MatrixMania
Matrix Programming Language

Cindy Espinosa, Desu Imudia, Diego Prado,
Sophie Reese-Wirpsa, Emily Ringel
Our Team

Cindy Espinosa
Tester

Desu Imudia
Language Guru

Diego Prado
System Architect

Emily Ringel
System Architect

Sophie Reese-Wirpsa
Project Manager
The Language
The Motivation

- **Matrix based problems**
- **Flexibility** with approach to problem solving
- **Simplicity** of syntax
- **Familiarity** with preceding languages
So, what is MatrixMania?

- imperative programming language
- matrix manipulation
- linear algebra calculations
- Java-like syntax
MatrixMania
Features

Matrix Data Type
Simple Control Flow

Matrix Functions
Basic Lexical Conventions

Matrix Operations
C-Programming Compatibility
Language Overview

Data Types
int, float, matrix, void

Comments
/* This is a comment in MatrixMania */

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

Function

Declarations & Scope
def int main() { return 0; }
def void example () {}

Control Flow
for (int i = 0; i < 2; i = i + 1) {}  
while (i < 2) {}  
if (i < 2) {}  
elif (i == 3) {}  
else {}
**Language Overview: Matrix Literal**

**MatrixMania Assignment**

```cpp
matrix<int> m = [1, 2; 3, 4];
matrix<float> n = [1.2, 7.3, 5.36];
```

**Matrix Access & Reassign**

```cpp
int a = m[1, 0];
m[0,0] = 5;
```

**Matrix Operations**

```cpp
int r = getRows(m);
int c = getColumns(m);
matrix <int> new_mat_add = m + n;
matrix <int> new_mat_sub = m - n;
matrix <int> new_mat_mult = m * n;
matrix <int> new_mat_scal = 2 * m;
```
The Implementation
Process

Weekly Meetings
- discussed each member’s progress over the week
- set deliverables for the next week

MatrixMania Functionality
- adapted MicroC code with features unique to our language
- used previous understanding of linear algebra computation to write built-in functions

MicroC Dissection
- referenced code for building blocks such as control flow and int/float operations

Program Testing
- added new tests iteratively with new functionality
- confirmed each new update did not break existing tests
Architecture

- Scanner
- Parser
- AST
- SAST
- Semantic
- Codegen
- LLVM
- C-Code
# Implementation Highlights

```c
def int main()
{
    int a = 5;
    float b = a;
    float c = b - 3;
    float d = b;
    matrix<float> m1 = [0.0, b; c, d];
    matrix<float> m2 = [1.0, 1.0; 1.0, 1.0];
    printf(m1*m2);
}
```

Output:

```
5.0 5.0
7.0 7.0
```
Implementation Highlights

```c
#include <stdio.h>

int main()
{
    int a = 5;
    float b = a;
    float c = b - 3;
    float d = b;
    matrix<float> m1 = [0.0, b; c, d];
    matrix<float> m2 = [1.0, 1.0; 1.0, 1.0];
    printf("%f %f\n%f %f\n", m1[0][0], m1[0][1], m1[1][0], m1[1][1]);
}
```

Flexible Variable Declaration

Int -> Float Casting

Matrix Operations

Internal Matrix Representation

Output:

```
5.0 5.0
7.0 7.0
```
Implementation Highlights

```c
int main()
{
    int a = 5;
    float b = a;
    float c = b - 3;
    float d = b;
    matrix<float> m1 = [0.0, b; c, d];
    matrix<float> m2 = [1.0, 1.0; 1.0, 1.0];
    printf(m1*m2);
}
```

Output:

```
5.0 5.0
7.0 7.0
```

Flexible Variable Declaration

Int -> Float Casting

Matrix Operations

Internal Matrix Representation
Implementation Highlights

```python
def int main(){
    int a = 5;
    float b = a;
    float c = b - 3;
    float d = b;
    matrix<float> m1 = [0.0, b; c, d];
    matrix<float> m2 = [1.0, 1.0; 1.0, 1.0];
    printmf(m1*m2);
}
```

m1 in LLVM/C:

```
[2, 2, 0.0, 5.0, 2.0, 5.0]
```

Flexible Variable Declaration

Int -> Float Casting

Matrix Operations

Internal Matrix Representation
The Future
Future Work

Create Standard Library

Empty Matrix

Garbage Collection
The Code
Demo

**Transpose a Matrix**

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6 \\
\end{bmatrix}^T = \begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6 \\
\end{bmatrix}
\]

**Invert a Matrix**

\[
\begin{bmatrix}
1 & 0 & -1 & 2 \\
0 & 1 & 1 & -1 \\
\end{bmatrix} \rightarrow \begin{bmatrix}
-1 & 2 \\
1 & -1 \\
\end{bmatrix}
\]
Thank you!

Questions?