StockQ

Ricky Martinez
Jesse Van Marter

Goals

- Create a lightweight, mathematical centered imperative language where users can quickly write and execute programs with low barriers. We wanted the syntax to be intuitive and easy to learn, type interactions with ints and floats to be simple, and recursiveness allowed

- Compile to LLVM for speed and portability. Also allows for importation and linking of C code

Software and Frameworks

- Ubuntu 16.04 VM to control for different testing environments
- Github and Git for version control and accessibility
- Opam to install OCaml packages, as well as compile OCaml

Syntax & Program Structure

Comments

/* … */

//

Operators

+ - * / % += -= *= /= %=

< <= > >= == !=

and or not

If/Else

```
if (a < 2) {

} else if (a>2) {

} else {

}
```

For, While

```
int x;

for (x=0; x < 10; x+=1) {
  print(x);
}

while (x < 20) {
  print(x); x += 1;
}
```

Syntax and Program Structure

Arrays

```
int[] a = int[10];

a[0] = 5;

print(a[0]);

print(a[1]);
```

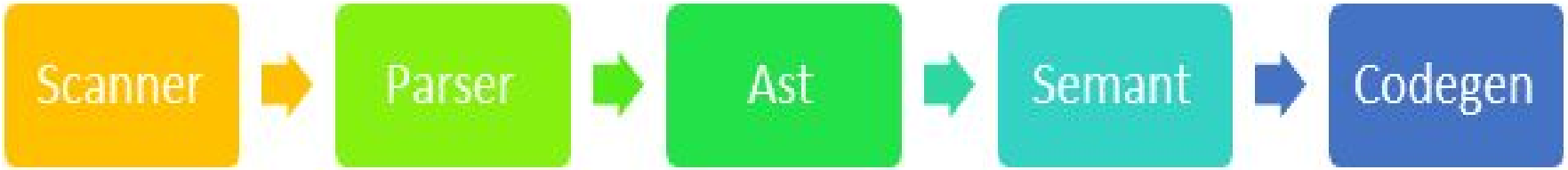
Functions

```
def int myfunc (int a, float f) {
  print(f);
  print(f + a);
  return 6;
}

// functions can be called within one

// another
```

Compiler Architecture



Test Suite

- Automated Test Suite (similar to MicroC)

- Test Driven Development

Test Suite

```
Running scanner tests...
  - checking scanner/_access.in...                    SUCCESS
  - checking scanner/_branch_control.in...            SUCCESS
  - checking scanner/_comment.in...                   SUCCESS
  - checking scanner/_datatypes.in...                 SUCCESS
  - checking scanner/_float.in...                     SUCCESS
  - checking scanner/_func_call.in...                 SUCCESS
  - checking scanner/_literals.in...                  SUCCESS
  - checking scanner/_logic.in...                     SUCCESS
  - checking scanner/_operators.in...                 SUCCESS
  - checking scanner/_symbols.in...                   SUCCESS
Running parser tests...
  - checking parser/_arithmetic.in...                 SUCCESS
  - checking parser/_call.in...                       SUCCESS
  - checking parser/_for.in...                        SUCCESS
  - checking parser/_func_def.in...                   SUCCESS
  - checking parser/_if_else.in...                    SUCCESS
  - checking parser/_literal.in...                    SUCCESS
  - checking parser/_relational.in...                 SUCCESS
  - checking parser/_while.in...                      SUCCESS
```

```
Running compiler tests...
  - checking compiler/test-add1.sq...                 SUCCESS
  - checking compiler/test-add2.sq...                 SUCCESS
  - checking compiler/test-arith1.sq...               SUCCESS
  - checking compiler/test-arith2.sq...               SUCCESS
  - checking compiler/test-arith3.sq...               SUCCESS
  - checking compiler/test-fib.sq...                  SUCCESS
  - checking compiler/test-for1.sq...                 SUCCESS
  - checking compiler/test-for2.sq...                 SUCCESS
  - checking compiler/test-func1.sq...                SUCCESS
  - checking compiler/test-func2.sq...                SUCCESS
  - checking compiler/test-func3.sq...                SUCCESS
  - checking compiler/test-func4.sq...                SUCCESS
  - checking compiler/test-func5.sq...                SUCCESS
  - checking compiler/test-func8.sq...                SUCCESS
  - checking compiler/test-gcd1.sq...                 SUCCESS
  - checking compiler/test-hello.sq...                SUCCESS
  - checking compiler/test-if1.sq...                  SUCCESS
  - checking compiler/test-if2.sq...                  SUCCESS
  - checking compiler/test-if3.sq...                  SUCCESS
  - checking compiler/test-if4.sq...                  SUCCESS
  - checking compiler/test-if5.sq...                  SUCCESS
  - checking compiler/test-local1.sq...               SUCCESS
  - checking compiler/test-local2.sq...               SUCCESS
  - checking compiler/test-ops1.sq...                 SUCCESS
  - checking compiler/test-ops2.sq...                 SUCCESS
```

Function Examples

```
for( x = 0; x < 10; x += 1 ) {
    a[x] = factorial(x);
    print(a[x]);
}

def int fibonacci (int x) {
    if ( x <= 1 ) {
        return 1;
    } else {
        return fibonacci( x - 1 ) + fibonacci ( x - 2 );
    }
}
```

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

print( gcd(14, 21) );
print( gcd(8, 36) );
print( gcd(99, 121) );
```