Team Members

- William Hom
- Joseph Baker
- Christopher Chang
- Yi Jian
Introduction

Macaw is a mathematical calculation language with native support for matrix data types.

- Strongly typed
- Imperative
- Supports if/else/for/while flow controls
- Functions
- Operator overloading
Project Plan
Complete our project proposal

Finish LRM

Finalize Scanner, Parser, and AST

Compile “Hello World” into LLVM

Semantic checking, SAST generation, Codegen, finished

Features, test suite complete
A Macaw program is written as series of functions and imperative statements. Function definitions and variable declarations must be made prior to referencing them.

#Does not compile

```cpp
foo("Hello World");
void foo (string s){
    print(s);
}
```

#Compiles

```cpp
void foo (string s){
    print(s);
}
foo("Hello World");
```
Data Types

number - Floating point numbers for arithmetic operations.

string - Character strings used for printing statements to the console. Can be stored in variables or used as constants.

matrix - Two dimensional arrays of numbers.
  ● Built-in support - initialization, access, insertion
  ● Standard library functions implemented.
    ○ [flattened] indexing - counted across columns, then rows.
Language Overview / Tutorial

# string concatenation is destructive

```javascript
string foo(string s) {
    return strcat("Hello ", s);
}
```

print(foo("World!")); # prints 'Hello World!'

# testing additive expressions, subtraction

```javascript
matrix a <- [1,2,3;4,5,6];
matrix b <- [1,2,3;4,5,6];
matrix c <- a - b;
print(c[5]); # prints 0.00
```
More interesting features

- Matrix Support
- Operator Overloading
- Function Overloading
- Statements are valid at the root (outside the functions)
Some things our language can do

```plaintext
# mixed variables in overloads

number operator + (string s, number b) {
    print(b);
    print(s);
    return 0;
}

number a <- "Hello World!" + 6;
print(a);

# prints the following:
# 6.000
# Hello World!
# 0.000

# global variable modifications are visible in functions
# variables are passed by value

number a <- 7;

number foo(number x) {
    x <- x * 5;
    println("value of a in function: "); print(a);
    println("value of x: "); print(x);
    return 2;
}

number b <- foo(a);

println("a is unchanged: "); print(a);
println("return value of foo: "); print(b)

# prints the following:
# value of a in function: 7.000
# value of x: 35.000
# a is unchanged: 7.000
# return value of foo: 2.000
```
Interlude Math Demos
Architecture

Scanner/Parser/AST:
- Scanner reads in source files and tokenizes them.
- Parser processes tokens into abstract syntax tree.
- Abstract syntax tree represents Macaw program

Semantic Checker (aka Evaluator):
- Receives AST and checks validity of semantics and syntax
  - Declarations, Types
- Create structure for the list of statements and functions.

SAST:
- Result of the semantic transformation of the AST
- Passed to codegen for code emission

Codegen (aka Compilator):
- Takes SAST and emits LLVM code.
- No logic or decision-making (except resolving data types); mechanically translates SAST to LLVM IR.
Testing Process

- Language reference manual used to devise test cases and scenarios.
  - Both success scenarios and expected failure scenarios
  - Write unit tests that **should** pass/fail.
- System architects implemented features, wrote test programs.
- Testers broke down test programs into component unit tests.
- “Test all” script implemented to run regressions.
Lessons Learned -- Our most important takeaways

- Chris: Complex project, project management, testing
- Yi: Learned about the language design process, testing to break the language
- William: Matrix time management, planning language architecture
- Joseph: TDD, Semantic checking/transforming is surprisingly powerful
Live Demo -- The coolest things we can do
Questions?