From Mathematics to Generic Programming

Course Slides – Part 1 of 3

Version 1.0

October 5, 2015

Lecture 1

Overview of the Course

(Covers material from Chapter 1)

What This Course Is About

The deep relationship between mathematics and programming, especially generic programming.

What Do We Mean By Generic Programming?

Generic Programming is an

- approach to programming…
- focused on designing algorithms and data structures so that they work in the most general setting…
- without loss of efficiency

Wait, what?

- What about things like templates, iterator traits, and metaprogramming?
  - Constructs like templates are tools needed for today's languages to support generic programming
  - You need to know how to use them, but they are just means to an end
- Generic programming is more of an *attitude* than a particular set of tools

Origins of Generic Programming Approach

Abstract Algebra: Branch of mathematics concerned with reasoning about entities in terms of abstract properties of operations on them

- This course will introduce you to some of the basic ideas of abstract algebra
- Then we'll see how to apply these ideas to programming

What Are We Abstracting?

- When you're trying to find the most general way to express an algorithm, or a mathematical idea, you need to start with a concrete problem and concrete examples
- In mathematics, the concrete problems that drove abstract algebra come from *number theory*.

Number Theory

Number Theory: Branch of mathematics that deals with properties of integers, especially divisibility.

- In this course, we'll introduce some classic problems and results in number theory.
- Later we'll see how some of these results apply to a modern, practical problem.

Approach

This course will weave together mathematics and programming

- Some parts will be just about math
- Some parts will be just about programming
- Some parts will be about both

Programming Prerequisites

- You should be comfortable reading and writing code in an imperative language like C, C++, or Java
- Our examples use C++, but the basic ideas of generic programming are not language-specific

Mathematical Prerequisites

- No specific knowledge beyond high-school algebra

But:

- You need to be able to think logically
- It will help if you have some experience with mathematical proofs

Course in a Nutshell

- *Understanding the principles of generic programming will make you a better programmer*
- *To understand the principles of generic programming, you need to understand abstraction*
- *To understand abstraction, you need to understand the mathematics on which it's based*

Textbook

- This course closely follows the book *From Mathematics to Generic Programming* by Alexander A. Stepanov and Daniel E. Rose (Addison-Wesley, 2015).
- For more about the book, including code for examples, see www.fm2gp.com.

Lecture 2

Egyptian Multiplication

(Covers material from Chapter 2)

Algorithms

- What is an algorithm?
- What's the first algorithm you learned?
- What's the first algorithm *anyone* learned?
- Where did it come from?
  - – Possibly ancient Egypt…

Ancient Egyptian Mathematics: Rhind Mathematical Papyrus

- Written by the scribe Ahmes
- 1650BC (copy of a much older text ≈1850BC)
- Algorithms for fast multiplication and division

What is multiplication?

- Informally:
  "Adding something to itself a certain number of times."
- Formally:

$$1a = a$$
$$(n + 1)a = na + a$$

- In code:

```
int multiply0(int n, int a) {
   if (n == 1) return a;
   return multiply0(n - 1, a) + a;
}
```

Is there a faster way? Ancient Egyptians had one…

The algorithmic insight:

$$\begin{aligned} 4a &= ((a + a) + a) + a \\ &= (a + a) + (a + a) \end{aligned}$$

- Relies on associativity
- Only need to compute $a + a$ once

Example of Ahmes Algorithm: 41 × 59

Make table where each line doubles line before

1	✓	59
2		118
4		236
8	✓	472
16		944
32	✓	1888

- Mark entries in first column that sum to first factor

  41 = 1 + 8 + 32
- Add corresponding entries in second column

  59 + 472 + 1888
- Result is desired product

i.e. 41 × 59 = (1 × 59) + (8 × 59) + (32 × 59)

Requires knowledge of *even* and *odd*

- The Egyptians must have known this distinction, because the technique relies on it
- Here's how the Ancient Greeks defined them:

$$n = \frac{n}{2} + \frac{n}{2} \implies \text{even}(n)$$

$$n = \frac{n-1}{2} + \frac{n-1}{2} + 1 \implies \text{odd}(n)$$

Halving and Doubling

- Also need to know that:

$$\mathrm{odd}(n) \implies \mathrm{half}(n) = \mathrm{half}(n-1)$$

- In code:

```
bool odd(int n) { return n & 0x1; }
int half(int n) { return n >> 1; }
```

Egyptian Multiplication Algorithm

```
int multiply1(int n, int a) {
    if (n == 1) return a;
    int result =
      multiply1(half(n), a + a);
    if (odd(n)) result = result + a;
    return result;
}
```

…aka *Russian Peasant* Algorithm

- Many computer scientists know this as the *Russian Peasant* Algorithm, which is what Knuth called it.
- But original source actually says:

“I have been told that there is a method in use to-day (some say in Russia, but *I have not been able to verify this*)...”

Sir Thomas Heath, *History of Greek Mathematics,* 1911

How many additions to multiply by n?

$$\#_+(n) = \lfloor \log n \rfloor + \nu(n) - 1$$

where $\nu(n)$ is the number of 1s in the binary representation of n (the population count or *pop count*)

We have replaced an O(n) algorithm with a O(log n) algorithm.

Is it optimal?

$$\#_{+}(15) = 3 + 4 - 1 = 6$$

multiply_by_15

```
int multiply_by_15(int a) {
    int b = (a + a) + a;    // b == 3*a
    int c = b + b;          // c == 6*a
    return (c + c) + b;     // 12*a + 3*a
}
```

- 5 additions!
- We have discovered an optimal *addition chain* for 15.
- However, we'll stick with Egyptian Multiplication for the general case.

Can we improve the Egyptian Multiplication code?

- The code of `multiply1` is a good implementation as far as the number of additions
- But it also does $\lfloor \log n \rfloor$ recursive calls
  - function call is much more expensive than *plus*

Another approach: use helper function **multiply-accumulate**

$$r + na$$

a and n are the values we're multiplying;
r is a running total.

Observation

It is often easier to do more rather than less.

Multiply-Accumulate: First Attempt

```
int mult_acc0(int r, int n, int a) {
  if (n == 1) return r + a;
  if (odd(n))
    return mult_acc0(r + a, half(n), a + a);
  } else {
    return mult_acc0(r, half(n), a + a);
  }
}
```

Invariant: $r + na = r_0 + n_0 a_0$

Tail Recursion

- A function is *tail recursive* if it calls itself in its return statement (only).

Tail Recursive Multiply-Accumulate

```
int mult_acc1(int r, int n, int a) {
    if (n == 1) return r + a;
    if (odd(n)) r = r + a;
    return mult_acc1(r, half(n), a + a);
}
```

Observations:

- n is usually not 1.
- if n is even, it is not 1.

More Efficient Multiply-Accumulate

We reduce number of comparisons with 1 by a factor of 2 by checking for oddness first:

```
int mult_acc2(int r, int n, int a) {
    if (odd(n)) {
        r = r + a;
        if (n == 1) return r;
    }
    return mult_acc2(r, half(n), a + a);
}
```

Strict Tail Recursion

- A function is *strictly tail recursive* if all the tail-recursive calls are done with the formal parameters of the procedure being the corresponding arguments.

Strictly Tail Recursive Multiply-Accumulate

```
int mult_acc3(int r, int n, int a) {
    if (odd(n)) {
        r = r + a;
        if (n == 1) return r;
    }
    n = half(n);
    a = a + a;
    return mult_acc3(r, n, a);
}
```

Iterative Multiply-Accumulate

```
int mult_acc4(int r, int n, int a) {
  while (true) {
    if (odd(n)) {
      r = r + a;
      if (n == 1) return r;
    }
    n = half(n);
    a = a + a;
  }
}
```

Multiply Using Multiply-Accumulate

```
int multiply2(int n, int a) {
    if (n == 1) return a;
    return mult_acc4(a, n - 1, a);
}
```

Observations:

- When n is 16, it does 7 additions instead of 4.
- We do not want to subtract 1 from an even number.
  Let us make it odd!

Improve Multiply

```
int multiply3(int n, int a) {
    while (!odd(n)) {
        a = a + a;
        n = half(n);
    }
    if (n == 1) return a;
    return mult_acc4(a, n - 1, a);
}
```

Final Version of Multiply

```
int multiply4(int n, int a) {
  while (!odd(n)) {
      a = a + a
      n = half(n);
  }
  if (n == 1) return a;
  // even(n - 1) => n - 1 != 1
  return mult_acc4(a, half(n - 1), a + a);
}
```

Things to Consider

- Even a simple algorithm can benefit from rewriting.
- Even a small optimization may have a large effect:
  - It may be called billions of times
  - A future use of the algorithm may replace your inexpensive operation with a very expensive one

Lecture 3

The Sieve of Eratosthenes

(Covers material from Sec. 3.1-3.3)

Pythagoras (570BC - 490BC)

- Studied with Thales of Miletus, the founder of philosophy
- Traveled to Egypt to learn mathematics, and then to Babylon
- Founded ascetic colony to study

Pythagorean Studies

- “Quadrivium” – original basis of European education:
  - Astronomy
  - Geometry
  - Number Theory
  - Music

Integration of Pythagorean Studies

- Each field related to the others
  - E.g. numerical relationships between frequencies in music
- Pythagoras could “hear the music of the celestial spheres”

Pythagorean Mathematics

Pythagoreans applied themselves to the study of mathematics. They thought that its principles must be the principles of all existing things.

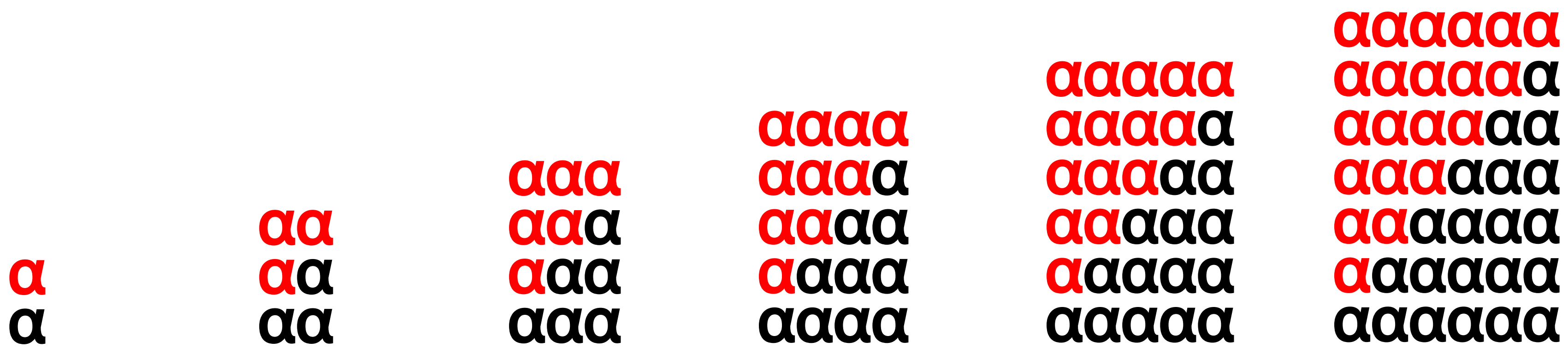
Aristotle, *Metaphysics*

Triangular Numbers: $\triangle_n$

```
                                   α
                          α       αα
                  α      αα      ααα
           α     αα     ααα     αααα
     α    αα    ααα    αααα    ααααα
α   αα   ααα   αααα   ααααα   αααααα
```

1 3 6 10 15 21

Oblong Numbers: $\square_n$

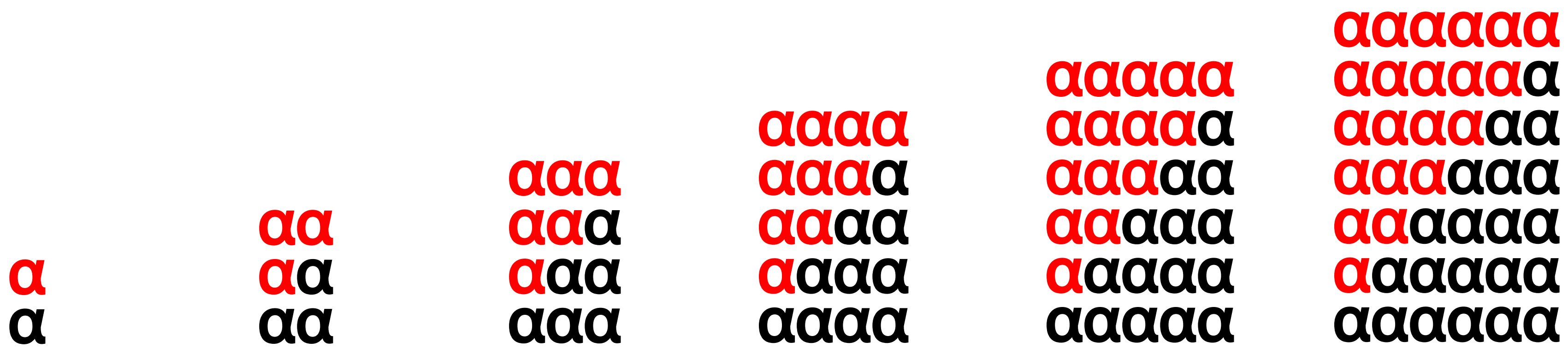


2 6 12 20 30 42

*n*th oblong number is area of the rectangle:

$$\square_n = n(n+1)$$

Relationship of Triangular and Oblong

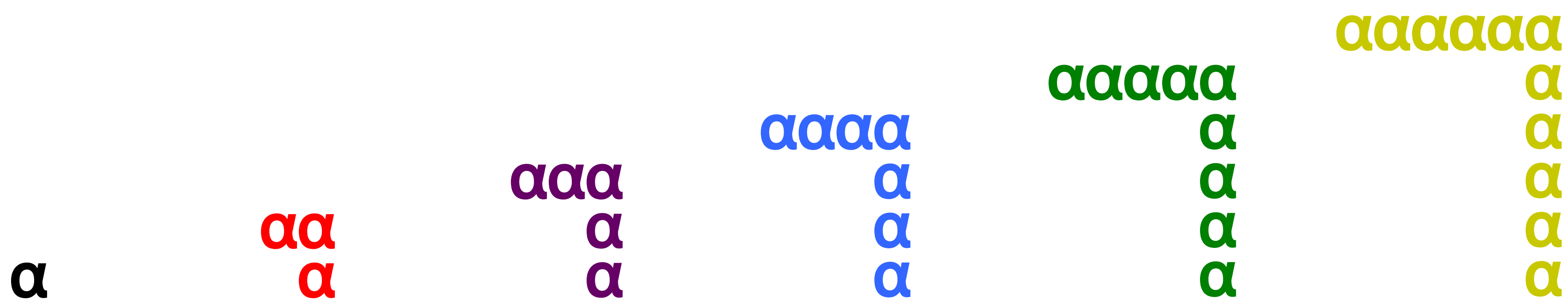


2 6 12 20 30 42

Triangle is half the rectangle, so sum of first *n* pos. integers is:

$$\triangle_n = \sum_{i=1}^{n} i = \frac{\square_n}{2} = \frac{n(n+1)}{2}$$

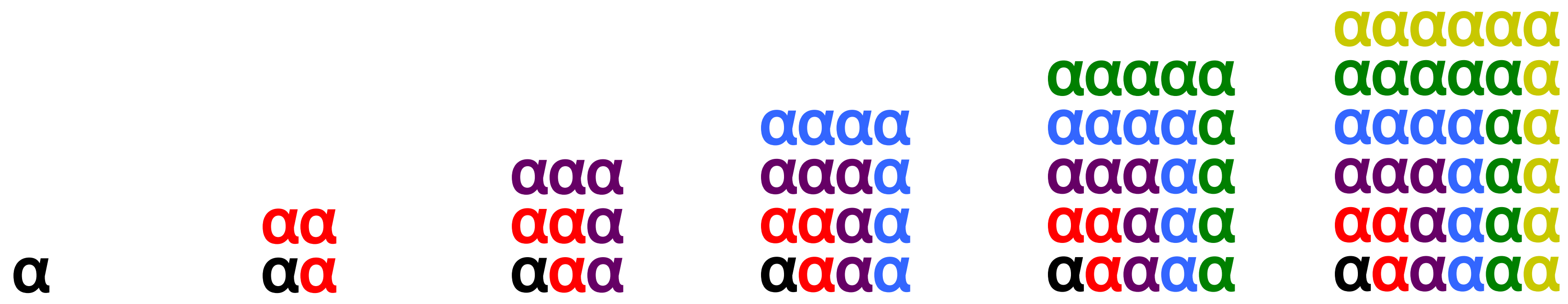
Gnomons



1 3 5 7 9 11

The *n*th gnomon is the *n*th odd number.

Combining gnomons gives us square numbers



1 4 9 16 25 36

And a formula for the sum of the first *n* positive odd integers:

$$\square_n = \sum_{i=1}^{n} (2i - 1) = n^2$$

Prime Numbers

Numbers that are not products of smaller numbers.

2, 3, 5, 7, 11, 13, 17…

Euclid VII, 32

Any number is either prime or divisible by some prime.

If not, “an infinite sequence of numbers will divide the number, each of which is less than the other; and this is impossible.”

Euclid IX, 20:

For any sequence of primes $\{p_1, \ldots, p_n\}$ *there is a prime* p *not in the sequence.*

Proof: Consider the number $q = 1 + \prod_{i=1}^{n} p_i$

By construction, q is not divisible by any p_i.

Either q is prime
    in which case it is a prime not in the sequence

or q is divisible by some new prime,
    which is by definition not in the sequence.

Therefore, there are infinitely many primes.

Eratosthenes (284BC – 195BC)

- Born in Cyrene, Libya
- Studied in Athens (possibly with Zeno)
- Tutored Ptolemy Euergetes (king of Egypt)
- Measured circumference of the Earth (within 2%!)
- A friend of Archimedes

Sieve of Eratosthenes

- Method for finding primes
- General Idea:
  - List all the integers starting with the first prime
  - Cross out multiples (they "fall through" the sieve")
  - Whatever is left is prime
- But don't waste time on even numbers (so actually start with 3)

Sieve of Eratosthenes Example

Find primes up to maximum *m* = 53

Start with all the candidates:

3	5	7	9	11	13	15	17	19	21	23	25	27
29	31	33	35	37	39	41	43	45	47	49	51	53

Sieve of Eratosthenes Example

$\textcircled{3}$	5	7	$\boxed{\cancel{9}}$	11	13	$\boxed{\cancel{15}}$	17	19	$\boxed{\cancel{21}}$	23	25	$\boxed{\cancel{27}}$
29	31	$\boxed{\cancel{33}}$	35	37	$\boxed{\cancel{39}}$	41	43	$\boxed{\cancel{45}}$	47	49	$\boxed{\cancel{51}}$	53

Sieve of Eratosthenes Example

3	$\textcircled{5}$	7	$\cancel{9}$	11	13	$\cancel{15}$	17	19	$\cancel{21}$	23	$\boxed{\cancel{25}}$	$\cancel{27}$
29	31	$\cancel{33}$	$\boxed{\cancel{35}}$	37	$\cancel{39}$	41	43	$\boxed{\cancel{45}}$	47	49	$\cancel{51}$	53

Sieve of Eratosthenes Example

3	5	$\textcircled{7}$	$\cancel{9}$	11	13	$\cancel{15}$	17	19	$\cancel{21}$	23	$\cancel{25}$	$\cancel{27}$
29	31	$\cancel{33}$	$\cancel{35}$	37	$\cancel{39}$	41	43	$\cancel{45}$	47	$\boxed{\cancel{49}}$	$\cancel{51}$	53

Sieve of Eratosthenes Example

Since we've crossed out all multiples up to $\lfloor \sqrt{m} \rfloor$ we're done – remaining numbers are prime.

3	**5**	**7**	~~**9**~~	**11**	**13**	~~**15**~~	**17**	**19**	~~**21**~~	**23**	~~**25**~~	~~**27**~~
29	**31**	~~**33**~~	~~**35**~~	**37**	~~**39**~~	**41**	**43**	~~**45**~~	**47**	~~**49**~~	~~**51**~~	**53**

A look back at the process

Let's keep an index array alongside the values:

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17…
values:	3	5	7	~~9~~	11	13	~~15~~	17	19	~~21~~	23	$\boxed{\cancel{25}}$	~~27~~	29	31	~~33~~	$\boxed{\cancel{35}}$	37…

Observations:

- When crossing out multiples of n, "step size" is n
- Difference between two values is twice the difference between two indices
- First number crossed out for a factor n is n^2

Implementation Data Structure

- Naive approach: store array of values and array of flags indicating whether corresponding value is prime.
- But values can be trivially computed from index: *i*th value is 3 + 2*i*.
- So, we only need the array of flags.

Digression: Concepts and Templates

- *Concepts* are requirements on types.
- We use them instead of `typename` to say what types our template arguments should be:

```
template <Iterator I>
```

- Since C++ doesn't know about concepts (yet), we'll fake it:

```
#define Iterator typename
```

End of Digression

Marking all multiples of a given factor

```
template <RandomAccessIterator I, Integer N>
void mark_sieve(I first, I last, N factor)
{
    // assert(first != last)
    *first = false;
    while (last - first > factor) {
        first = first + factor;
        *first = false;
    }
}
```

Sifting Lemmas

- The square of the smallest prime factor of a composite number c is less or equal than c.
- Any composite number less than p^2 is sifted by (i.e. crossed out as a multiple of) a prime less than p.
- When sifting by p, start marking at p^2.
- If we want to sift numbers up to m, stop sifting when $p^2 \geq m$.

Sifting Formulas (1)

$$\text{value at index } i: \ \text{value}(i) = 3 + 2i = 2i + 3$$

$$\text{index of value } v: \ \text{index}(v) = \frac{v-3}{2}$$

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17…
values:	3	$\bigcirc\!\!\!\!\!\,5$	7	$\cancel{9}$	11	13	$\cancel{15}$	17	19	$\cancel{21}$	23	$\boxed{\cancel{25}}$	$\cancel{27}$	29	31	$\cancel{33}$	$\boxed{\cancel{35}}$	37…

Sifting Formulas (2)

step between multiple k and multiple $k+2$ of value at i:

$$\begin{aligned}\text{step}(i) &= \text{index}((k+2)(2i+3)) - \text{index}(k(2i+3)) \\ &= \text{index}(2ki+3k+4i+6) - \text{index}(2ki+3k) \\ &= \frac{(2ki+3k+4i+6)-3}{2} - \frac{(2ki+3k)-3}{2} \\ &= \frac{4i+6}{2} = 2i+3\end{aligned}$$

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17…
values:	3	⑤	7	~~9~~	11	13	~~15~~	17	19	~~21~~	23	$\boxed{\cancel{25}}$	~~27~~	29	31	~~33~~	$\boxed{\cancel{35}}$	37…

Sifting Formulas (3)

index of square of value at *i*:

$$\begin{aligned}\text{index}(\text{value}(i)^2) &= \frac{(2i+3)^2-3}{2}\\ &= \frac{4i^2+12i+9-3}{2}\\ &= 2i^2+6i+3\end{aligned}$$

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17…
values:	3	$\bigcirc\!\!\!\!\!\;5$	7	$\cancel{9}$	11	13	$\cancel{15}$	17	19	$\cancel{21}$	23	$\boxed{\cancel{25}}$	$\cancel{27}$	29	31	$\cancel{33}$	$\boxed{\cancel{35}}$	37…

First Attempt at Sieve

```
template <RandomAccessIterator I, Integer N>
void sift0(I first, N n) {
    std::fill(first, first + n, true);
    N i(0);
    N index_square(3);
    while (index_square < n) {
        // invariant: index_square = 2i^2 + 6i + 3
        if (first[i]) {
            // if current candidate is prime
            mark_sieve(first + index_square,
                       first + n,    // last
                       i + i + 3);   // factor
        }
        ++i;
        index_square = 2*i*(i + 3) + 3;
    }
}
```

Hoist Common Subexpressions

```
template <RandomAccessIterator I, Integer N>
void sift1(I first, N n) {
  I last = first + n;
  std::fill(first, last, true);
  N i(0);
  N index_square(3);
  N factor(3);
  while (index_square < n) {
      // invariant: index_square = 2i^2 + 6i + 3, factor = 2i + 3
      if (first[i]) {
          mark_sieve(first + index_square, last, factor);
      }
      ++i;
      factor = i + i + 3;
      index_square = 2*i*(i + 3) + 3;
  }
}
```

Strength Reduction

Strength reduction is a *compiler optimization* where expensive operations are replaced with equivalent but less expensive operations.

Using Strength Reduction

Replace

```
   factor = i + i + 3;
   index_square =  3 + 2*i*(i + 3);
```

with

factor += δ_{factor};

index_square += $\delta_{\text{index\_square}}$;

where δ_{factor} and $\delta_{\text{index\_square}}$ are the differences between successive values of factor and index_square.

δ computations

$$\delta_{factor} : \quad (2(i+1)+3) - (2i+3) = 2$$

$$\begin{aligned}\delta_{index\_square} : \quad & (2(i+1)^2 + 6(i+1) + 3) - (2i^2 + 6i + 3) \\ & = 2i^2 + 4i + 2 + 6i + 6 + 3 - 2i^2 - 6i - 3 \\ & = 4i + 8 = (2i+3) + (2i+2+3) \\ & = (2i+3) + (2(i+1)+3) \\ & = \text{factor}(i) + \text{factor}(i+1)\end{aligned}$$

Final Version

```
template <RandomAccessIterator I, Integer N>
void sift(I first, N n) {
  I last = first + n;
  std::fill(first, last, true);
  N i(0);
  N index_square(3);
  N factor(3);
  while (index_square < n) {
      // invariant: index_square = 2i^2 + 6i + 3, factor = 2i + 3
      if (first[i]) {
          mark_sieve(first + index_square, last, factor);
      }
      ++i;
      index_square += factor;
      factor += N(2);
      index_square += factor;
  }
}
```

Things to Consider

- The ancient Greeks' fascination with seemingly superficial properties (e.g. "shapes" of numbers) led to the discovery of important mathematical concepts (e.g. primes) and the field of number theory
- Optimizing an algorithm is often literally a matter of elementary algebraic manipulation

Lecture 4

The Pythagorean Program

(Covers material from Sec. 3.4-3.7)

Perfect Numbers

A number is *perfect* if it is the sum of its proper divisors.

Example: $28 = 1 + 2 + 4 + 7 + 14$

Ancient Greeks Knew of Four Perfect Numbers

$$\begin{aligned} 6 &= 1+2+3 \\ 28 &= 1+2+4+7+14 \\ 496 &= 1+2+4+8+16+31+62+124+248 \\ 8128 &= 1+2+4+8+16+32+64+127+254+508+1016+2032+4064 \end{aligned}$$

Was there a predictable pattern?

- Greeks looked at the numbers' prime factorization:

$$\begin{aligned} 6 &= 2 \cdot 3 = 2^1 \cdot 3 \\ 28 &= 4 \cdot 7 = 2^2 \cdot 7 \\ 496 &= 16 \cdot 31 = 2^4 \cdot 31 \\ 8128 &= 64 \cdot 127 = 2^6 \cdot 127 \end{aligned}$$

Yes. Perfect when second term prime

$$
\begin{aligned}
6 &= 2 \cdot 3 = 2^1 \cdot (2^2 - 1) \\
28 &= 4 \cdot 7 = 2^2 \cdot (2^3 - 1) \\
120 &= 8 \cdot \mathbf{15} = 2^3 \cdot (2^4 - 1) \text{ not perfect} \\
496 &= 16 \cdot 31 = 2^4 \cdot (2^5 - 1) \\
2016 &= 32 \cdot \mathbf{63} = 2^5 \cdot (2^6 - 1) \text{ not perfect} \\
8128 &= 64 \cdot 127 = 2^6 \cdot (2^7 - 1)
\end{aligned}
$$

Can we prove this is always true?

Yes. Euclid did it around 300 BC

Elements Book IX, Proposition 36:

$$\textit{If } \sum_{i=0}^{n} 2^i \ \textit{ is prime then } 2^n \sum_{i=0}^{n} 2^i \ \textit{ is perfect.}$$

Difference of Powers

$$\begin{aligned}
x^2 - y^2 &= (x-y)(x+y) \\
x^3 - y^3 &= (x-y)(x^2+xy+y^2) \\
&\vdots \\
x^{n+1} - y^{n+1} &= (x-y)(x^n + x^{n-1}y + \cdots + xy^{n-1} + y^n)
\end{aligned}$$

Derivation:

$$\begin{aligned}
x(x^n + x^{n-1}y + \cdots + xy^{n-1} + y^n) &= x^{n+1} + x^n y + x^{n-1}y^2 + \cdots + xy^n \\
y(x^n + x^{n-1}y + \cdots + xy^{n-1} + y^n) &= \qquad\qquad x^n y + x^{n-1}y^2 + \cdots + xy^n + y^{n+1}
\end{aligned}$$

Sum of Odd Powers

$$x^{2n+1} + y^{2n+1} = (x+y)(x^{2n} - x^{2n-1}y + \cdots - xy^{2n-1} + y^{2n})$$

Derivation:
Convert sum to difference and use previous result.

$$\begin{aligned} x^{2n+1} + y^{2n+1} &= x^{2n+1} - -y^{2n+1} \\ &= x^{2n+1} - (-y)^{2n+1} \\ &= (x - (-y))(x^{2n} + x^{2n-1}(-y) + \cdots + (-y)^{2n}) \\ &= (x+y)(x^{2n} - x^{2n-1}y + \cdots - xy^{2n-1} + y^{2n}) \end{aligned}$$

Restating Euclid IX, 36

$$\sum_{i=0}^{n-1} 2^i = \qquad (2^{n-1} + 2^{n-2} + \cdots + 2 + 1)$$

$$= (2-1)(2^{n-1} + 2^{n-2} + \cdots + 2 + 1)$$

By difference of powers formula

$$(2-1)(2^{n-1} + 2^{n-2} + \cdots + 2 + 1) = 2^n - 1$$

So we will rewrite the theorem as:

If $2^n - 1$ *is prime, then* $2^{n-1}(2^n - 1)$ *is perfect.*

Sum of divisors of a number

Example: 24

- Prime factorization: $24 = 2^3 \cdot 3^1$
- So set of all divisors is:

  $\{2^0 3^0, 2^1 3^0, 2^2 3^0, 2^3 3^0, 2^0 3^1, 2^1 3^1, 2^2 3^1, 2^3 3^1\}$
  $= \{1, 2, 4, 8, 3, 6, 12, 24\}$
- Sum of all divisors is:

  $2^0 3^0 + 2^1 3^0 + 2^2 3^0 + 2^3 3^0 + 2^0 3^1 + 2^1 3^1 + 2^2 3^1 + 2^3 3^1$

  $= (2^0 + 2^1 + 2^2 + 2^3)(3^0 + 3^1)$
- So we can write sum of divisors as product of sums

Define $\sigma(n)$: sum of divisors of n

If the prime factorization of n is

$$n = p_1^{a_1} p_2^{a_2} \ldots p_m^{a_m}$$

Then the sum of the divisors of *n* is

$$\sigma(n) = \prod_{i=1}^{m} (1 + p_i + p_i^2 + \cdots + p_i^{a_i}) = \prod_{i=1}^{m} \frac{p_i - 1}{p_i - 1} (1 + p_i + p_i^2 + \cdots + p_i^{a_i})$$

$$= \prod_{i=1}^{m} \frac{(p_i - 1)(1 + p_i + p_i^2 + \cdots + p_i^{a_i})}{p_i - 1}$$

$$= \prod_{i=1}^{m} \frac{p_i^{a_i+1} - 1}{p_i - 1}$$

Aliquot Sum

- The *aliquot sum* of n is the sum of all the proper divisors of n:

$$\alpha(n) = \sigma(n) - n$$

Now we're ready to prove Euclid's theorem:

If $2^n - 1$ *is prime, then* $2^{n-1}(2^n - 1)$ *is perfect.*

Euclid IX, 36:

If $2^n - 1$ *is prime, then* $2^{n-1}(2^n - 1)$ *is perfect.*

Proof:

Let $q = 2^{n-1}(2^n - 1)$

We know 2 and (by assumption) $2^n - 1$ are prime, so q is a prime factorization of the form

$$n = p_1^{a_1} p_2^{a_2} \ldots p_m^{a_m}$$

where

$m = 2$, $p_1 = 2$, $a_1 = n - 1$, $p_2 = 2^n - 1$, and $a_2 = 1$.

So using our $\sigma(n)$ formula…

Proof of Euclid IX, 36 continued

$$\begin{aligned}
\sigma(q) &= \frac{2^{(n-1)+1}-1}{1} \cdot \frac{(2^n-1)^2-1}{(2^n-1)-1} \\
&= (2^n-1) \cdot \frac{(2^n-1)^2-1}{(2^n-1)-1} \cdot \frac{(2^n-1)+1}{(2^n-1)+1} \\
&= (2^n-1) \cdot \frac{((2^n-1)(2^n-1)-1)((2^n-1)+1)}{((2^n-1)(2^n-1)-1)} \\
&= (2^n-1)((2^n-1)+1) \\
&= 2^n(2^n-1) = 2 \cdot 2^{n-1}(2^n-1) = 2q
\end{aligned}$$

Then $\alpha(q) = \sigma(q) - q = 2q - q = q$
i.e. q is the sum of its proper divisors (perfect).

Is the converse true?

- Euclid’s theorem IX,36:

  “If a number has form $2^{n-1}(2^n - 1)$, then it is perfect.”
- Converse:

  “If a number is perfect, then it has form $2^{n-1}(2^n - 1)$.”

  – In the 18th century, Leonhard Euler proved this for even perfect numbers.

  – It has never been proved for all perfect numbers. (We don’t even know if odd perfect numbers exist!)

The Pythagorean Program: Unification of Mathematics

Mathematics as the science of the two fundamental perceptible aspects of reality: quantity (numbers) and space (geometry).

Can they be unified?

- How to ground geometry in numbers, i.e. positive integers?

Discretizing Space with Measure

- Segment *V* is a *measure* of segment *A* iff *A* can be represented as a finite concatenation of copies of *V*.
- Segment *V* is a *common measure* of segments *A* and *B* iff it is a measure of both.
- Segment *V* is the *greatest common measure (GCM)* of *A* and *B* if it is greater than any other common measure of *A* and *B*.

Pythagoreans Observed Properties of GCM

$$\begin{aligned} & \mathrm{gcm}(a,a) = a \\ & \mathrm{gcm}(a,b) = \mathrm{gcm}(a,a+b) \\ b < a \implies & \mathrm{gcm}(a,b) = \mathrm{gcm}(a-b,b) \\ & \mathrm{gcm}(a,b) = \mathrm{gcm}(b,a) \end{aligned}$$

Can use these properties to compute GCM…

Computing GCM

- Most important procedure from Greek mathematics.
- “Computations” are ruler and compass operations on line segments.

```
line_segment gcm(line_segment a,
                 line_segment b) {
    if (a == b)      return a;
    if (b < a)       return gcm(a - b, b);
 /* if (a < b) */    return gcm(a, b - a);
}
```

Example: GCM(196, 42)

$$
\begin{array}{llclcl}
\quad a \qquad b & & & & & \\
196 > 42, & \text{gcm}(196, 42) & = & \text{gcm}(196 - 42, 42) & = & \text{gcm}(154, 42) \\
154 > 42, & \text{gcm}(154, 42) & = & \text{gcm}(154 - 42, 42) & = & \text{gcm}(112, 42) \\
112 > 42, & \text{gcm}(112, 42) & = & \text{gcm}(112 - 42, 42) & = & \text{gcm}(70, 42) \\
70 > 42, & \text{gcm}(70, 42) & = & \text{gcm}(70 - 42, 42) & = & \text{gcm}(28, 42) \\
28 < 42, & \text{gcm}(28, 42) & = & \text{gcm}(28, 42 - 28) & = & \text{gcm}(28, 14) \\
28 > 14, & \text{gcm}(28, 14) & = & \text{gcm}(28 - 14, 14) & = & \text{gcm}(14, 14) \\
14 = 14, & \text{gcm}(14, 14) & = & 14 & &
\end{array}
$$

So GCM(196, 42) = 14

An Ancient Greek Proof Technique

- Well-ordering principle: the fact that any set of natural numbers has a smallest element

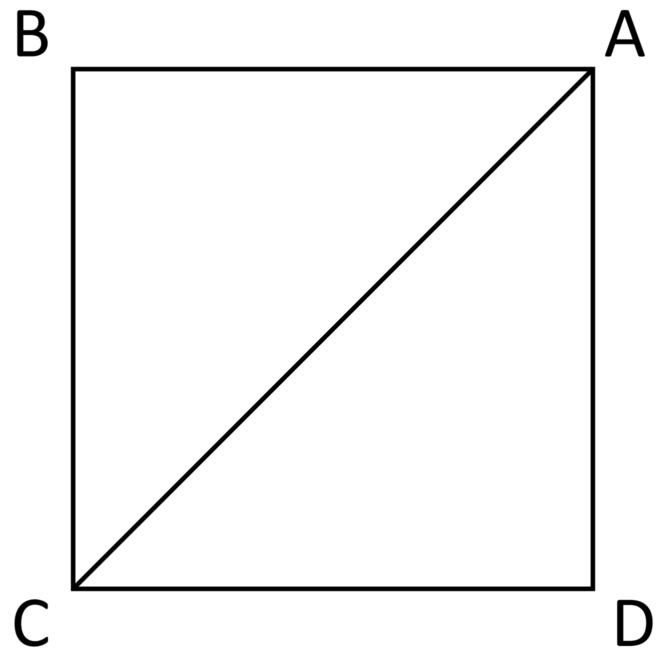
- Use in proofs:

  In order to prove that something does not exist, prove that if it exists, a smaller one also exists.

The Proof That Ruined the Pythagorean Plan (1)

There is no segment that can measure both the side and the diagonal of a square.

Proof: Assume contrary – that there is such a segment. Consider smallest square that segment can measure.

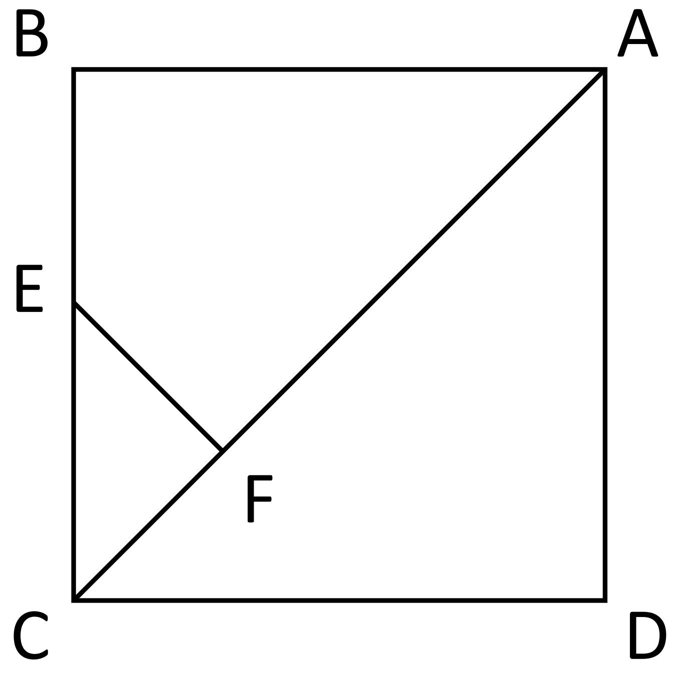


Proof (2)

- Using ruler and compass, construct segments

$$\overline{AF} = \overline{AB}$$

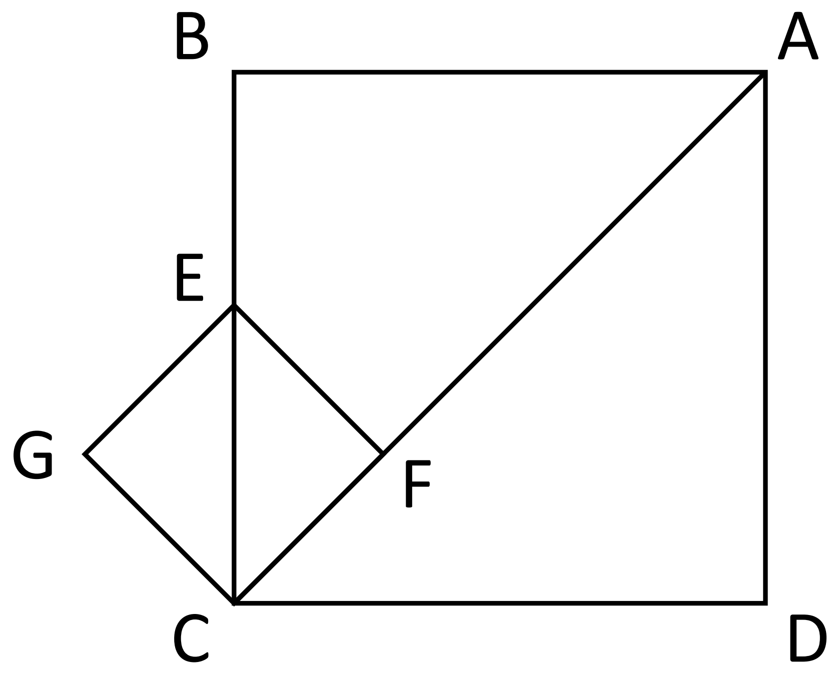
$$\overline{EF} \perp \overline{AC}$$



Proof (3)

- Construct two more perpendicular segments:

$$\overline{CG} \perp \overline{AC}$$
$$\overline{EG} \perp \overline{EF}$$



Proof (4)

- We know that

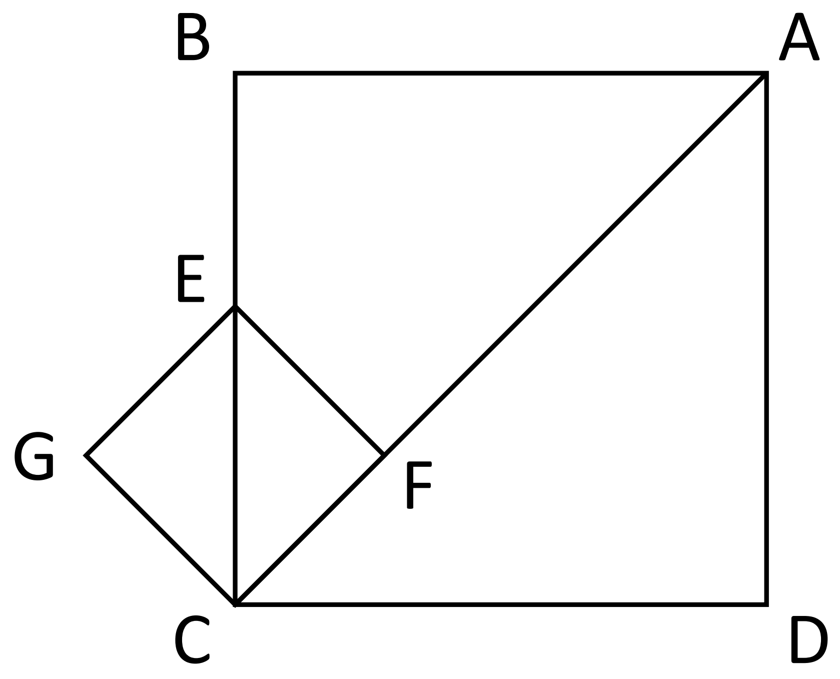
$$\angle CFE = 90^\circ$$
$$\angle ECF = 45^\circ$$

- And that three angles of a triangle sum to 180, so:

$$\begin{aligned}\angle CEF &= 180^\circ - \angle CFE - \angle ECF \\ &= 180^\circ - 90^\circ - 45^\circ \\ &= 45^\circ\end{aligned}$$

- Which means

$$\angle CEF = \angle ECF$$
$$\implies \overline{CF} = \overline{EF}$$



Proof (5)

- Add final segment BF.
- Triangle ABF is isoceles, so

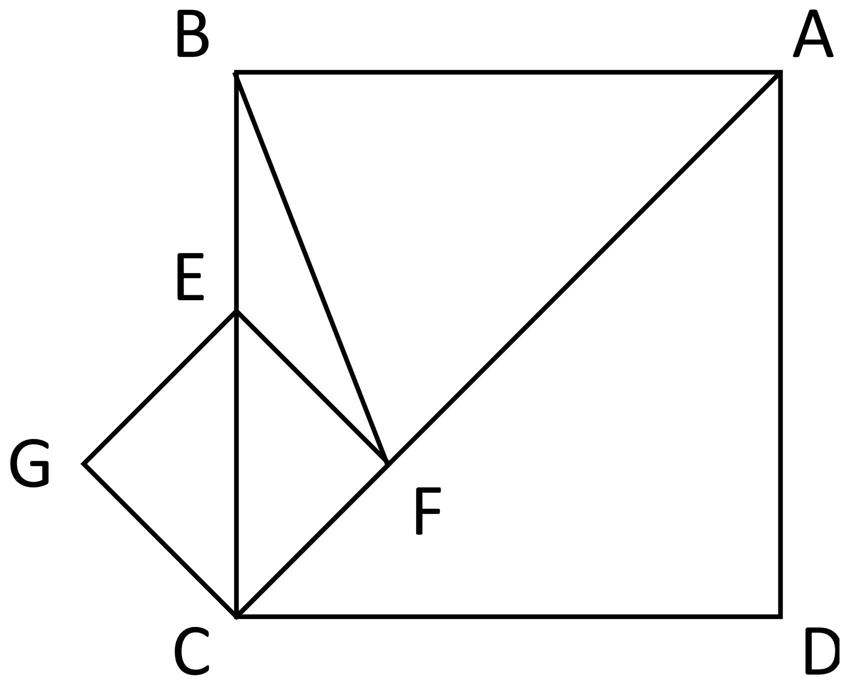
$$\angle ABF = \angle AFB$$

- By construction

$$\angle ABC = \angle AFE$$

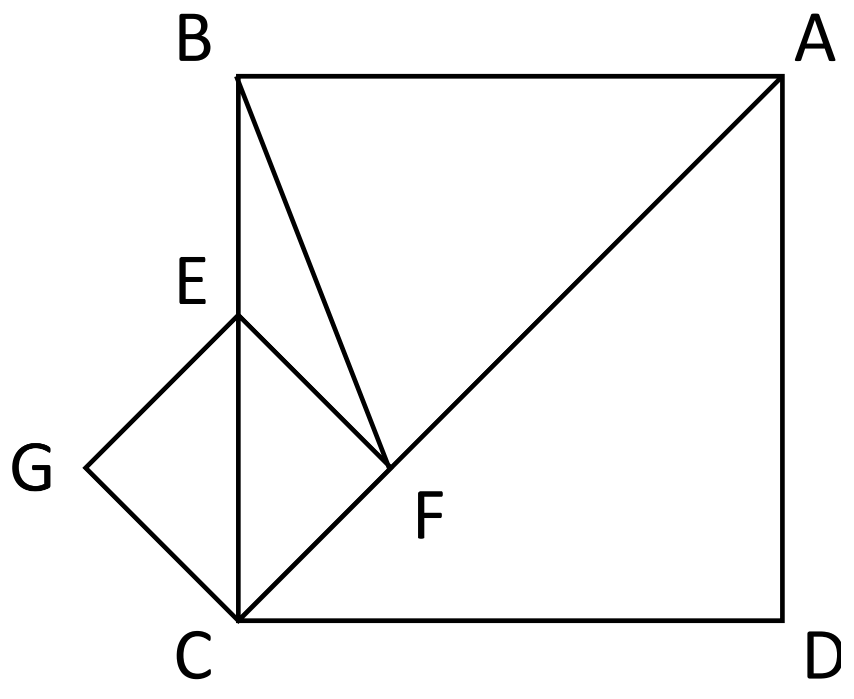
- So

$$\angle ABC - \angle ABF = \angle AFE - \angle AFB$$
$$\angle EBF = \angle EFB$$
$$\implies \overline{BE} = \overline{EF}$$



Proof (6)

- AC measurable by premise
- AF measurable since = AB
- So difference CF = AC – AF is measurable
- Since isoceles triangles
  $$\overline{CF} = \overline{EF} = \overline{BE}$$
- Since BC, CF, BE are measurable, difference EC = BC – BE is measurable.
- But now we have smaller square with measurable side and diagonal – contradiction!



What this means

- *There is no segment that can measure both the side and the diagonal of a square.*
  - If you try to find one using the GCM procedure, it will not terminate.
- The ratio of the diagonal and side of a square cannot be expressed as a ratio of integers
  - Pythagoreans discovered irrational numbers!
- Pythagorean program failed
  - Geometry cannot be constructed from integers

Consequences

- Discovery was shocking
- Pythagoreans swore followers to secrecy, but it leaked out
- Eventually, came up with a new plan:
  Use geometry, not numbers, as the unifying foundation of mathematics.

Alternate Proof of Irrationality of √2

- Assume √2 is rational. Then it can be expressed as irreducible ratio of two integers:

$$\begin{aligned} \frac{m}{n} &= \sqrt{2} \\ \left(\frac{m}{n}\right)^2 &= 2 \\ m^2 &= 2n^2 \end{aligned}$$

Alternate Proof of Irrationality of √2 (continued)

- m^2 is even, so m is even, so we can write as 2 times some other number u:

$$\begin{aligned} m &= 2u \\ (2u)^2 &= 2n^2 \\ 4u^2 &= 2n^2 \\ 2u^2 &= n^2 \end{aligned}$$

- n^2 is even, so n is even. But if m and n are both even, then m/n is not irreducible.

  Contradiction! Assumption must be false; i.e. we can't express √2 as the ratio of two integers.

Things to Consider

- It might seem that the ancient Greeks were naïve to try to represent continuous space with discrete numbers
  - Yet we do this every day when we use limited-precision computer arithmetic
- The tension between discrete and continuous has been a source of progress in mathematics

Lecture 5

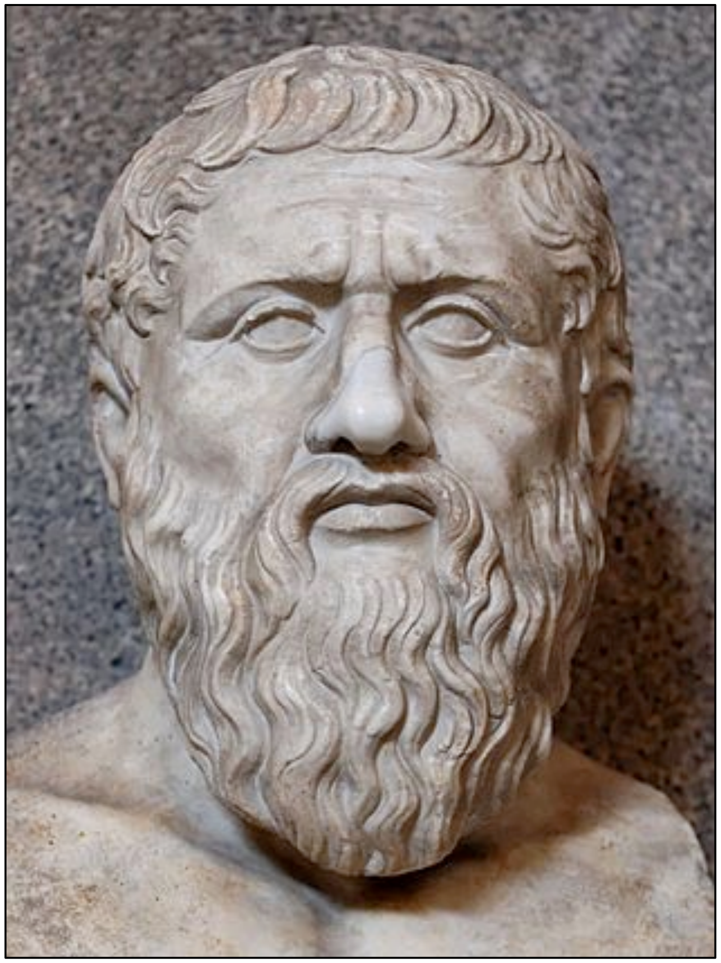
Euclid’s Greatest Common Measure Algorithm

(Covers material from Sec. 4.1-4.4)

Golden Age of Athens

- After unlikely defeat of invading Persians in 479 BC, Athens became center of culture, learning, and science for 150 years.
- Home of Plato’s Academy, essentially the world’s first university.

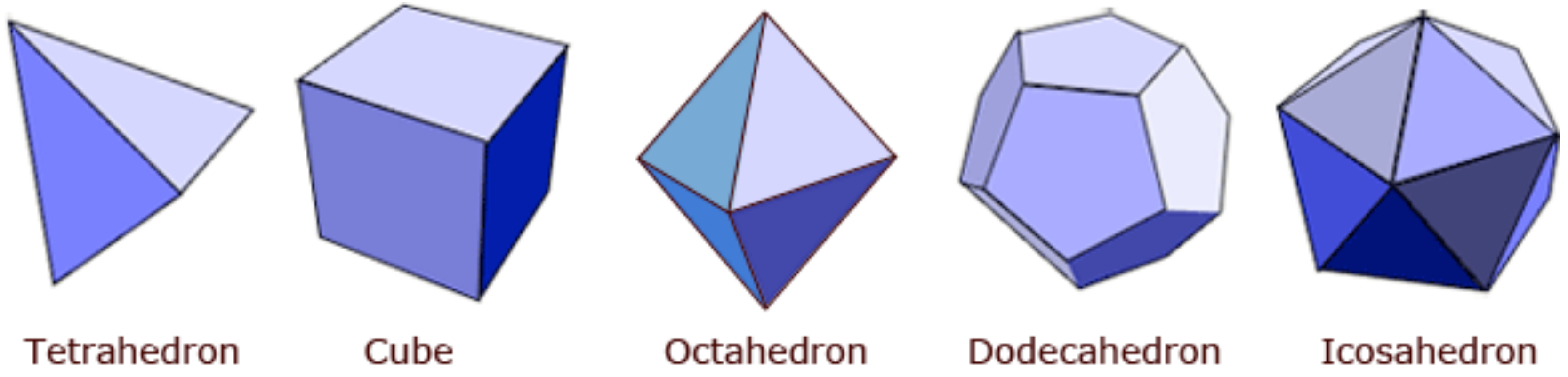
Plato (429BC - 347BC)



- Follower of Socrates, who taught that “the unexamined life is not worth living.”
- Founded Academy in ~385 BC
- European philosophy “consists of a series of footnotes to Plato.”
  – Alfred North Whitehead

Academy
"Let no one ignorant of geometry enter"

- 10 out of 15 years of study were fully dedicated to mathematics
- Among their discoveries: Five Platonic Solids
  (convex 3D shapes where each face is identical regular polygon)



Alexandria

- Founded in 331BC by (and named after) Alexander the Great
- Mouseion (“Institution of the Muses”)
  - A research institute with over 1000 scholars
- Library of Alexandria (part of Mouseion)
  - Goal of collecting all the world’s knowledge
  - 500,000 scrolls

Euclid (flourished ca. 300BC)

- May have studied at the Academy
- Worked at the Mouseion
- Wrote *Elements*
- “No royal road to geometry”

Euclid’s *Elements*

- A set of mathematical results and proofs woven into a coherent story
- Covers both geometry and number theory
  - Book I starts with basic ruler-and-compass constructions and builds to Pythagorean Theorem.
  - Book XIII shows how to construct the five Platonic solids, and proves that there are no others.
- No wasted operations
- Basis of mathematical education for centuries

Elements, Book X, Proposition 2

If, when the less of two unequal magnitudes is continually subtracted in turn from the greater, that which is left never measures the one before it, then the two magnitudes are incommensurable.

Euclid Provides the GCM Algorithm: *Elements* X,3

Given two commensurable magnitudes, to find their greatest common measure.

- Let the two given commensurable magnitudes be AB, CD, of which AB is the less; thus it is required to find the greatest common measure of AB, CD.

Elements X,3 Proof (2)

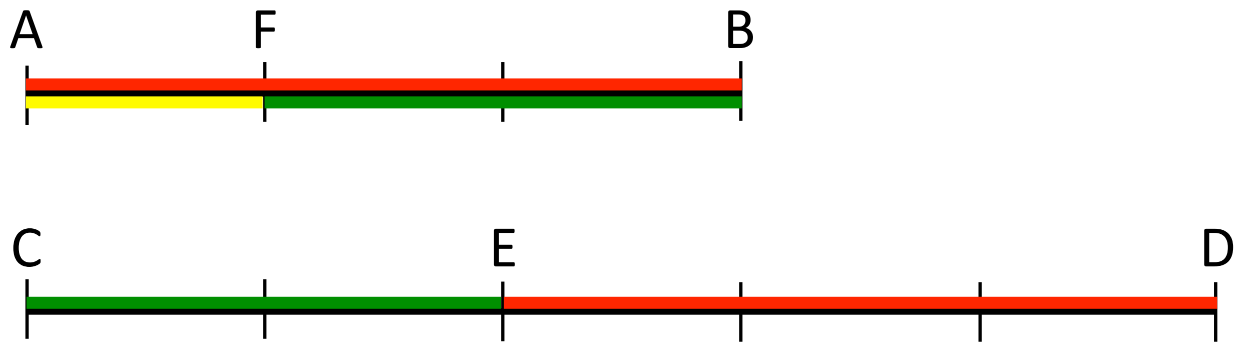
- Now the magnitude AB either measures CD or it does not.
- If then it measures it – and it measures itself also – AB is a common measure of AB, CD.
- And it is manifest that it is also the greatest; for a greater magnitude than the magnitude AB will not measure AB.

Elements X,3 Proof (3)

- Next, let AB not measure CD.
- Then, **if the less be continually subtracted in turn from the greater, that which is left over will sometime measure the one before it**…

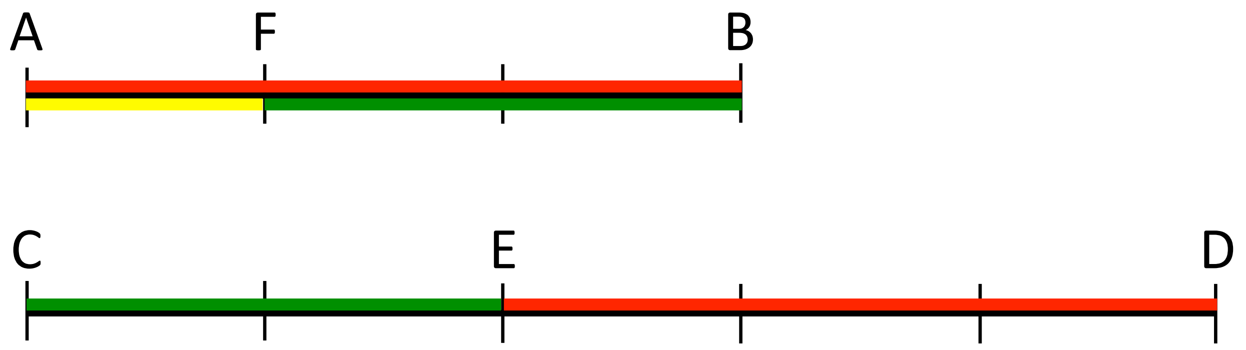
Elements X,3 Proof (4)

- let AB, measuring ED, leave EC less than itself
- let EC, measuring FB, leave AF less than itself
- and let AF measure CE.



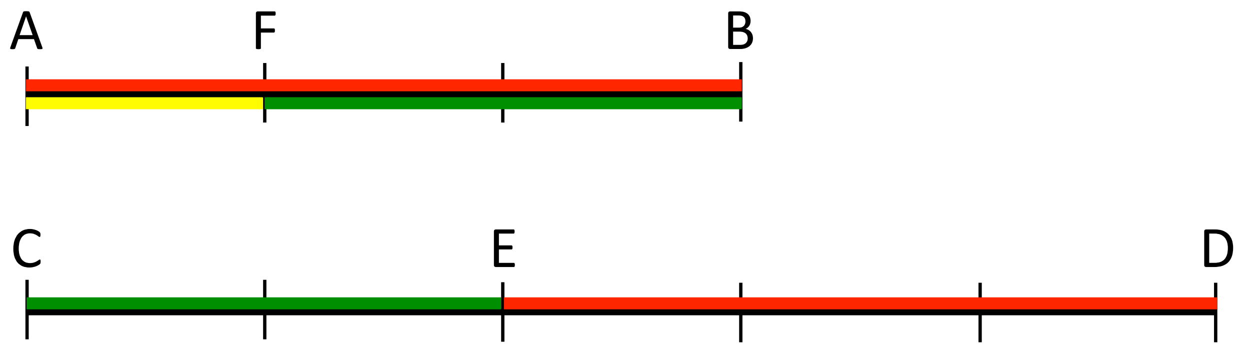
Elements X,3 Proof (5)

- Since, then, AF measures CE, while CE measures FB, therefore AF will also measure FB.
- But it measures itself also; therefore AF will also measure the whole AB.



Elements X,3 Proof (6)

- But AB measures DE; therefore AF will also measure ED.
- But it measures CE also; therefore it also measures the whole CD.
- **Therefore AF is a common measure of AB, CD.**



Euclid’s Algorithm in C++

```
line_segment gcm0(line_segment a,
                  line_segment b) {
    while (a != b) {
        if (b < a) a = a - b;
        else b = b - a;
    }
    return a;
}
```

a and *b* are incommensurable iff `gcm0` does not terminate.

GCM with Rearranged Operations

```
line_segment gcm1(line_segment a,
                  line_segment b) {
    while (a != b) {
        while (b < a) a = a - b;
        std::swap(a, b);
    }
    return a;
}
```

Observation: Inner `while` loop is computing remainder of *a* and *b*.

Factoring Out Remainder

```
line_segment segment_remainder(line_segment a,
                               line_segment b)
{
    while (b < a) a = a - b;
    return a;
}
```

How do we know the loop will terminate?

Axiom of Archimedes

For any quantities a and b, there is a natural number n such that $a \leq nb$.

GCM Using Remainder

```
line_segment gcm(line_segment a,
                 line_segment b) {
    while (a != b) {
        a = segment_remainder(a, b);
        std::swap(a, b);
    }
    return a;
}
```

Recursive Remainder Lemma

If $r = \text{segment\_remainder}(a, 2b)$, then

$$\text{segment\_remainder}(a, b) = \begin{cases} r & \text{if} \quad r \leq b \\ r - b & \text{if} \quad r > b \end{cases}$$

Recursive Remainder Example

- Suppose we want to find remainder(*n*, 10).
- Lemma tells us to try remainder(*n*, 20).
  - If the result is ≤ 10, we're done.
  - If it's between 11 and 20, we subtract 10 from the result and get the remainder that way.

e.g. want remainder(75, 10)

- compute remainder(75, 20) = 15
- original result is 15 – 10 = 5

fast_segment_remainder

```
line_segment
fast_segment_remainder(line_segment a,
                       line_segment b) {
    if (a <= b) return a;
    if (a - b <= b) return a - b;
    a = fast_segment_remainder(a, b + b);
    if (a <= b) return a;
    return a - b;
}
```

Tracing `fast_segment_remainder`

```
if (a <= b) return a;
if (a - b <= b) return a - b;
a = fast_segment_remainder(a, b + b);
if (a <= b) return a;
return a - b;
```

$a = 45, b = 6.$

$a \leq b?\ (45 \leq 6?)$ No.

$a - b \leq b?\ (39 \leq 6?)$ No.

Recurse:

> $a = 45, b = 12$
>
> $a \leq b?\ (45 \leq 12?)$ No.
>
> $a - b \leq b?\ (33 \leq 12?)$ No.
>
> Recurse:
>
> > $a = 45, b = 24$
> >
> > $a \leq b?\ (45 \leq 24?)$ No.
> >
> > $a - b \leq b?\ (21 \leq 24?)$ Yes, return $a - b = 21$
>
> $a \leftarrow 21$
>
> $a \leq b?\ (21 \leq 12?)$ No.
>
> return $a - b = 9$

$a \leftarrow 9$

$a \leq b?\ (9 \leq 6?)$ No.

return $a - b = 9 - 6 = 3$

GCM Using Fast Remainder

```
line_segment fast_segment_gcm(line_segment a,
                              line_segment b) {
    while (a != b) {
        a = fast_segment_remainder(a, b);
        std::swap(a, b);
    }
    return a;
}
```

Don’t disturb my circles!

Mathematics Goes Dormant (in the West)

- Ancient mathematics peaks around 3rd century BC
- Decline begins around the time of Archimedes’ death
- Romans interested in engineering, not math
- 1500 years of stagnation follows…

Meanwhile, in other places…

- Other great civilizations had developed their own mathematical traditions:
- In China, 3rd c. mathematician Liu Hui wrote commentaries on *Nine Chapters on the Mathematical Art*
- In India, 5th c. mathematician Aryabhata wrote the *Aryabhatiya*, a foundational text containing many important algorithms

Relationship to Modern Mathematics

- Modern mathematics (and computer science) descends from European mathematical tradition.
- But European mathematics was influenced by Arab, Jewish, and Persian mathematicians, writing in Arabic.
- These Arabic-speaking mathematicians were themselves influenced by ideas from India.

A Brief History of Zero

- Concept of zero and positional notation dates back at least to 1500 BC, used by Babylonian astronomers
  - – But they used it with base 60
- Greek astronomers borrowed this base 60 positional notation for trigonometry
- Still, no decimal 0 for 1000 years – despite widespread use of abacus.

Abacus

- Known throughout from ancient China to Rome
- Positional decimal representation
- Still widely used in 20th century

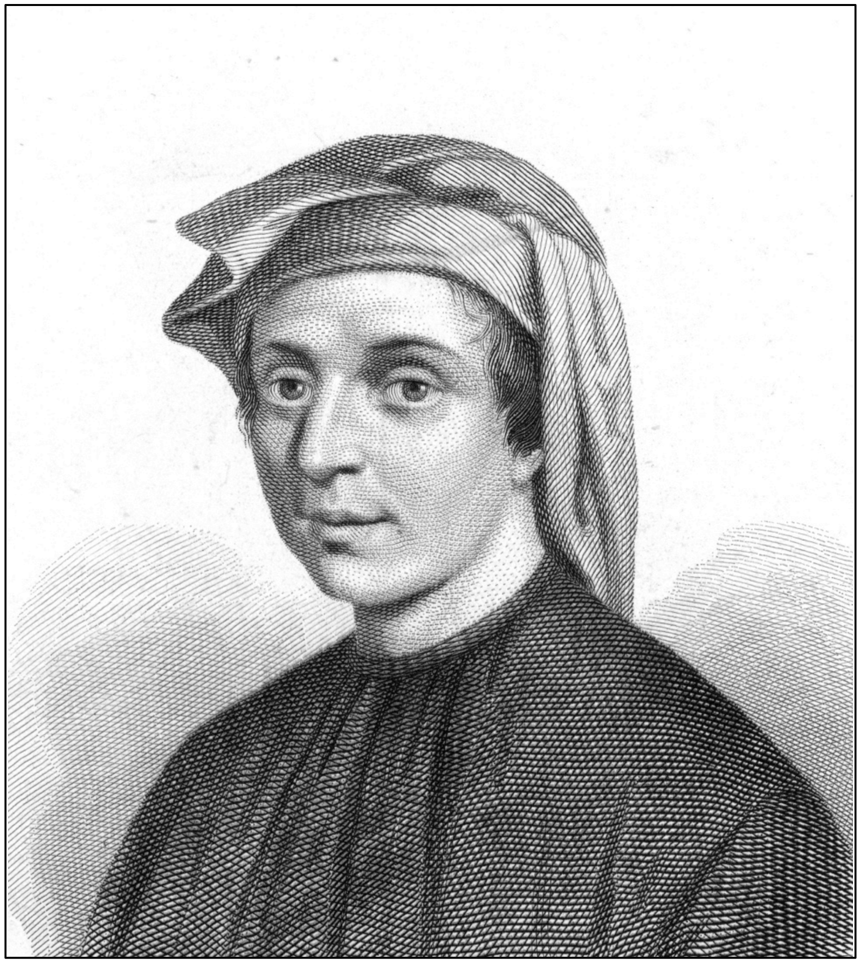
Decimal Zero Finally Arrives

- Indian mathematicians combined “natural” decimal integers with positional notation and zero around 6th century AD.
- Notation spread through India to Persia from 6th to 9th century AD.
- Arab scholars adopted it and it was taught from Bagdad to Cairo to Cordoba.

Rebirth of European Mathematics

- Publication in 1203 of *Liber Abaci* (“The Book of Calculation”) by Leonardo Pisano.
- Introduces 0 and decimal positional notation to Europeans
- Also now-standard algorithms for arithmetic
  - “long” addition, subtraction, division, multiplication

Leonardo Pisano, aka Fibonacci (1170-ca. 1240)



- Son of Bonacci, a Pisan trader, who brought Leonardo along when working in Algeria.
- Learned “Hindu digits” and other math from the Arabs
- Wrote most important math books in 1500 years

Things to Consider

- Many solutions in mathematics depend on having the right representation
- The same is true of programming

Lecture 6

Remainder and Quotient Algorithms

(Covers material from Sec. 4.5-4.8)

Zero and Segment Length

- Once we have the idea of zero, we can imagine a zero-length segment AA.
- Then we need to tweak our functions to handle zero-length segments.
  - Can't allow 2nd argument of remainder function to be 0.
  - Remainders shift to range [0, n–1].

Fast Segment Remainder with Zero

```
line_segment
fast_segment_remainder1(line_segment a,
                        line_segment b) {
    // precondition: b != 0
    if (a < b) return a;
    if (a - b < b) return a - b;
    a = fast_segment_remainder1(a, b + b);
    if (a < b) return a;
    return a - b;
}
```

Eliminating the Recursion

- Every time we recurse, we double argument *b*.
- Instead, let's precompute how many times we'll need to double.

First repeated doubling of *b* that exceeds $a - b$

```
line_segment largest_doubling(line_segment a,
                              line_segment b) {
    // precondition: b != 0
    while (a - b >= b) b = b + b;
    return b;
}
```

Undoubling

- Even though we double *b* for each recursive call, we still need the "undoubled" *b* in the rest of the function call.
- Fortunately, we know how to "undouble" – i.e. halve – a segment in our ruler-and-compass computer.

Iterative Remainder

```
line_segment remainder(line_segment a,
                       line_segment b) {
    // precondition: b != 0
    if (a < b) return a;
    line_segment c = largest_doubling(a, b);
    a = a - c;
    while (c != b) {
        c = half(c);
        if (c <= a) a = a - c;
    }
    return a;
}
```

Tracing Iterative Remainder

```
if (a < b) return a;
line_segment c =
    largest_doubling(a, b);
a = a - c;
while (c != b) {
    c = half(c);
    if (c <= a) a = a - c;
}
return a;
```

$a = 45, b = 6$

$a < b?\ (45 < 6?)$ No.

$c \leftarrow$ `largest_doubling`$(45, 6) = 24$

$a \leftarrow a - c = 45 - 24 = 21$

```
loop :
    $c \neq b?\ (24 \neq 6)?$ Yes, keep going.
        $c \leftarrow \texttt{half}(c) = \texttt{half}(24) = 12$
        $c \leq a?\ (12 \leq 21)?$ Yes. $a \leftarrow a - c = 21 - 12 = 9$
    $c \neq b?\ (12 \neq 6)?$ Yes, keep going.
        $c \leftarrow \texttt{half}(c) = \texttt{half}(12) = 6$
        $c \leq a?\ (6 \leq 9)?$ Yes. $a \leftarrow a - c = 9 - 6 = 3$
    $c \neq b?\ (6 \neq 6)?$ No, done with loop.
return $a = 3$
```

What if we want quotient instead of remainder?

```
integer quotient(line_segment a,
                 line_segment b) {
    // Precondition: b > 0
    if (a < b) return integer(0);
    line_segment c = largest_doubling(a, b);
    integer n(1);
    a = a - c;
    while (c != b) {
        c = half(c); n = n + n;
        if (c <= a) { a = a - c; n = n + 1; }
    }
    return n;
}
```

Tracing quotient(45, 6)

```
if (a < b) return integer(0);
line_segment c =
    largest_doubling(a, b);
integer n(1);
a = a - c;
while (c != b) {
    c = half(c); n = n + n;
    if (c <= a) {
        a = a - c; n = n + 1;
    }
}
return n;
```

$a = 45, b = 6$

$a < b?\ (45 < 6?)$ No.

$c \leftarrow \texttt{largest\_doubling}(45, 6) = 24$

$n \leftarrow 1$

$a \leftarrow a - c = 45 - 24 = 21$

loop :

$\quad c \neq b?\ (24 \neq 6)?$ Yes, keep going.

$\qquad c \leftarrow \texttt{half}(c) = \texttt{half}(24) = 12;\ \ n \leftarrow n + n = 1 + 1 = 2$

$\qquad c \leq a?\ (12 \leq 21)?$ Yes. $a \leftarrow a - c = 21 - 12 = 9;$

$\qquad\qquad n \leftarrow n + 1 = 2 + 1 = 3$

$\quad c \neq b?\ (12 \neq 6)?$ Yes, keep going.

$\qquad c \leftarrow \texttt{half}(c) = \texttt{half}(12) = 6;\ \ n \leftarrow n + n = 3 + 3 = 6$

$\qquad c \leq a?\ (6 \leq 9)?$ Yes. $a \leftarrow a - c = 9 - 6 = 3;$

$\qquad\qquad n \leftarrow n + 1 = 6 + 1 = 7$

$\quad c \neq b?\ (6 \neq 6)?$ No, done with loop.

return $n = 7$

Relationship of Multiplication and Division

- This quotient algorithm is the “algorithmic inverse” of Egyptian Multiplication
- A version appears in the Rhind Papyrus
- Greeks called it *Egyptian Division*

Since so much overlap between quotient and remainder, why bother with two functions?

Combining Quotient and Remainder

```
std::pair<integer, line_segment>
quotient_remainder(line_segment a,
                   line_segment b) {
    // Precondition: b > 0
    if (a < b) return {integer(0), a};
    line_segment c = largest_doubling(a, b);
    integer n(1);
    a = a - c;
    while (c != b) {
        c = half(c); n = n + n;
        if (c <= a) { a = a - c; n = n + 1; }
    }
    return {n, a};
}
```

The Law of Useful Return

If you’ve already done the work to get some useful result, don’t throw it away. Return it to the caller.

Could we do without half()?

- The half() operation is cheap on normal CPUs – it's just a right shift.
- But what if halving was difficult, expensive, or unavailable?

Floyd-Knuth: No Halving

```
line_segment remainder_fibo(line_segment a,
                            line_segment b) {
  // Precondition: b > 0
  if (a < b) return a;
  line_segment c = b;
  do {
    line_segment tmp = c; c = b + c; b = tmp;
  } while (a >= c);
  do {
    if (a >= b) a = a - b;
    line_segment tmp = c - b; c = b; b = tmp;
  } while (b < c);
  return a;
}
```

GCM Using Iterative Remainder

```
line_segment gcm_remainder(line_segment a,
                           line_segment b) {
    while (b != line_segment(0)) {
        a = remainder(a, b);
        std::swap(a, b);
    }
    return a;
}
```

Euclidean Algorithm for Integers

```
integer gcd(integer a, integer b) {
    while (b != integer(0)) {
        a = a % b;
        std::swap(a, b);
    }
    return a;
}
```

Now we call it Greatest Common Divisor (GCD).

How do we know the algorithm works?

What does it mean for an algorithm to “work”?

- It must terminate.
- It must compute what it’s supposed to (in this case, GCD).

Termination

We rely on the fact that

$$0 \leq \text{remainder}(a, b) < b$$

So in each iteration, b gets smaller.

Since any decreasing sequence of positive integers is finite, the algorithm must terminate.

Computing GCD (1)

- In each iteration, algorithm computes remainder(a, b), which by definition is:

$$r = a - bq$$

where q is the quotient a divided by b.

- Since gcd(a, b) by definition divides a and also divides b (and therefore bq), it must divide r.

Computing GCD (2)

- We can rewrite the remainder equation as:

$$a = bq + r$$

- Since gcd(b, r) by definition divides b (and therefore bq) and also r, it must divide a.

- Since pairs (a, b) and (b, r) have the same common divisors, they have the same GCD.

Computing GCD (3)

- So we have shown that

$$a = bq + r \implies \text{gcd}(a, b) = \text{gcd}(b, r)$$

- At each iteration, the algorithm replaces gcd(a, b) with gcd(b, r) by taking the remainder and swapping the arguments.

Computing GCD (4)

Remainders computed at each iteration:

$$
\begin{aligned}
r_1 &= \text{remainder}(a_0, b_0) \\
r_2 &= \text{remainder}(b_0, r_1) \\
r_3 &= \text{remainder}(r_1, r_2) \\
&\cdots \\
r_n &= \text{remainder}(r_{n-2}, r_{n-1})
\end{aligned}
$$

Using definition of remainder, rewrite as:

$$
\begin{aligned}
r_1 &= a_0 - b_0 q_1 \\
r_2 &= b_0 - r_1 q_2 \\
r_3 &= r_1 - r_2 q_3 \\
&\cdots \\
r_n &= r_{n-2} - r_{n-1} q_n
\end{aligned}
$$

Computing GCD (5)

- We already showed that GCD stays the same each time, i.e.

$$gcd(a_0, b_0) = gcd(b_0, r_1) = gcd(r_1, r_2) = \cdots = gcd(r_{n-1}, r_n)$$

- But we know that remainder($r_{n\text{-}1}$, r_n) = 0, because that's what terminates the algorithm.
- And gcd(x, 0) = x. So

$$gcd(a_0, b_0) = \cdots = gcd(r_{n-1}, r_n) = gcd(r_n, 0) = r_n$$

which is the value returned by the algorithm.
So the algorithm computes GCD.

Things to Consider

- When programming, look for ways to avoid duplicating work – and once you've computed something, don't throw it away!
  - i.e. Follow the Law of Useful Return

Lecture 7

First Ideas in Modern Number Theory

(Covers material from Sec. 5.1-5.3)

Deep Dive Into Mathematics

The next few lectures focus exclusively on math:

- Some important results in number theory
- Foundations of abstract algebra

Later we'll see how these ideas apply to programming.

The Renaissance of Number Theory

- Mathematicians of the 15th-18th centuries revived the Ancient Greek interest in primes and perfect numbers.
- Special interest in primes of the form $2^n - 1$, since they could generate perfect numbers.

When is $2^n - 1$ prime?

- Greeks: true for $n = 2, 3, 5, 7$, possibly 13
- Hudalricus Regius (1536): false for $n = 11$

$$2^{11} - 1 = 2047 = 23 \times 89$$

- Pietro Cataldi (1603): true for $n = 17, 19, 23, 29, 31, 37$
- But half of those (in red) were wrong;
  Pierre de Fermat showed that:

$$2^{23} - 1 = 8388607 = 47 \times 178481$$

$$2^{37} - 1 = 137438953471 = 223 \times 61631877$$

Mersenne's Conjecture

In 1644, Mersenne wrote that
for $n \leq 257$, $2^n - 1$ is prime if and only if

$$n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257$$

Two of these (shown in red) are wrong, and he missed 89 and 107, but we still call primes of this form *Mersenne Primes*.

Marin Mersenne (1588 – 1648)

- His letters shared scientific results across countries
- Corresponded with friends
  - like Galileo, Descartes, Torricelli, Huygens, Pascal
- Helped Galileo get his work published

How did Fermat factor $2^{37} - 1$?

In June 1640 Fermat wrote to Mersenne that his factorization of $2^{37} - 1$ depends on the following three observations:

1. If n is not a prime, $2^{n} - 1$ is not a prime.
2. If n is a prime, $2^{n} - 2$ is a multiple of $2n$.
3. If n is a prime, and p is a prime divisor of $2^{n} - 1$, then $p - 1$ is a multiple of n.

Fermat’s Logic

- If $2^{37} – 1$ isn’t prime it must have an odd prime factor p.
- By Fermat’s observation 3, $p - 1$ is a multiple of 37, i.e.

$$p = 37u + 1$$

- Since p is odd, $p – 1 = 37u$ must be even, so u must be even. So we can write u as $2v$, i.e.

$$p = 74v + 1$$

- So the problem is reduced to trying values of v:

  $v = 1$? No, 75 is not a prime

  $v = 2$? No, 149 is prime, but is not a divisor of $2^{37} – 1$.

  $v = 3$? Yes, 223 is prime and is a divisor of $2^{37} – 1$.

Proving Fermat's Observation 1: If *n* is not a prime, $2^n - 1$ is not a prime

- Instead of proving "If X then Y," we'll prove "If not Y, then not X" – the *contrapositive*, which is equivalent.

- In this case:

  If $2^n - 1$ is prime, then *n* is prime

If $2^n - 1$ is prime, then *n* is prime

Proof: Suppose *n* is not prime. Then there must be factors *u* and *v* such that

$$n = uv,\ \ u > 1, v > 1$$

So we can rewrite, and use difference of powers:

$$\begin{aligned} 2^n - 1 &= 2^{uv} - 1 \\ &= (2^u)^v - 1 \\ &= (2^u - 1)((2^u)^{v-1} + (2^u)^{v-2} + \cdots + (2^u) + 1) \end{aligned}$$

But then we've factored $2^n - 1$ into two numbers each > 1, so $2^n - 1$ is not prime. Contradiction!

What about Fermat’s other 2 observations?

- He said in a letter that he would have shared the proof, but it would make the letter too long.
- Typical Fermat.

Pierre de Fermat (1601 – 1665)

- A true Renaissance man
- The last great amateur mathematician
- Invented Analytic Geometry
- Co-invented Probability (with Pascal)
- Rarely shared his proofs

Fermat's Unproved Conjecture

- Proofs have been found for all of Fermat's conjectures, except one:

$$2^n + 1 \text{ is prime} \iff n = 2^i$$

- Numbers of this form are called *Fermat Primes*.
- We can prove the first half:

$$2^n + 1 \text{ is prime} \implies n = 2^i$$

$2^n + 1$ is prime $\implies$ $n = 2^i$

- Suppose $n \neq 2^i$. Then it must have an odd factor, which we can write as $2q + 1$, where $q > 0$.
- Since $2q + 1 > 1$, we can write n as
$$n = m(2q + 1)$$
- Substituting and using sum of odd powers:
$$\begin{aligned} 2^n + 1 &= 2^{m(2q+1)} + 1 \\ &= 2^{m(2q+1)} + 1^{m(2q+1)} \\ &= (2^m)^{2q+1} + 1^{2q+1} \\ &= (2^m + 1)((2^m)^{2q} - (2^m)^{2q-1} + \cdots + 1) \end{aligned}$$
- We factored $2^n + 1$: contradiction!

Other Primes of Form $2^{2^i}+1$

- Fermat conjectured that all of these are prime
- Fermat said that 3, 5, 17, 257, 65537, 4294967297 and 18446744073709551617 are primes. (Red ones are wrong.)
- In 1732 Leonhard Euler showed that

$$2^{32}+1 = 4294967297 = 641 \times 6700417$$

- For $5 \leq i \leq 32$ they are composite.
- Are there any more Fermat primes? Unknown.

Fermat’s Little Theorem

If p is prime, $a^{p-1} - 1$ is divisible by p
for any $0 < a < p$

- Fermat claimed to have proved in 1640
- Leibniz claimed to have proved in 1670s
- Finally, Euler published proof in 1742

Leonhard Euler (1707 – 1783)

- A math superstar, recruited by kings
- Developed modern analysis (calculus and differential equations)
- Wrote first book on popular science
- It took 60 years to publish all his work after his death

Proving Fermat’s Little Theorem: Laying the Foundation

1. Euclid proposition VII, 30
2. Permutation of Remainders Lemma
3. Cancellation Law
4. Self-canceling Law
5. Wilson’s Theorem

Then we’ll put all the pieces together.

Euclid VII, 30: *The product of two integers smaller than a prime p is not divisible by p.*

Another way to say this is if p is prime and a and b are smaller than p, then ab is not divisible by p.

If a number x is not divisible by a number y, we can write x as a multiple of y plus a remainder: $x = my + r$. So we can similarly restate the proposition as:

$$p \text{ is prime } \;\wedge\; 0 < a, b < p \implies ab = mp + r \;\wedge\; 0 < r < p$$

Euclid VII, 30 Restated:

$$p \text{ is prime} \;\; \wedge \;\; 0 < a, b < p \;\; \Longrightarrow \;\; ab = mp + r \;\; \wedge \;\; 0 < r < p$$

Proof: Assume contrary, that ab is a multiple of p.

For a given a, let b be smallest integer such that $ab = mp$.
Since p is prime, dividing p by b leaves a remainder $v < b$:

$$p = bu + v \;\; \wedge \;\; 0 < v < b$$

Multiply both sides by a and substitute $ab = mp$:

$$\begin{aligned} ap &= abu + av \\ ap - abu &= av \\ ap - mpu &= av \\ av &= (a - mu)p \quad \wedge \quad 0 < v < b \end{aligned}$$

Then v is an integer smaller than b such that av is a multiple of p: contradiction.

Permutation of Remainders Lemma

If p is prime, then for any $0 < a < p$,

$$a \cdot \{1,\ \ldots,\ p-1\} =$$
$$\{a,\ \ldots,\ a(p-1)\} = \{q_1 p + r_1,\ \ldots,\ q_{p-1} p + r_{p-1}\}$$

where

$$0 < r_i < p \ \wedge \ i \neq j \implies r_i \neq r_j$$

Permutation of Remainders Example

If p = 7 and a = 4, then the lemma says that

$$\{4, 8, 12, 16, 20, 24\} =$$
$$\{0 \cdot 7 + 4,\ 1 \cdot 7 + 1,\ 1 \cdot 7 + 5,\ 2 \cdot 7 + 2,\ 2 \cdot 7 + 6,\ 3 \cdot 7 + 3\}$$

so the remainders are

$$\{4, 1, 5, 2, 6, 3\}$$

which is a permutation of

$$\{1, \ldots, 7 - 1\}$$

Permutation of Remainders Proof

- Suppose $r_i = r_j$ and $i < j$
- Then we could take difference of two corresponding elements, and the remainders r_i and r_j would cancel:

$$(q_j p + r_j) - (q_i p + r_i) = q_j p - q_i p = (q_j - q_i)p$$

- Since ith and jth elements are ai and aj, we can write the difference as $ai - aj$. That is:

$$aj - ai = (q_j - q_i)p$$
$$a(j - i) = (q_j - q_i)p$$

- But this has form $ab = mp$, which implies product of two integers smaller than p is divisible by p.
- Contradicts Euclid VII, 30, which we just proved.

Cancellation

- If we have two numbers x and y, and xy = 1, we say that they *cancel*.
- If x and y cancel, each is the *multiplicative inverse* of the other.
- We're used to rational numbers canceling, but today we'll look at integer cancellation.

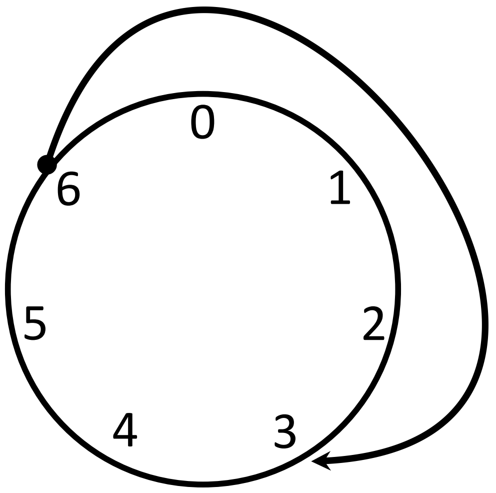
Modular Arithmetic

Analogy: 12-hour clock

- If it’s 10 o’clock, and something will take 5 hours, then it will be done at 3 o’clock.
- In a sense, $10 + 5 = 3$.
- More precisely, $(10 + 5) \bmod 12 = 3$.

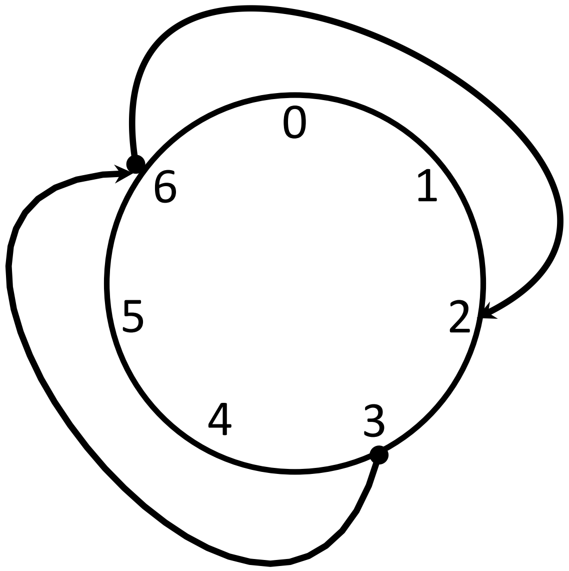
Modular Arithmetic with Other Bases

$(6 + 4) \bmod 7 = 3$



$(3 \times 3) \bmod 7 =$

$(3 + 3 + 3) \bmod 7 = 2$



$3 \times 3 = 9 = (1 \times 7) + 2$

Modular Arithmetic and Cancellation

- Consider the equation

$$(2 \times 4) \bmod 7 = 1$$

- Since the product of 2 and 4 is 1, they cancel and are each other's inverse.

Modular Arithmetic and Negation

- A negative number $x \bmod n$ is equal to the positive number $n - x$.
- It's like turning the odometer back x positions.

Modular Multiplication Table

×	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Example: $5 \times 4 = 20 = 2 \times 7 + 6 = 6 \bmod 7.$

Modular Multiplication with Inverses

$\times$	1	2	3	4	5	6	
1	**1**	2	3	4	5	6	1
2	2	4	6	**1**	3	5	4
3	3	6	2	5	**1**	4	5
4	4	**1**	5	2	6	3	2
5	5	3	**1**	6	4	2	3
6	6	5	4	3	2	**1**	6

Example: 2 and 4 are inverses, because $2 \times 4 = 1$.

Formal Definition of Integer Inverses

For integer $n > 1$ and integer $u > 0$, v is a *multiplicative inverse modulo* n if there is an integer q such that $uv = 1 + qn$.

In other words, u and v are inverses if their product divided by n yields a remainder of 1.

Cancellation Law

If p is prime, then for any $0 < a < p$
there is a number $0 < b < p$ such that

$$ab = mp + 1$$

Proof:

By the Permutation of Remainders Lemma, we know that one of the possible products in the set

$$a \cdot \{1, \ \ldots, \ p - 1\}$$

will have remainder 1.

So there must be another element b that cancels a.

Next Lecture…

- Proving Fermat’s Little Theorem

Lecture 8

Proving Fermat's Little Theorem
(Covers material from Sec. 5.3-5.7)

Proving Fermat's Little Theorem

If p is prime, $a^{p-1} - 1$ is divisible by p
for any $0 < a < p$

Foundations:

1. Euclid proposition VII, 30 (done)
2. Permutation of Remainders Lemma (done)
3. Cancellation Law
4. Self-canceling Law
5. Wilson's Theorem

Cancellation Refresher

- If we have two numbers x and y, and $xy = 1$, we say that they *cancel*.
- If x and y cancel, each is the *multiplicative inverse* of the other.
- In modular arithmetic, x and y are inverses modulo n if there is an integer q such that

$$xy = 1 + qn$$

Self-Canceling Elements

- 1 and $p - 1$ are *self-canceling* elements
- If you multiply each by itself, the result is $1 \bmod p$, or equivalently, can be expressed as $mp + 1$.

Self-Canceling Law

For any $0 < a < p$,

$$a^2 = mp + 1 \quad \Longrightarrow \quad a = 1 \ \lor \ a = p - 1$$

Proof:

- Suppose there were some other self-canceling a that's neither 1 nor $p - 1$. It must be $1 < a < p - 1$.
- Rearranged proof condition gives us $a^2 - 1 = mp$.
- Factoring gives $(a - 1)(a + 1) = mp$.
- But now we have two integers $< p$ whose product is divisible by p. Contradicts Euclid VII, 30.

Wilson's Theorem

If p is prime, there is an integer m such that

$$(p-1)! = mp + (p-1)$$

or, equivalently:

$$(p-1)! = (p-1) \bmod p$$

Proof of Wilson's Theorem: p prime $\implies \exists m : (p-1)! = mp + (p-1)$

- By definition: $(p-1)! = 1 \cdot 2 \cdot 3 \ldots (p-1)$
- By Cancellation and Self-canceling laws, each number in above product except 1 and $p-1$ is cancelled by its own inverse, i.e. has remainder 1.
- So we could collapse all cancelled terms as $np+1$:

$$\begin{aligned}(p-1)! &= 1 \cdot (np+1) \cdot (p-1) \\ &= np \cdot p - np + p - 1 \\ &= (np-n)p + (p-1)\end{aligned}$$

- Then $m = np - n$ satisfies the theorem.

Proof of Fermat's Little Theorem (1)

If p prime, $a^{p-1} - 1$ is divisible by p for any $0 < a < p$

- Consider the product: $\displaystyle\prod_{i=1}^{p-1} ai = a^{p-1} \prod_{i=1}^{p-1} i$
- Write Wilson's Theorem as: $\displaystyle\prod_{i=1}^{p-1} i = (p-1) + mp$
- Substitute: $\displaystyle\prod_{i=1}^{p-1} ai = a^{p-1}((p-1) + mp)$

$$= a^{p-1}p - a^{p-1} + a^{p-1}mp$$

$$= \boxed{(a^{p-1} + a^{p-1}m)p - a^{p-1}}$$

Proof of Fermat's Little Theorem (2)

If p prime, $a^{p-1} - 1$ is divisible by p for any $0 < a < p$

- The terms of the first product are $\{a, 2a, \ldots, (p\text{-}1)a\}$ which by Permutation of Remainders is $\{q_1 p + r_1, \ldots, q_{p\text{-}1} p + r_{p\text{-}1}\}$
- So we can write: $$\prod_{i=1}^{p-1} ai = \prod_{i=1}^{p-1} (q_i p + r_i)$$
- Expand and group p terms: $$\prod_{i=1}^{p-1} ai = up + \prod_{i=1}^{p-1} r_i$$
- Apply Wilson's, group again: $$\prod_{i=1}^{p-1} ai = up + vp + (p-1)$$ $$= \boxed{wp - 1}$$

where $w = u + v + 1$.

Proof of Fermat's Little Theorem (3)

If p prime, $a^{p-1} - 1$ is divisible by p for any $0 < a < p$

- We know the two boxed expressions are equal, so we put them together and rearrange:

$$\boxed{wp - 1} = \boxed{(a^{p-1} + a^{p-1}m)p - a^{p-1}}$$

$$a^{p-1} + wp - 1 = (a^{p-1} + a^{p-1}m)p$$

$$a^{p-1} - 1 = (a^{p-1} + a^{p-1}m)p - wp$$

- Collapse multiples of p on right:

$$a^{p-1} - 1 = np$$

- So $a^{p-1} - 1$ is divisible by p. QED.

Another Observation

a^{p-2} is an inverse of a, since

$$a^{p-2} \cdot a = a^{p-1}$$

which by Fermat's Little Theorem is $mp + 1$.

Converse of Fermat's Little Theorem

If for all a, $0 < a < n$

$$a^{n-1} = 1 + q_a n$$

then n is prime.

Coprimes

Two numbers m and n are *coprime* if

$$\gcd(m, n) = 1$$

Equivalently, m and n are coprime if they have no common factor greater than 1.

Non-Invertibility Lemma

If $n = uv \;\wedge\; u, v > 1,$ then u is not invertible modulo n.

Proof: Let $n = uv$ and w be an inverse of u. Then

$$\begin{aligned} wn &= wuv \\ &= (mn+1)v \\ &= mvn + v \\ wn - mvn &= v \end{aligned}$$

If we define $z = (w - mv)$, then

$$(w - mv)n = zn = v$$

Since $n > v$, then $zn > v$, which is a contradiction with $zn = v$. So u cannot have an inverse.

Converse of Fermat's Little Theorem

If for all a, $0 < a < n$, $a^{n-1} = 1 + q_a n$, then n is prime.

Proof: Suppose n is not prime, i.e. $n = uv$.

Then by Non-Invertibility lemma, u is not invertible.

But by condition of the theorem,

$$u^{n-1} = u^{n-2}u = 1 + q_a n$$

In other words, u has an inverse u^{n-2}, which is a contradiction.

So n must be prime.

Generalizing Fermat’s Little Theorem

- Fermat’s Little Theorem is a result about primes.
- Euler wondered if there is a similar result for composite numbers.

Reminder: Modular Multiplication for Prime

×	1	2	3	4	5	6	
1	**1**	2	3	4	5	6	1
2	2	4	6	**1**	3	5	4
3	3	6	2	5	**1**	4	5
4	4	**1**	5	2	6	3	2
5	5	3	**1**	6	4	2	3
6	6	5	4	3	2	**1**	6

Modular Multiplication for Composite

×	1	2	3	4	5	6	7	8	9	
1	1	2	3	4	5	6	7	8	9	1
2	2	4	6	8	0	2	4	6	8	
3	3	6	9	2	5	8	1	4	7	7
4	4	8	2	6	0	4	8	2	6	
5	5	0	5	0	5	0	5	0	5	
6	6	2	8	4	0	6	2	8	4	
7	7	4	1	8	5	2	9	6	3	3
8	8	6	4	2	0	8	6	4	2	
9	9	8	7	6	5	4	3	2	1	9

Example: $7 \times 9 = 63 = 3 \bmod 10$

Modular Multiplication: Prime vs. Composite

Prime

- All rows are permutations.
- Every element has inverse.
- All elements are > 0.

Composite

- Rows are not permutations.
- Only some elements have inverse.
- Some elements are 0.

Huh?
How can the product of two nonzero things be 0?

A Subset of Composite Modular Multiplication Table

×	1	2	3	4	5	6	7	8	9	
$\boxed{1}$	**1**	2	**3**	4	5	6	**7**	8	**9**	**1**
2	2	4	6	8	0	2	4	6	8	
$\boxed{3}$	**3**	6	**9**	2	5	8	**1**	4	**7**	7
4	4	8	2	6	0	4	8	2	6	
5	5	0	5	0	5	0	5	0	5	
6	6	2	8	4	0	6	2	8	4	
$\boxed{7}$	**7**	4	**1**	8	5	2	**9**	6	**3**	**3**
8	8	6	4	2	0	8	6	4	2	
$\boxed{9}$	**9**	8	**7**	6	5	4	**3**	2	**1**	**9**

How Many Coprimes?

The *totient* of a positive integer n is the number of positive integers less than n that are coprime with n:

$$\phi(n) = |\{0 < i < n \ \wedge \ \text{coprime}(i, n)\}|$$

For our modulo 10 table: $\phi(10) = 4$

(the number of shaded rows)

How Many Coprimes for a Prime?

$$\phi(p) = p - 1$$

In other words, all numbers less than a given prime are coprime with it.

Euler’s Insight

The “p–1” in Fermat’s Little Theorem is just a special case of $\phi(p)$ for primes.

Euler's Theorem

$$\text{coprime}(a, n) \iff a^{\phi(n)} - 1 \textit{ is divisible by } n.$$

Proof strategy: Modify proof of Fermat's Little Theorem.

- Replace Permutation of Remainders Lemma with Permutation of Coprime Remainders Lemma
- Prove that every coprime remainder has a multiplicative inverse.
- Use the product of all coprime remainders where the proof of Little Fermat has the product of all nonzero remainders.

Computing $\phi(n)$ for any integer

- We can express any integer as the product of powers of primes, so first we'll see how to compute the totient of a power of a prime p.
- How many coprimes of p^m? At most $p^m - 1$.
- But need to subtract multiples of p, because we know those aren't coprime.

Totient of Prime Power

$$\begin{aligned}\phi(p^m) &= (p^m - 1) - |\{p, 2p, \ldots, p^m - p\}| \\ &= (p^m - 1) - |\{1, 2, \ldots, p^{m-1} - 1\}| \\ &= (p^m - 1) - (p^{m-1} - 1) \\ &= p^m - p^{m-1} \\ &= p^m \left(1 - \frac{1}{p}\right)\end{aligned}$$

Totient of Product of Prime Powers: $\phi(p^u q^v)$

Let $n = p^u q^v$.

$$\begin{aligned}
\phi(n) &= (n-1) - \left(\frac{n}{p} - 1\right) - \left(\frac{n}{q} - 1\right) + \left(\frac{n}{pq} - 1\right) \\
&= n - \frac{n}{p} - \frac{n}{q} + \frac{n}{pq} \\
&= n\left(1 - \frac{1}{p} - \frac{1}{q} + \frac{1}{pq}\right) \\
&= n\left[\left(1 - \frac{1}{p}\right) - \frac{1}{q}\left(1 - \frac{1}{p}\right)\right] \\
&= n\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) \\
&= p^u\left(1 - \frac{1}{p}\right) q^v\left(1 - \frac{1}{q}\right) \\
&= \phi(p^u)\phi(q^v)
\end{aligned}$$

Special Case: Product of Two Primes

$$\phi(p_1 p_2) = \phi(p_1)\,\phi(p_2)$$

- Example: Since 10 = 5 x 2

$$\phi(10) = \phi(5)\,\phi(2) = 4$$

General Case: Product of m Prime Powers

Let $n = \prod_{i=1}^{m} p_i^{k_i}$

$$\begin{aligned} \phi(n) &= \phi\left(\prod_{i=1}^{m} p_i^{k_i}\right) \\ &= n \prod_{i=1}^{m} \left(1 - \frac{1}{p_i}\right) \\ &= \prod_{i=1}^{m} \phi\left(p_i^{k_i}\right) \end{aligned}$$

Wilson's Theorem with Modular Arithmetic

- Wilson's theorem says for a prime p, there is an m such that

$$(p-1)! = (p-1) + mp$$

  or equivalently

$$(p-1)! = (p-1) \bmod p$$

- Suppose $p = 7$ and $p{-}1 = 6$. So let's expand 6!:

$$\begin{aligned}\mathbf{6!} &= 1 \times 2 \times 3 \times 4 \times 5 \times 6 \\ &= 1 \times (2 \times 4) \times (3 \times 5) \times 6 \\ &= (1 \times 1 \times 1 \times 6) \bmod 7 \\ &= 6 \bmod 7\end{aligned}$$

  which is what Wilson's theorem predicts.

Fermat's Little Theorem with Modular Arithmetic

Restate "If p is prime, $a^{p-1} - 1$ is divisible by p for any $0 < a < p$" as "If p is prime, $a^{p-1} = 1 \bmod \mathrm{p}$ for any $0 < a < p$."

We'll try $p = 7$ and $a = 2$:

$$\begin{aligned}
2^6 &= (2 \times 2 \times 2 \times 2 \times 2 \times 2) \\
2^6 \times 6! &= (2 \times 2 \times 2 \times 2 \times 2 \times 2) \times (1 \times 2 \times 3 \times 4 \times 5 \times 6) \\
&= (2 \times 1) \times (2 \times 2) \times (2 \times 3) \times (2 \times 4) \times (2 \times 5) \times (2 \times 6) \\
&= (2 \times 4 \times 6 \times 1 \times 3 \times 5) \bmod 7 \\
&= (1 \times 2 \times 3 \times 4 \times 5 \times 6) \bmod 7 \\
&= 6! \bmod 7 \\
2^6 &= 1 \bmod 7
\end{aligned}$$

It works!

Things to Consider

- Euler saw how to take a result for a specific smaller set (primes) and generalize it to work for a larger set (all positive integers)
- This idea of relaxing requirements is at the heart of generic programming