...well there’s a language called Go...

James Stenger	Jillian Knoll	Lusa Zhan	Jonathan Barrios
System Architect	Language Guru	Project Manager	Tester

Goals

Both functional and object oriented:

- We want to create a functional programming language where users also have access to object-oriented style structures.

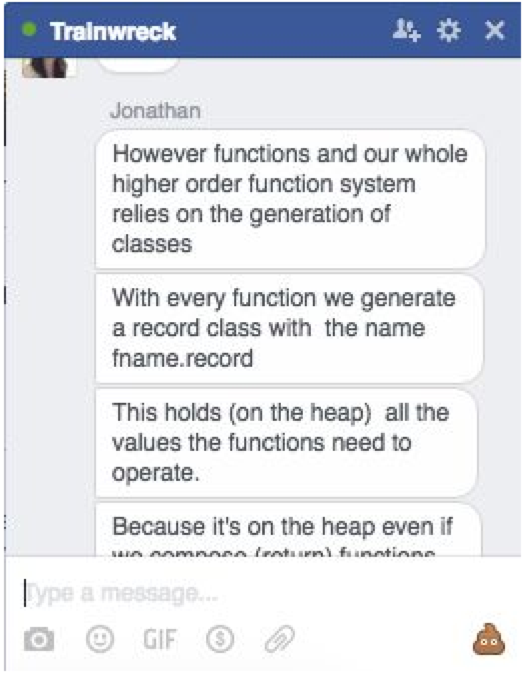
Compiling to LLVM:

- Compiling to LLVM allows for cross-language integrations that would allow a user to combine the functionality of Stop with a library from C.

Software & Frameworks

- Virtual Machine: Ubuntu VM allowed for PC-independent code generation
- Bitbucket: Used to create a private repository and track errors. We spent a lot of time programming in pairs for major architectural designs. We fixed bugs by raising issues after group work sessions.
- Ocaml Core library: Preferred to standard library due to named parameters

We constantly communicated



Jonathan

However functions and our whole higher order function system relies on the generation of classes

With every function we generate a record class with the name fname.record

This holds (on the heap) all the values the functions need to operate.

Because it's on the heap even if we compose (return) functions

Syntax & Program Structure

Comments

```
/* This is a comment */
// So is this
```

Operators

```
+ - ++ -- * / % = == !=
< > <= >= && || !
```

Variables

```
var a:Int = 1;/* capital
letter for types */
```

If/Else If/Else

```
if (a > 2){
     printf(“%d”,a);
} else if(a < 2){
     printf(“%d”,a);
} else {
     printf(“a is
2.”);
}
```

Arrays

```
var arr:Int[] = Int[2]();
```

For, While

```
var a:Int;

for (i=0; i<5; i++){
     printf(“%d”,i);
}

while (i<10){
     printf(“%d”,i);
     i++;
}
```

Syntax & Program Structure

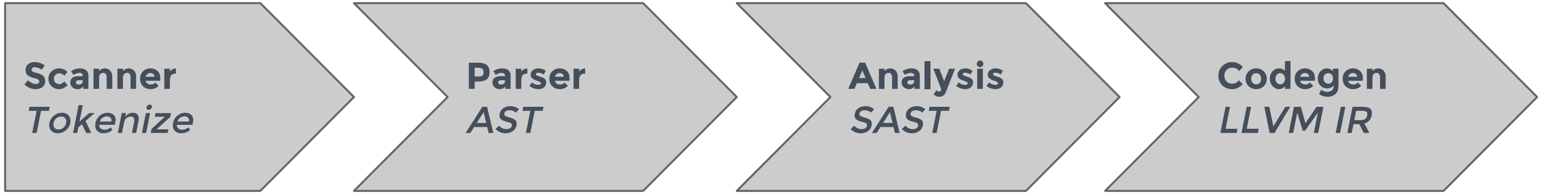
Functions

```
//main function with no arguments
def main = ():Int { return 0;}

//function takes single integer argument
def square=(a:Int):Int {return a*a;}

//anonymous function
var a = @(b:Int):Int { return b*b;}

//fn that returns an anonymous function
def outer = (a:Int):Int->Int{
     var inner = @(b:Int):Int{
          return b*b; }
return inner; }
```

Classes

```
//classes are user declared data types
class Rectangle = {
    var j:Int;
    var y:Int;
    var i:Int[];}

//instantiate class instance and
declare variables
def main = ():Int {
    var q:Rectangle;
    q.y = 9;
    q.j = 10;
    q.i = Int[5]();
    q.i[4]=9; }
```

System Architecture



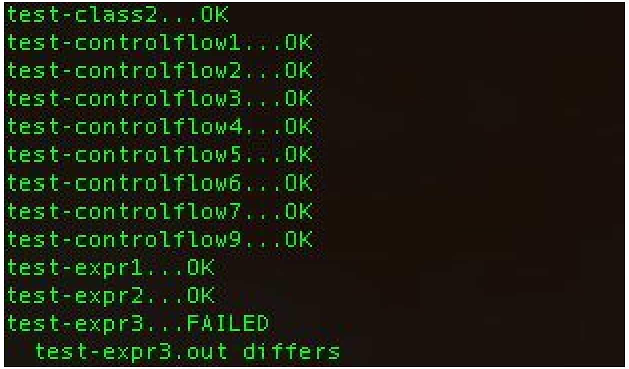
Scanner
Tokenize

Parser
AST

Analysis
SAST

Codegen
LLVM IR

Testing



```
test-class2...OK
test-controlflow1...OK
test-controlflow2...OK
test-controlflow3...OK
test-controlflow4...OK
test-controlflow5...OK
test-controlflow6...OK
test-controlflow7...OK
test-controlflow9...OK
test-expr1...OK
test-expr2...OK
test-expr3...FAILED
  test-expr3.out differs
```

- Automated Test Suite

The first thing we did was create an automated test suite, partially borrowed from MicroC. The regression test suite is executed by the ./testall script.

- Test Driven Development

We followed TDD by first writing tests for new features followed by the feature implementation.

Demo