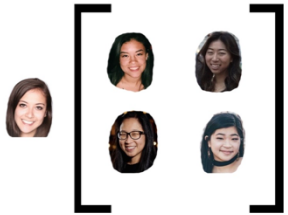


J-STEM

A Matrix Manipulation Language

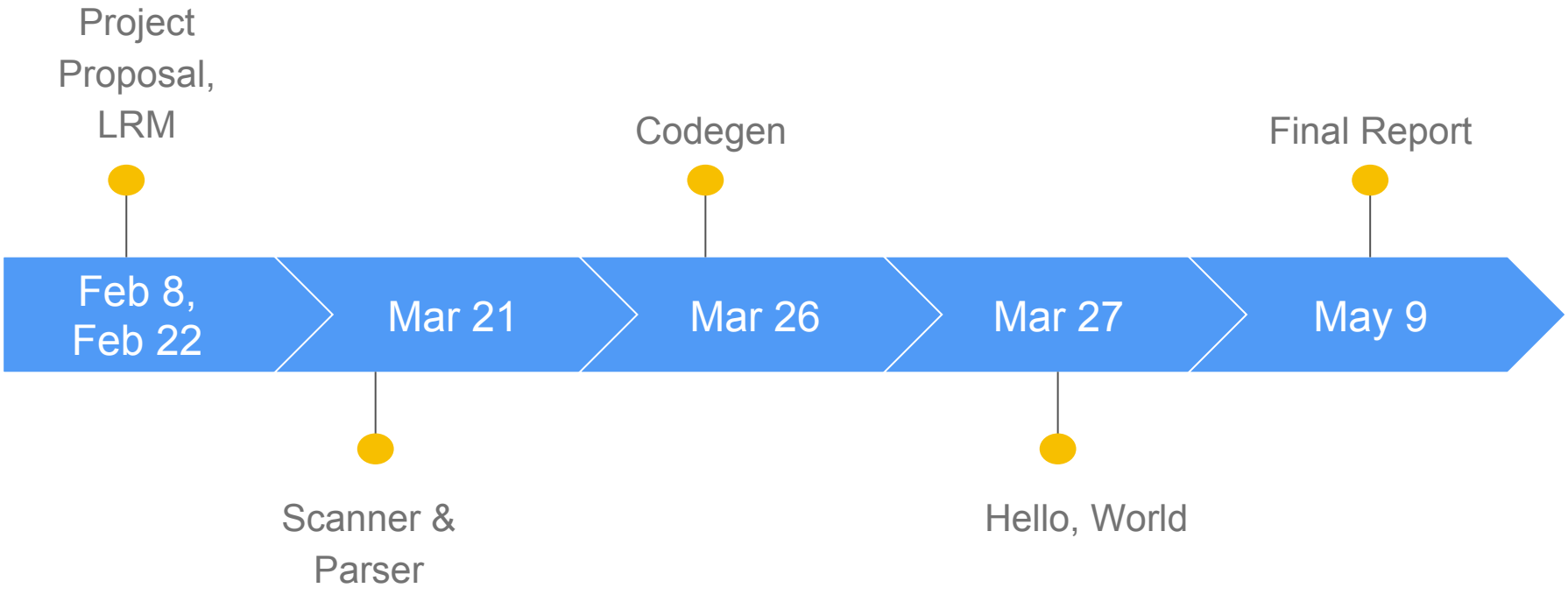
Julia Troxell (jst2152) - **S**am Stultz (sks2200) **T**essa Hurr (trh2124)
Emily Song (eks2138) **M**ichelle Lu (ml3720)

Introduction

J-STEM is an imperative programming language that facilitates matrix manipulation. Its key features are as follows:

- ❏ Strongly typed
- ❏ The main datatype is a matrix
- ❏ Standard library focused on matrices
- ❏ Compiles to the Low Level Virtual Machine (LLVM)

Timeline


Project Proposal, LRM
Feb 8, Feb 22
Scanner & Parser
Mar 21
Codegen
Mar 26
Hello, World
Mar 27
Final Report
May 9

Language Overview

Primitive Types:
int, float, bool, char, string

Data Types:
Rows and Matrices

Declaration/Initialization:

```
int z;
z = 3;
```

Tuple:

```
int(%3%) t;
T = (% 1, 2, 3 %);
```

Row:

```
int[5] x;
x[0] = 4;
```

Matrix:

```
int[2][2] y;
y[0][1] = 4;
```

Function Declaration:

```
def int multiply(int a,
int b) {
        /* function */
}
```

Operators:

Standard Arithmetic Operators
Scalar Matrix Operations
Arithmetic Matrix Operations

$ access pointer
$$
dereference pointer
~~ pointer increment

File Extension: .JSTEM

Control Flow:

```
if (True){
            print (42);
}
else{
            print(8);
}

while (a>0){
            a = a - 1;
}

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

int[2][2] m;
m = {% 1, 2 | 3, 4 %};
for (row in m) {
            print_rowi(row);
}
```

Hello, World!

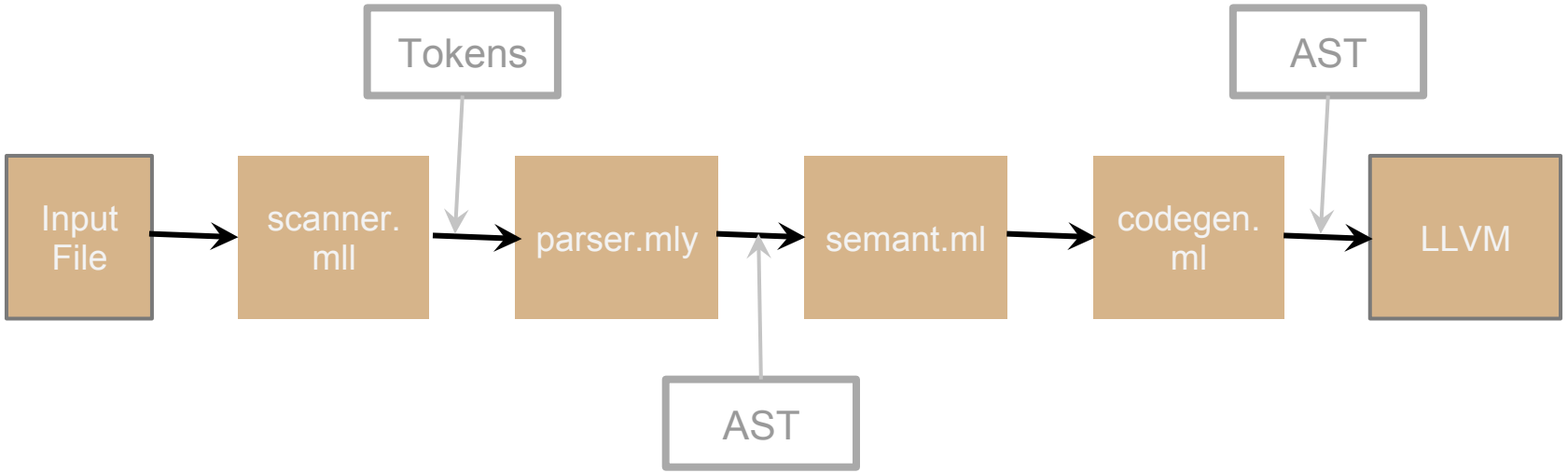
```
def int main() {
        prints(“Hello world”);
        return 0;
}
```

Brief Tutorial

```
def int main() {

    int[4] int_row;
    int[2][4] int_matrix;

    float[3] float_row;
    float[2][3] float_matrix;

    int(%3%) t;
    int(%3%)[2] tuple_row;
    int(%3%)[2][2] tuple_matrix;

    int_row = [1,2,3,4];
    int_matrix = {% 1,2,3,4 | 5,6,7,8 %};

    float_row = [1.1,2.2,3.3];
    float_matrix = {% 1.1,2.2,3.3 | 4.4,5.5,6.6 %};

    t = (%1,2,3%);
    tuple_row = [(%1,2,3%),(%3,4,5%)];
    tuple_matrix = {% (%1,2,3%), (%4,5,6%) | (%7,8,9%), (%10,11,12%) %};

    prints("ints: \n");
    print_rowi($int_row, int_row.length);
    prints("\n");
    print_matrixi($$int_matrix, int_matrix.length, int_matrix.width);
    prints("\n");

    prints("floats: \n");
    print_rowf($float_row, float_row.length);
    prints("\n");
    print_matrixf($$float_matrix, float_matrix.length, float_matrix.width);
    prints("\n");

    prints("tuples: \n");
    print_tuple(t);
    prints("\n");
    print_rowt($tuple_row, tuple_row.length);
    prints("\n");
    print_matrixt($$tuple_matrix, tuple_matrix.length, tuple_matrix.width);
    prints("\n");

}
```

```
ints:
[ 1, 2, 3, 4 ]
{ 1, 2, 3, 4
  5, 6, 7, 8 }
floats:
[ 1.100000, 2.200000, 3.300000 ]
{ 1.100000, 2.200000, 3.300000
  4.400000, 5.500000, 6.600000 }
tuples:
( 1, 2, 3 )
[ ( 1, 2, 3 ), ( 3, 4, 5 ) ]
{ ( 1, 2, 3 ), ( 4, 5, 6 )
  ( 7, 8, 9 ), ( 10, 11, 12 ) }
```

Architecture



Testing

Testing automation program that goes through all of the “compiler_tests” and runs them

Tests Directory

- Compiler Tests: compiler_testing.sh
  - Success & fail tests included
- Scanner Tests: scanner_testing.sh
- Parser Tests: parser_testing.sh

Continuous integration with **Travis CI**

Lessons Learned

Tessa: *Ask a lot of questions because there isn't much documentation, keep asking questions if you don't understand*

Julia: *The best way to understand the code is to try to write your own functions. You'll run into tons of bugs but debugging forces you to really get to know the code*

Michelle: *Continuously added in-depth failing and successful tests as you add each new feature to catch bugs early on, add descriptive fail exceptions and errors to help you debug*

Emily: *Understand OCaml programming, how the components of the compiler fit together, and set concrete goals/deadlines in order to always be making progress*

Sam: *The importance of using version control and having good team communication*

Demo