MX

A C-like Matrix Manipulation Language

By: Aaron Jackson (arj2145), Wilderness Oberman (wo2168),
Rashel Rojas (rdr2139), & Mauricio Guerrero (mg4145)

Motivation

Matrices are **tedious**

C is even **more tedious**

Handling Matrices in C is **downright unbearable**

The Solution: MX

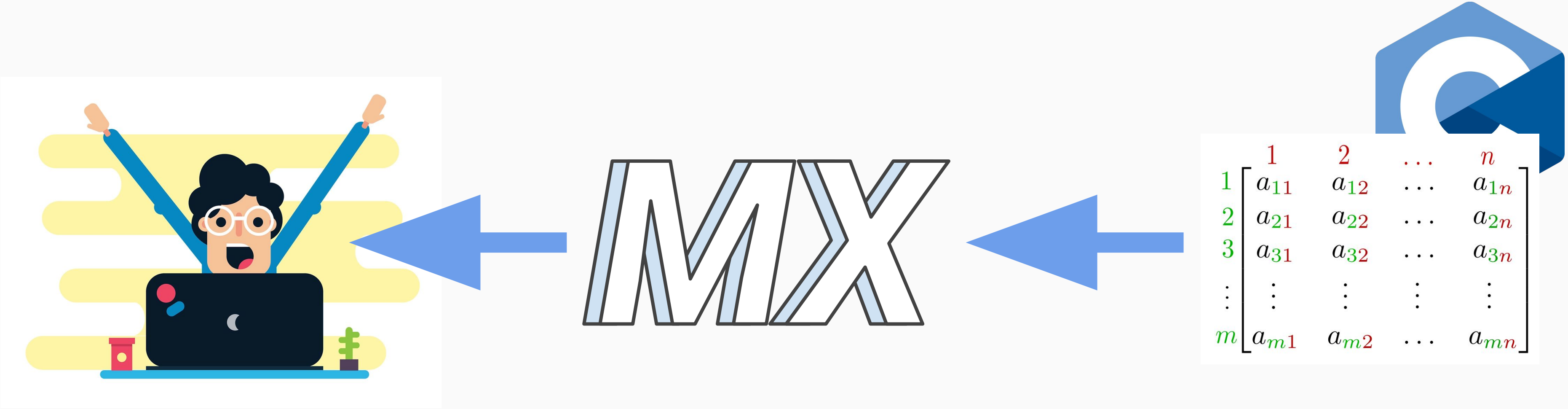
C-Like Syntax

- Familiar to programmers
- Modular!
  - Matrices as important as you want them to be.
  - Free to make regular C-style programs

Matrices

- Built-in Matrix Data Type
  - Intuitive
  - Lightweight
- Robust Matrix Library
  - Automates tedious Matrix operations

Simplified MX Architecture



$$\begin{array}{c|cccc} & 1 & 2 & \ldots & n \\ \hline 1 & a_{11} & a_{12} & \ldots & a_{1n} \\ 2 & a_{21} & a_{22} & \ldots & a_{2n} \\ 3 & a_{31} & a_{32} & \ldots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m & a_{m1} & a_{m2} & \ldots & a_{mn} \end{array}$$

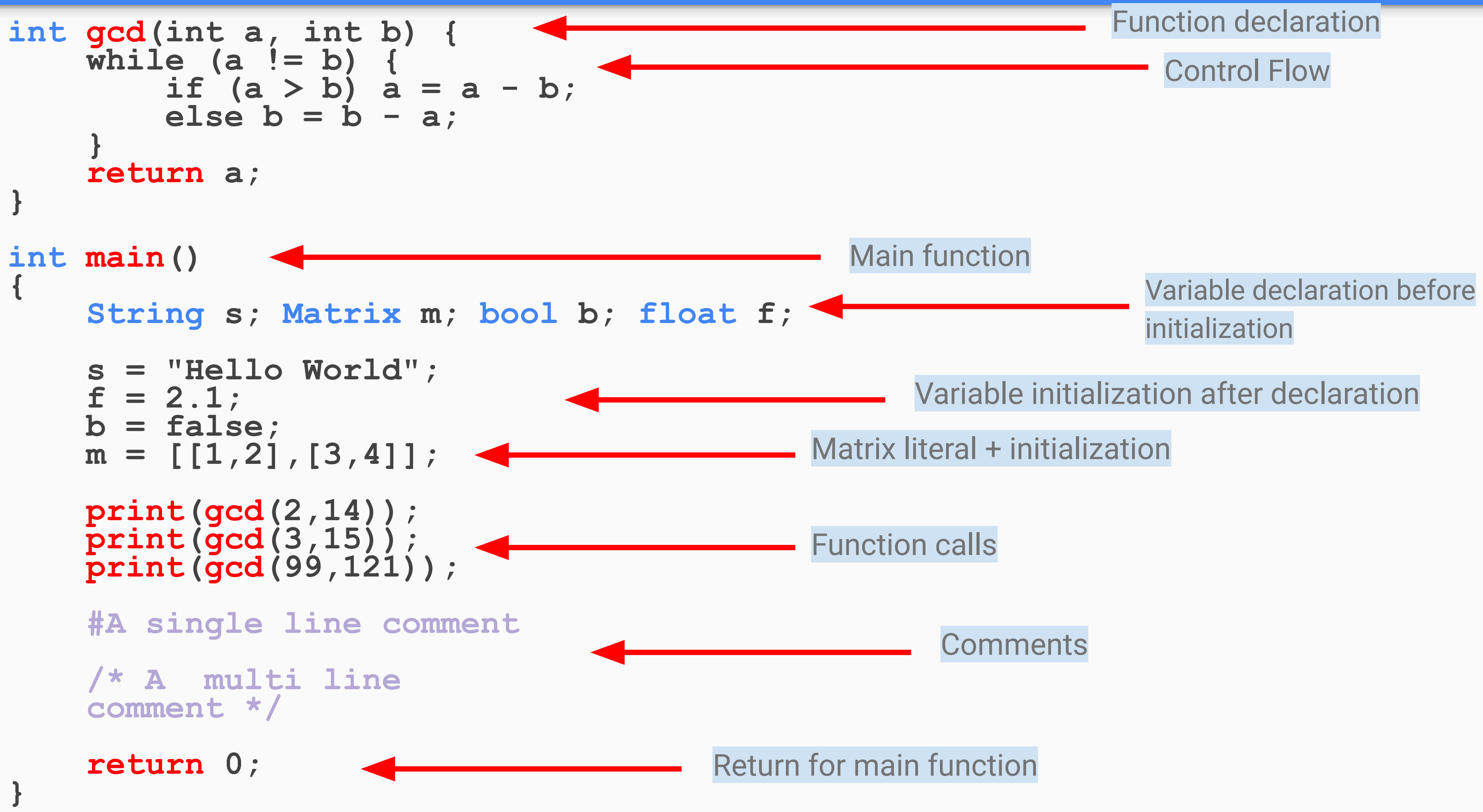
Happy Programmer

Intuitive MX syntax

Verbose C Matrix Handling

MX streamlines the use of a pre-existing C Matrix Library

Language Overview



```
int gcd(int a, int b) {
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}

int main()
{
    String s; Matrix m; bool b; float f;

    s = "Hello World";
    f = 2.1;
    b = false;
    m = [[1,2],[3,4]];

    print(gcd(2,14));
    print(gcd(3,15));
    print(gcd(99,121));

    #A single line comment

    /* A  multi line
    comment */

    return 0;
}
```

Function declaration

Control Flow

Main function

Variable declaration before initialization

Variable initialization after declaration

Matrix literal + initialization

Function calls

Comments

Return for main function

Language overview: Data Types + Operators

- Types: int, float, boolean, strings, matrices
  - Implicit casting between ints and floats to float for arithmetic operations
  - Variables must be declared before they are instantiated
- Unary operators: !, - (negation)
- Arithmetic operators: +, -, /, *
- Relational operators: >, <, >=, <=, ==, !=
- Logical operators: &&, ||, !
- Assignment operators: +=, -=, *=

```
int x;
bool b;
float f;
String s;
Matrix m;

i = 3;
f = 4.2 + 3; #
7.2
b = false;
s = “mx”;
m =
[[1,2],[3,4]];
```

Language overview: Built-in functions & Control Flow

```
main()

print()

printb()

prints()

printf()

pi()
```

* matrix built-in functions in the next few slides

```
if (boolean condition) {
      body;
}
```

```
while (boolean condition) {
      body;
}
```

```
int i;
for (i = 0; i < 10; i += 1 ) {
      body;
}
```

Language overview: Matrix Data Type

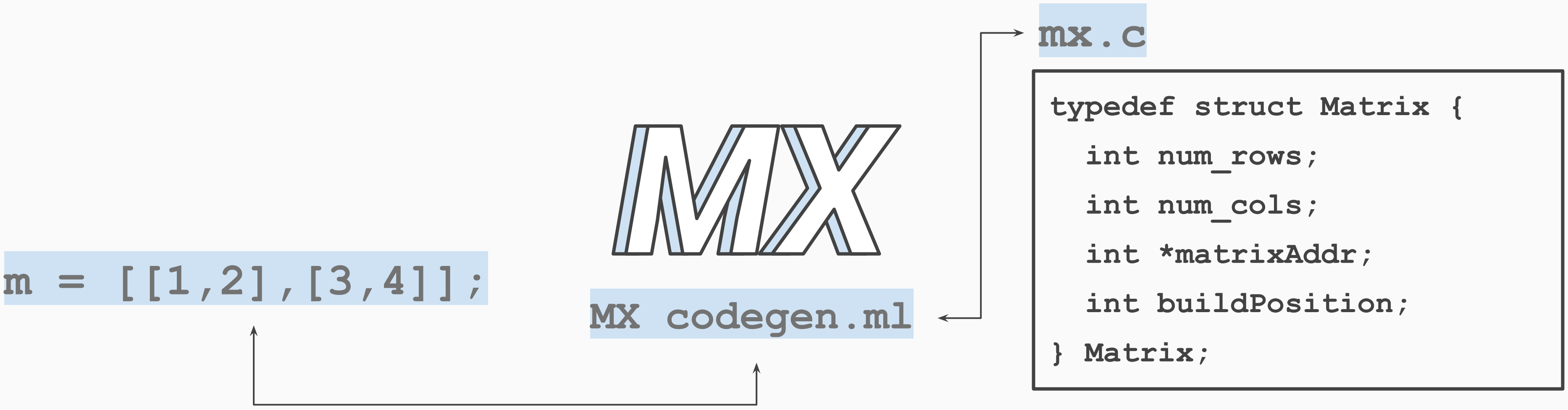
Matrix Declaration:

```
Matrix m;

/*Matrix of ints only*/
```

Matrix Initialization:

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

/* Each list of elements
corresponds to a row in the
matrix */
```

Language overview: Matrices Data Type



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

MX codegen.ml

mx.c

```
typedef struct Matrix {
   int num_rows;
   int num_cols;
   int *matrixAddr;
   int buildPosition;
} Matrix;
```

Created a C library consisting of matrix functions and linked it to our compiler through codegen

Language overview: Matrix Library

```
Matrix m1;
Matrix m2;
Matrix m3;

m1 = [[1,1],[2,2]];
m2 = [[3,3],[4,4]];

m3 = m1 +. m2;
m3 = m1 -. m2;
m3 = m1 *. m2;
m3 = m1 **. 2;
m3 = m1’;
m3 = identity(2);
m3 = transformation(m1, 1);
m3 = transformation(m1, 2);
m3 = transformation(m1, 3);
m3 = transformation(m1, 4);
m3 = transformation(m1, 5);
m3 = transformation(m1, 6);
m3 = transformation(m1, 7);
```

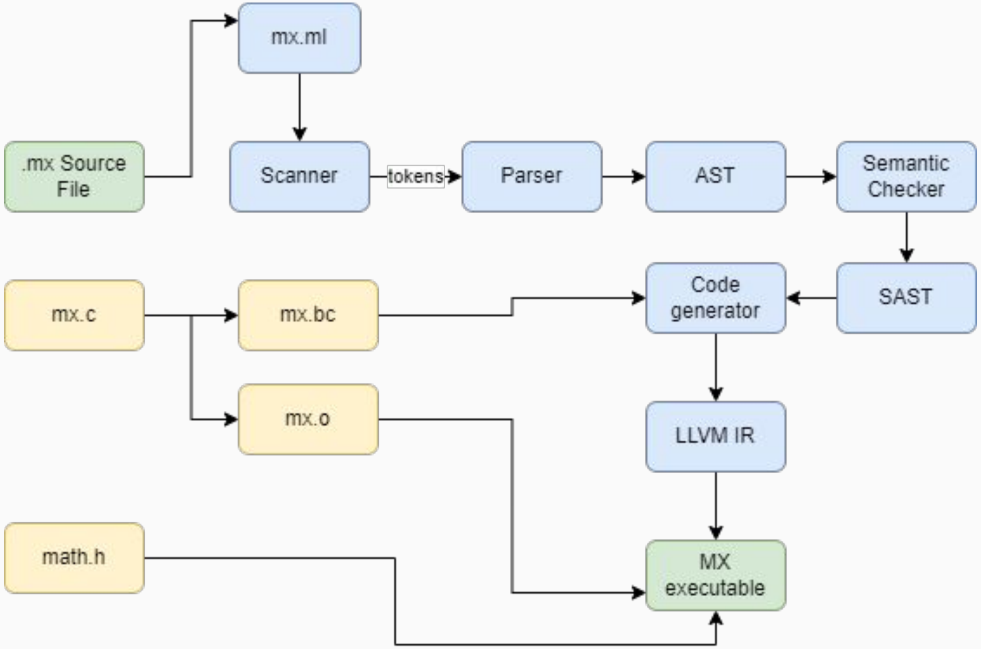
- Add
- Subtract
- Matrix multiplication
- Scalar multiplication
- Transpose
- Identity
- Reflection
  - line y = x
  - line y = -x
  - X-axis
  - Y-axis
- Rotations:
  - 90° (anti)clockwise
  - 180°

Language overview: Matrix Library

```
Matrix m;

m = [[2,4],[3,6],[4,8]];

print_matrix(m);
/*
[ 2, 4 ]
[ 3, 6 ]
[ 4, 8 ]
*/

print(numRows(m)); # 3
print(numCols(m)); # 2
```

- ➢ Print matrix
- ➢ Get the number of rows
- ➢ Get the number of cols

Compiler Architecture: Overview



Semant + Codegen

Matrix Error Checking

```
| Mx l        ->

      let rows = List.length l in

      let cols = List.length (List.hd l) in

      let col_check list = List.map (fun v
-> if List.length v != cols then raise
(Failure "Matrix rows are not all the same
length")) list in

      ignore(col_check l); (Matrix(Int), SMx
(l, rows, cols))
```

Arithmetic Operator Casting

```
| SBinop (((A.Float,_ ) as e1), op,
((A.Int,_) as e2)) ->
       let e1' = expr builder e1
       and e2' = expr builder e2 in
       (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 | A.Or | A.Mxadd | A.Mxsub |
A.Mxtimes | A.Mxscale ->
           raise (Failure "internal error:
semant should have rejected and/or on float")
       ) e1' (L.build_uitofp e2' float_t
"tmp" builder) "tmp" builder
```

Testing

- Created passing/failure test cases
    - output in .out and .err files, respectively
- Checks for semantic/syntax errors
- Demonstrates variable assignment, arithmetic operations, control flow, matrix operations, user-created functions, etc.
- Regression testing script (testall.sh) to test all test cases
    - Compares output file with expected output file

DEMO

Post Mortem

- Less verbose back end
- Implementing pointers, Arrays; Better Matrix structure
- Implement float matrices
- More matrix functions
    - Instantiate an empty matrix given number of rows and columns
    - Get a column/row from a matrix
    - rref
    - rank
    - horizontal/vertical shear
    - inverse
- More implicit casting
    - float/int for assignment

THANK YOU!

Thank you to Professor Edwards, all of the TAs, and the guy that made MicroC for your help!