Objects V

Digression: Algorithms

- CS is just as much about inventing and implementing your own algorithms as it is implementing other people's.
    - Examples: mesostic project, reduce()
- Just because an algorithm ***works*** doesn't mean it's the best way to solve the problem.
- Let's investigate a number of different ways to implement reduce().

Reducing a fraction to lowest terms

- Basic idea (same as elementary school):
- Find the largest integer that goes evenly into the numerator and denominator.
- Divide both the numerator and denominator by this integer (called the ***greatest common divisor***, or GCD).

4/16 => 1/4

18/30 => 3/5

100/25 => 4/1

3/7 => 3/7

Reducing a fraction to lowest terms

- **Think** before you code:
- How do I test if a potential divisor goes evenly into the numerator and denominator?
- What is the smallest possible GCD of two integers?
- What is the largest possible GCD?
- I need a loop to examine all the possible divisors between the largest and the smallest. In what order should I examine them?

4/16 => 1/4

18/30 => 3/5

100/25 => 4/1

3/7 => 3/7

```
void rational::reduce()
{
  int start = min(numer, denom);
  int gcd;

  for (int divisor = start; divisor >= 1; divisor--)
  {
    if ((numer % divisor == 0) && (denom % divisor == 0))
    {
        gcd = divisor;
        break;
    }
  }

  numer = numer / gcd;
  denom = denom / gcd;
}
```

Is this the best algorithm for GCD?

- No, it’s not.
- The Euclidean algorithm is faster, but is harder to understand why it works.

```
int gcd(int a, int b) {
  while (b != 0) {
    int t = b;
    b = a % b;
    a = t;
  }
  return a;  // a is the GCD of a and b.
}
```

- This is one of the oldest algorithms we know (375 BC or older).

Back to classes…

- Should this reduce() method be public or private? What are the pros and cons of each way?

- Underlying question: should we let the user go around using unreduced fractions? Or should we let the user assume that whenever they use our class, the fractions will always be in lowest terms?
    - Common question when designing classes: Do we take some control away from the user in order to simplify the way they use the class?

“Everything should be made as simple as possible, but not simpler.”

--- Albert Einstein
(attributed)

Back to classes…

- Question – do we want to support fractions not in lowest terms?
- If **yes**, then we make the method public and have the user call reduce() when they want to.
- If **no**, then we make the method private and we will call reduce() when appropriate to ensure the user never sees a fraction not in lowest terms.

- Let’s look at one way of doing this...

Lab time!