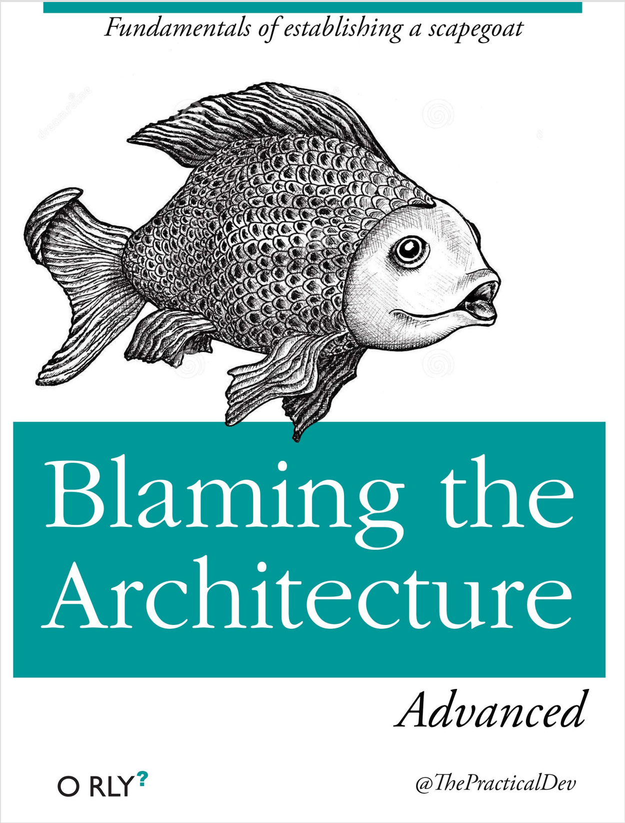


```
int gcd(){
    if(#C>#W) {
        return #C - #W;
    }
    else{
        return #C;
    }
}

int main() {
    int h;
    Mat<int> [1][2] m;
    m = [50, 40];
    while (m[0][0] !=  m[0][1]){
        gcd @ m;
    }
    h = m[0][0];
    print(h);
}
```



```
int entry() {
    int sum;
    sum = #NW + #N + #NE + #W + #S + #E + #SW + #SE;
    sum = #C * 8 - sum;
    if(sum < 0)
        sum = 0;
    return sum;
}

int main() {
    Mat<int>[512][512] img;
    pgmread("lena.pgm", img);
    entry @ img;
    pgmwrite("lena-out.pgm", img);
}
```

Architecture



Fundamentals of establishing a scapegoat

Blaming the Architecture

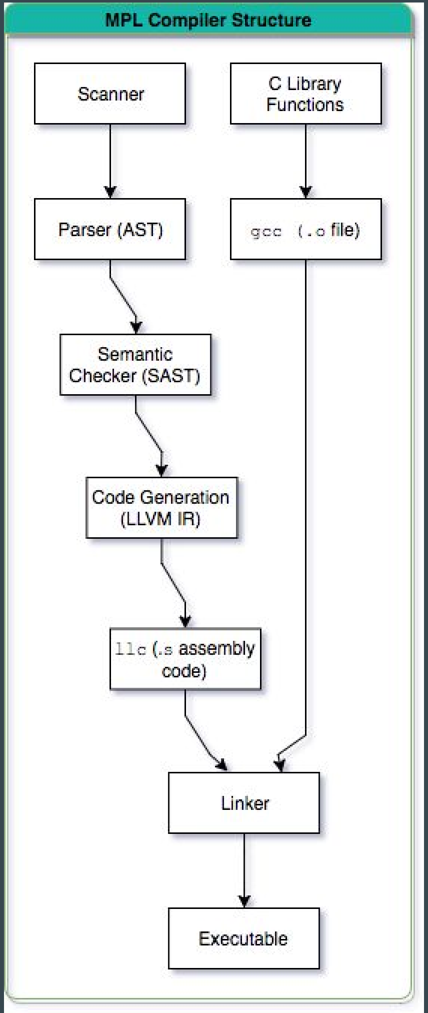
Advanced

O RLY?

@ThePracticalDev

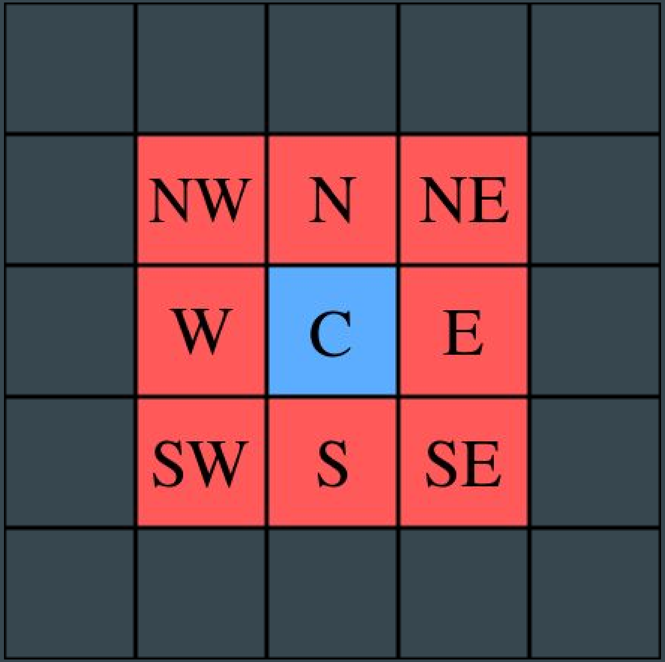
Architecture

- ➔ Added SAST for matrix dimensional information inferred by Semant
- ➔ C functions for image and console IO
- ➔ Not too different from MicroC
- ➔ Generating code for the Apply operator

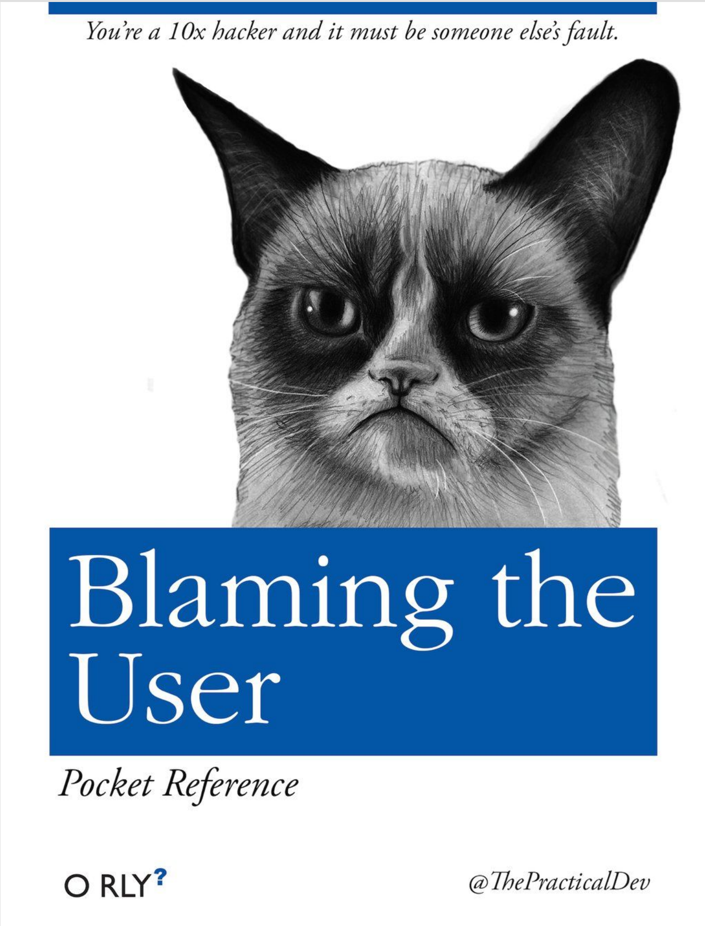


Generating Code for Entry functions

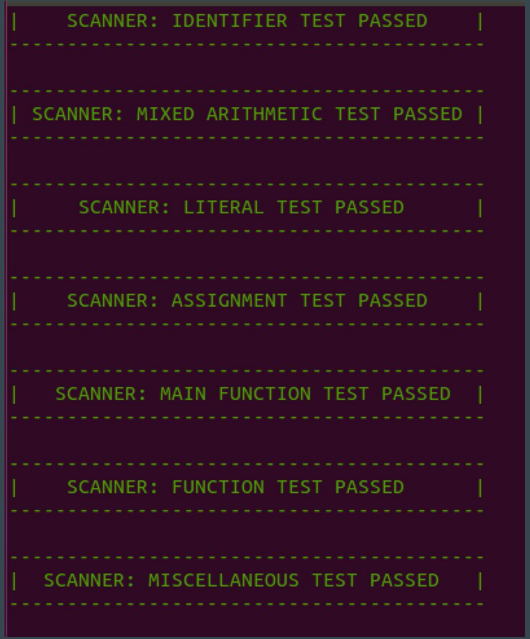
- `<function> @ <Mat>`
- Generate while loops over the target matrix
- neighbors passed in by value
- Moore neighborhood
- Edge problem: a torus!



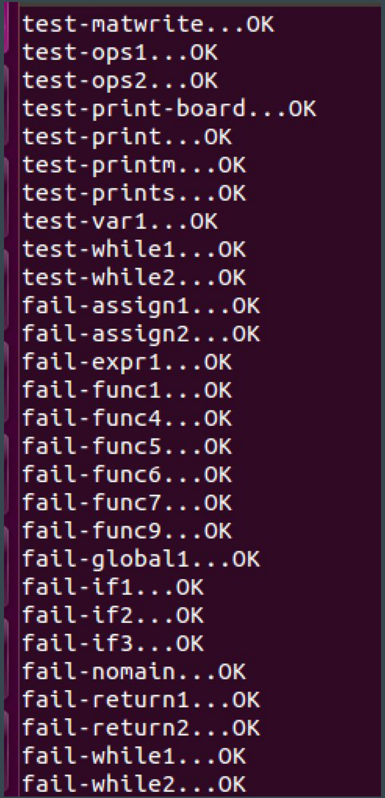
Testing


You're a 10x hacker and it must be someone else's fault.
Blaming the User
Pocket Reference
O RLY?
@ThePracticalDev

Testing



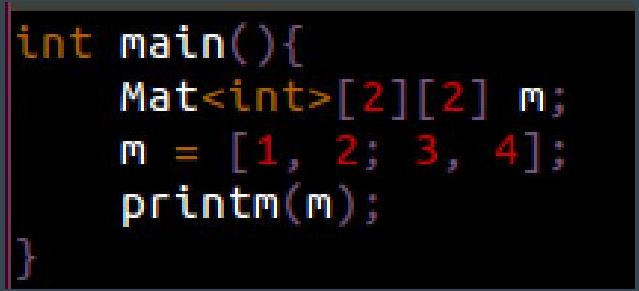
```
|    SCANNER: IDENTIFIER TEST PASSED    |
-----------------------------------------

-----------------------------------------
| SCANNER: MIXED ARITHMETIC TEST PASSED |
-----------------------------------------

-----------------------------------------
|     SCANNER: LITERAL TEST PASSED      |
-----------------------------------------

-----------------------------------------
|    SCANNER: ASSIGNMENT TEST PASSED    |
-----------------------------------------

-----------------------------------------
|   SCANNER: MAIN FUNCTION TEST PASSED  |
-----------------------------------------

-----------------------------------------
|    SCANNER: FUNCTION TEST PASSED      |
-----------------------------------------

-----------------------------------------
|  SCANNER: MISCELLANEOUS TEST PASSED   |
-----------------------------------------
```



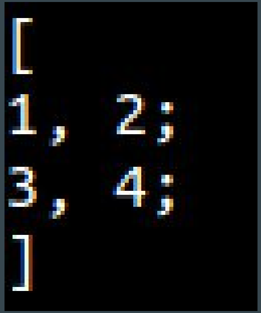
```
test-matwrite...OK
test-ops1...OK
test-ops2...OK
test-print-board...OK
test-print...OK
test-printm...OK
test-prints...OK
test-var1...OK
test-while1...OK
test-while2...OK
fail-assign1...OK
fail-assign2...OK
fail-expr1...OK
fail-func1...OK
fail-func4...OK
fail-func5...OK
fail-func6...OK
fail-func7...OK
fail-func9...OK
fail-global1...OK
fail-if1...OK
fail-if2...OK
fail-if3...OK
fail-nomain...OK
fail-return1...OK
fail-return2...OK
fail-while1...OK
fail-while2...OK
```

- → Scanner test and Program test
- → MicroC’s style of test is efficient.

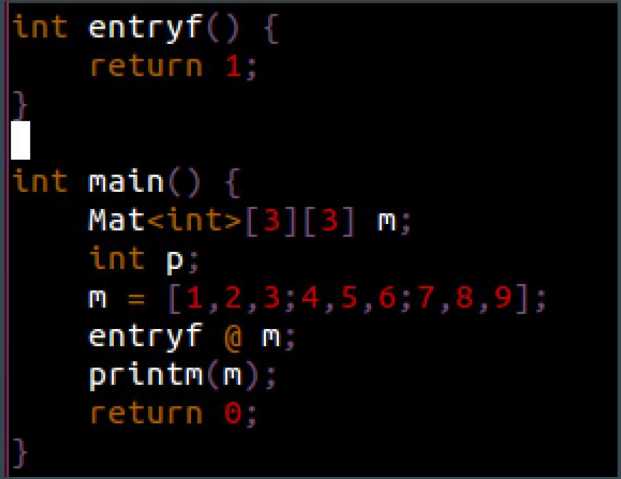
- → For our language, printm() is the most useful function for testing.
- → Example : @ Apply test



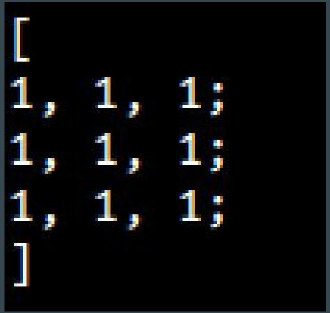
```
int main(){
    Mat<int>[2][2] m;
    m = [1, 2; 3, 4];
    printm(m);
}
```



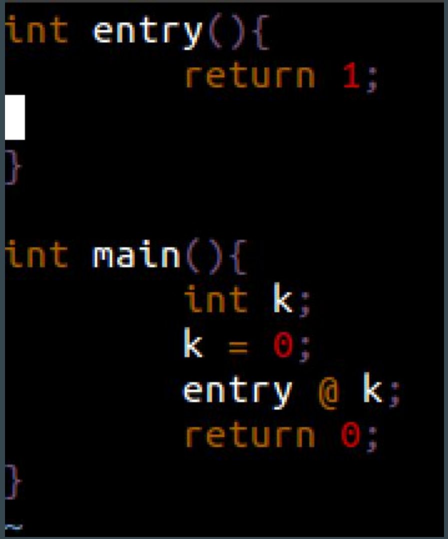
```
[
1, 2;
3, 4;
]
```



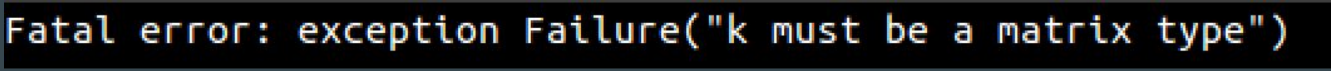
```
int entryf() {
    return 1;
}

int main() {
    Mat<int>[3][3] m;
    int p;
    m = [1,2,3;4,5,6;7,8,9];
    entryf @ m;
    printm(m);
    return 0;
}
```



```
[
1, 1, 1;
1, 1, 1;
1, 1, 1;
]
```

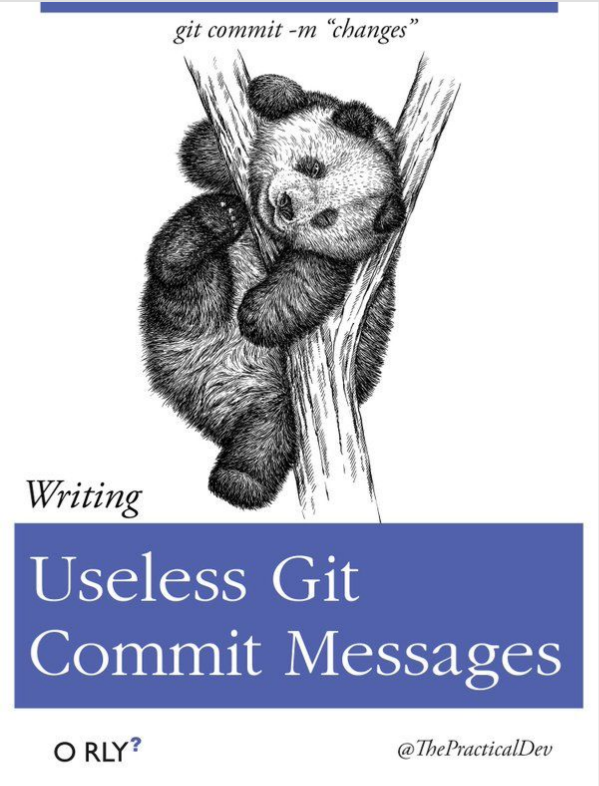


```
int entry(){
        return 1;

}

int main(){
        int k;
        k = 0;
        entry @ k;
        return 0;
}
~
```

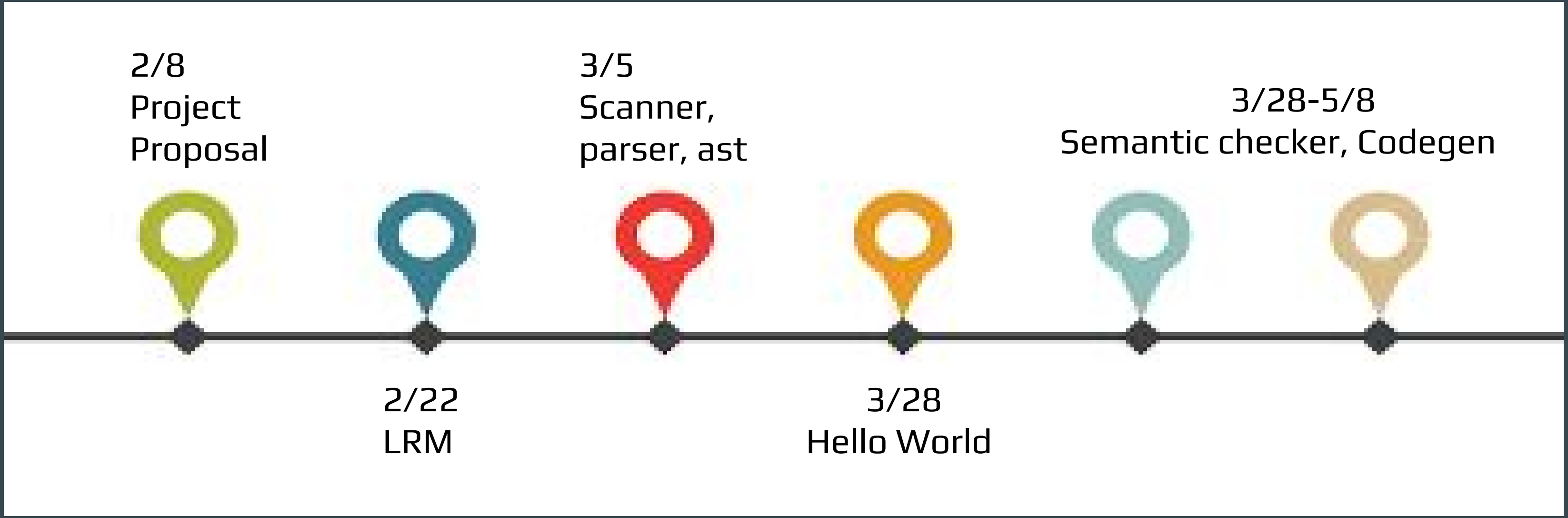


```
Fatal error: exception Failure("k must be a matrix type")
```

Project Management



Project Timeline



2/8
Project
Proposal

2/22
LRM

3/5
Scanner,
parser, ast

3/28
Hello World

3/28-5/8
Semantic checker, Codegen

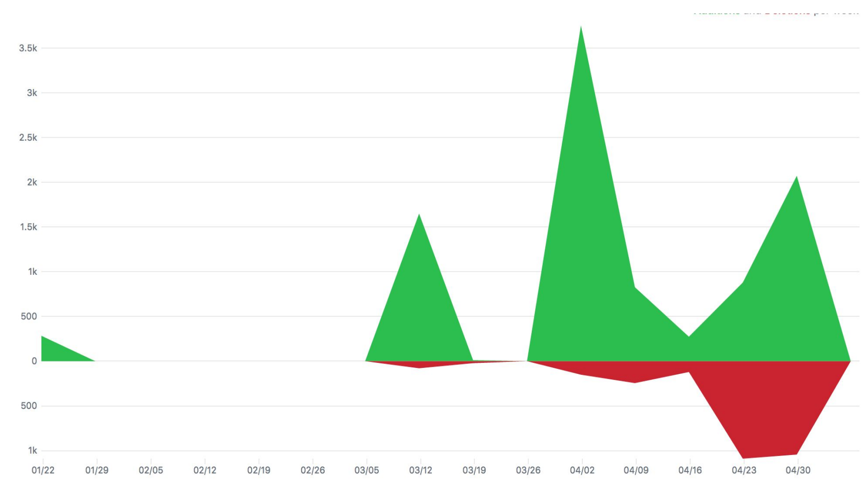
Project Management

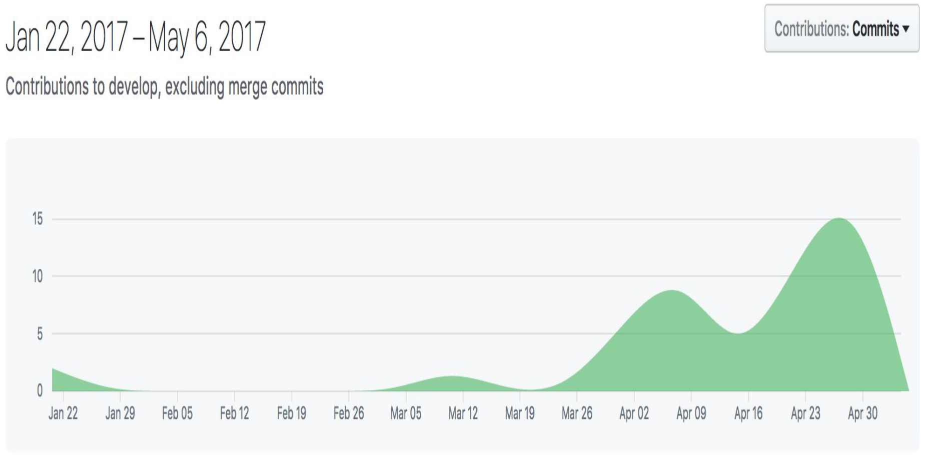
- 3-4 weekly meetings
- TA advising meetings
- Dividing tasks and pair programming
- Multiple branches

Contribution

Jiangfeng and David: Design, scanner, parser, ast, semantic checker, sast

Nimo and Chi: Skeleton of Scanner and Parser, Codegen, example programs, test suite, game of life





Jan 22, 2017 – May 6, 2017

Contributions to develop, excluding merge commits

Lessons Learned

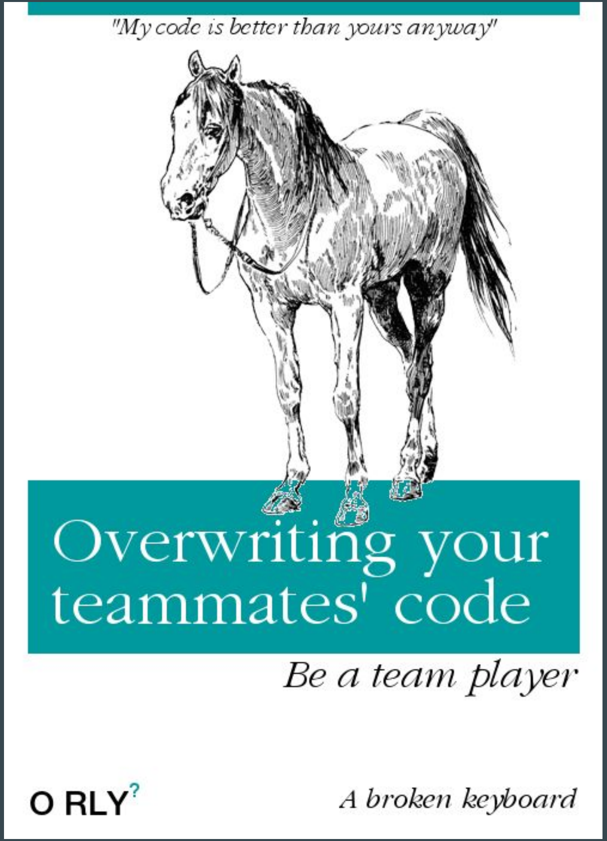
Lessons Learned

Jiangfeng: Start early. Micro C and previous projects are extremely helpful as sources of instruction.

David: It’s better to argue out the features of the language so that everyone is on board. Pair programming keeps everyone on board and provides sanity checks.

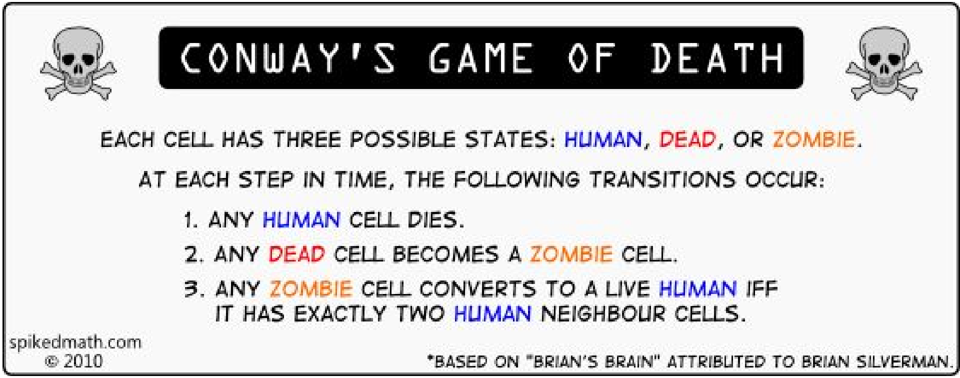
Chi: Understanding of code is important. Especially when you try to learn from previous project.

Nimo: Frequency of the meetings is important. Incremental development is always better than merging big chunks of code


"My code is better than yours anyway"
Overwriting your teammates' code
Be a team player
O RLY?
A broken keyboard

Conway's Game of Life

- ➔ Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
- ➔ Any live cell with two or three live neighbours lives on to the next generation.
- ➔ Any live cell with more than three live neighbours dies, as if by overpopulation.
- ➔ Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
- ➔ There are known patterns



CONWAY'S GAME OF DEATH

EACH CELL HAS THREE POSSIBLE STATES: HUMAN, DEAD, OR ZOMBIE.

AT EACH STEP IN TIME, THE FOLLOWING TRANSITIONS OCCUR:

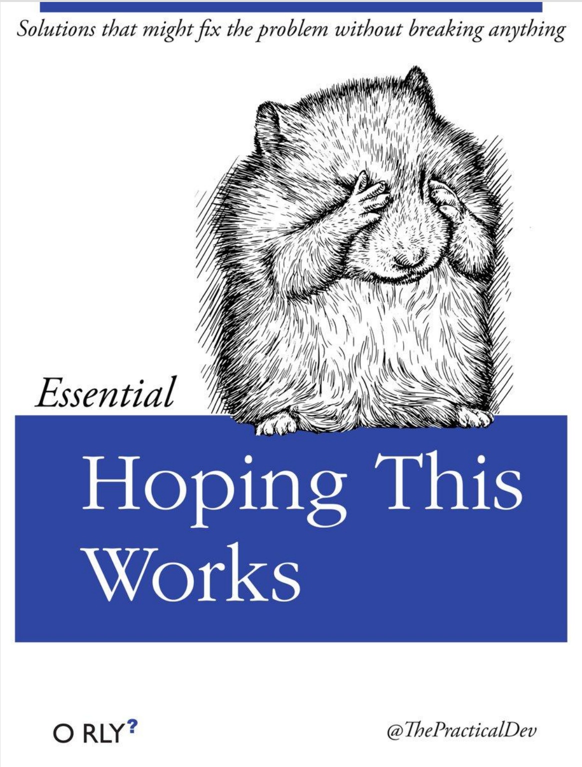
1. ANY HUMAN CELL DIES.
2. ANY DEAD CELL BECOMES A ZOMBIE CELL.
3. ANY ZOMBIE CELL CONVERTS TO A LIVE HUMAN IFF IT HAS EXACTLY TWO HUMAN NEIGHBOUR CELLS.

spikedmath.com
© 2010

*BASED ON "BRIAN'S BRAIN" ATTRIBUTED TO BRIAN SILVERMAN.

Demo

- ➔ Image Convolution
- ➔ Game of Life Simulation


Solutions that might fix the problem without breaking anything
Essential
Hoping This Works
O RLY?
@ThePracticalDev