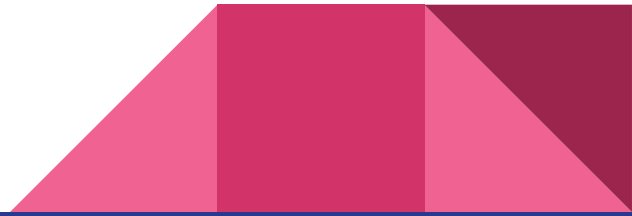
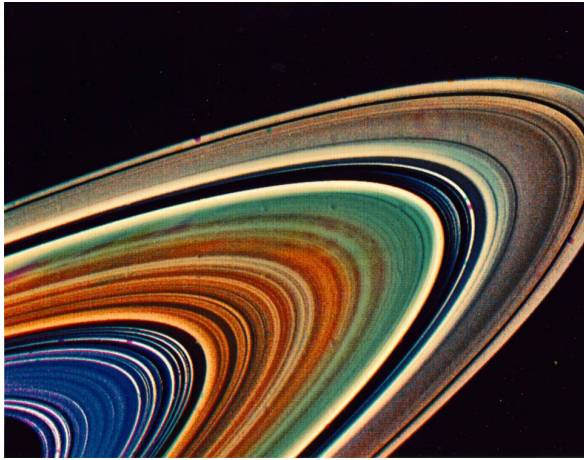
FRAC: Recursive Art Compiler

Annie Zhang, Calvin Li, Justin Chiang, Kunal Kamath

Motivation

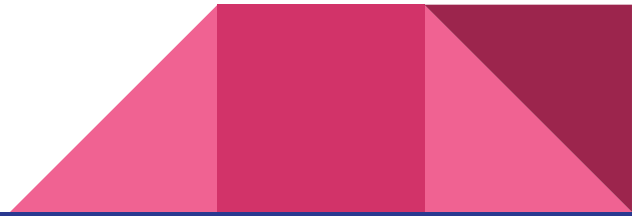
- Why fractals?
- Fractals are fascinating geometric objects that reflect natural patterns
  - Snowflakes
  - Pineapples
  - Saturn’s rings
- What if there was an easier way to visualize them?
- Better yet, what if we could visualize fractal generation in motion?





The FRAC Language

- Imperative, statically typed
- Primary feature: uses L-systems to generate fractals
- Grammar declarations consist of:
  - Alphabet
  - Init string
  - Rules (recursive and/or terminal)
- and are used to generate fractals when they are passed into system function calls
  - draw(gram g, int n)
  - grow(gram g, int n)



GCD

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

```
main() {
    int n = gcd(8, 12);
    print(n);
}
```

Koch Snowflake

```
gram koch = {
    alphabet: [F, p, m],
    init: 'F p p F p p F',
    rules: {
        'F' -> 'F m F p p F m F',
        'F' -> move(1),
        'm' -> rturn(60),
        'p' -> lturn(60)
    }
}
```

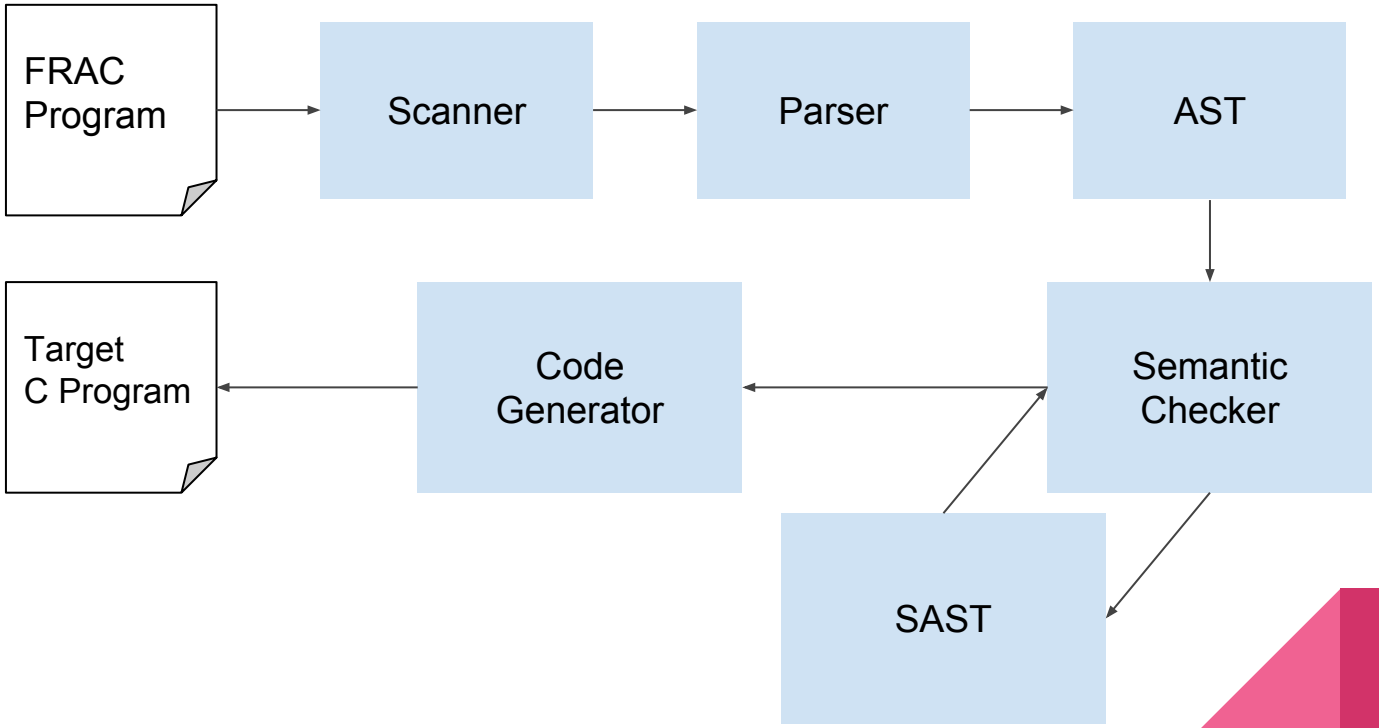
For a static BMP image:

```
main() {
    draw(koch, 6);
}
```

For a growing GIF:

```
main() {
    grow(koch, 6);
}
```

Compiler Structure



Turtle Graphics in C

- In turtle graphics, a “turtle” is given commands to move around a grid, drawing out its path
- Commonly associated with the Logo programming language
- An adaptation for C was obtained from https://w3.cs.jmu.edu/lam2mo/cs240_2015_08/turtle.html,
- Recursive rules are evaluated to turtle functions to draw fractals

Init string

```
void koch_start(int iter) {
    koch('F', iter);
    koch('p', iter);
    koch('p', iter);
    koch('F', iter);
    koch('p', iter);
    koch('p', iter);
    koch('F', iter);
}
```

Rules

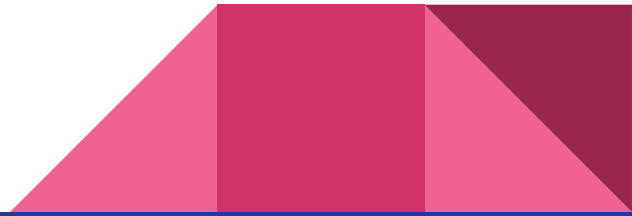
```
void koch(char var, int iter) {
    if (iter < 0) {
        if (var == 'F') {
            turtle_forward(1);
        }
    } else {
        if(var == 'F') {
            koch('F', iter - 1);
            koch('m', iter - 1);
            koch('F', iter - 1);
            koch('p', iter - 1);
            koch('p', iter - 1);
            koch('F', iter - 1);
            koch('m', iter - 1);
            koch('F', iter - 1);
        }
        if (var == 'm') {
            turtle_turn_right(60);
        }
        if (var == 'p') {
            turtle_turn_left(60);
        }
    }
}
```

Draw function

```
int main(){
    turtle_init(2000, 2000);
    koch_start(6);
    turtle_save_bmp("koch.bmp");
    turtle_cleanup();
    return 0;
}
```

Testing

- Regression test suite
  - Checks programs that should pass
  - Confirms programs that should fail
- Tests every aspect of our language, from expressions to program structure
- Runs compiled C code and compares result with the expected output



Lessons Learned

- Do work incrementally
- Come up with more concrete goals for ourselves
- Even basic semantic checking can be tricky!
- Pair programming is the way to go
- Git/Github is your best friend

Demos!!

- Koch Snowflake
  - Static image
- Sierpinski Triangle
  - Static image
- Heighway Dragon
  - Growing image