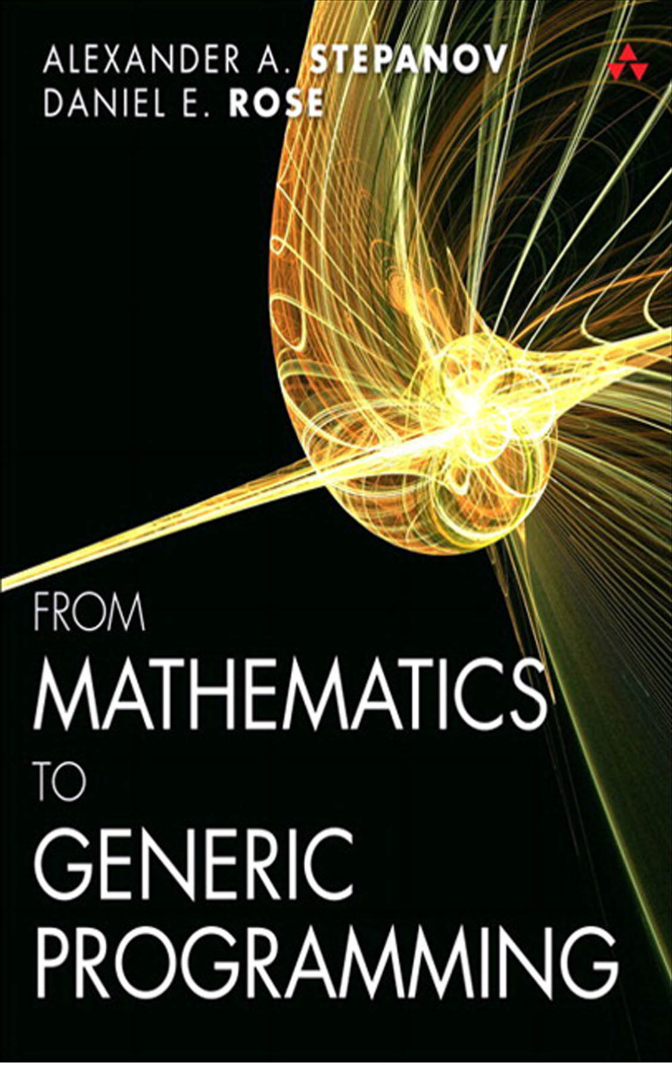
ALEXANDER A. STEPANOV
DANIEL E. ROSE



FROM MATHEMATICS TO GENERIC PROGRAMMING

From Mathematics to Generic Programming

This page intentionally left blank

From Mathematics to Generic Programming

Alexander A. Stepanov

Daniel E. Rose

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Stepanov, Alexander A.
From mathematics to generic programming / Alexander A. Stepanov, Daniel E. Rose.
pages cm
Includes bibliographical references and index.
ISBN 978-0-321-94204-3 (pbk. : alk. paper)
1. Generic programming (Computer science)—Mathematics. 2. Computer algorithms. I. Rose, Daniel E. II. Title.
QA76.6245.S74 2015
005.1'1—dc23

2014034539

Photo credits are listed on page 293.

ISBN-13: 978-0-321-94204-3
ISBN-10: 0-321-94204-3
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, November 2014

Contents

Acknowledgments ix
About the Authors xi
Authors' Note xiii

1 What This Book Is About 1
- **1.1 Programming and Mathematics 2**
- **1.2 A Historical Perspective 2**
- **1.3 Prerequisites 3**
- **1.4 Roadmap 4**

2 The First Algorithm 7
- **2.1 Egyptian Multiplication 8**
- **2.2 Improving the Algorithm 11**
- **2.3 Thoughts on the Chapter 15**

3 Ancient Greek Number Theory 17
- **3.1 Geometric Properties of Integers 17**
- **3.2 Sifting Primes 20**
- **3.3 Implementing and Optimizing the Code 23**
- **3.4 Perfect Numbers 28**
- **3.5 The Pythagorean Program 32**
- **3.6 A Fatal Flaw in the Program 34**
- **3.7 Thoughts on the Chapter 38**

4 Euclid's Algorithm 41
- **4.1 Athens and Alexandria 41**
- **4.2 Euclid's Greatest Common Measure Algorithm 45**
- **4.3 A Millennium without Mathematics 50**
- **4.4 The Strange History of Zero 51**
- **4.5 Remainder and Quotient Algorithms 53**
- **4.6 Sharing the Code 57**
- **4.7 Validating the Algorithm 59**
- **4.8 Thoughts on the Chapter 61**

5 The Emergence of Modern Number Theory 63
5.1 Mersenne Primes and Fermat Primes 63
5.2 Fermat's Little Theorem 69
5.3 Cancellation 72
5.4 Proving Fermat's Little Theorem 77
5.5 Euler's Theorem 79
5.6 Applying Modular Arithmetic 83
5.7 Thoughts on the Chapter 84

6 Abstraction in Mathematics 85
6.1 Groups 85
6.2 Monoids and Semigroups 89
6.3 Some Theorems about Groups 92
6.4 Subgroups and Cyclic Groups 95
6.5 Lagrange's Theorem 97
6.6 Theories and Models 102
6.7 Examples of Categorical and Non-categorical Theories 104
6.8 Thoughts on the Chapter 107

7 Deriving a Generic Algorithm 111
7.1 Untangling Algorithm Requirements 111
7.2 Requirements on A 113
7.3 Requirements on N 116
7.4 New Requirements 118
7.5 Turning Multiply into Power 119
7.6 Generalizing the Operation 121
7.7 Computing Fibonacci Numbers 124
7.8 Thoughts on the Chapter 127

8 More Algebraic Structures 129
8.1 Stevin, Polynomials, and GCD 129
8.2 Göttingen and German Mathematics 135
8.3 Noether and the Birth of Abstract Algebra 140
8.4 Rings 142
8.5 Matrix Multiplication and Semirings 145
8.6 Application: Social Networks and Shortest Paths 147
8.7 Euclidean Domains 150
8.8 Fields and Other Algebraic Structures 151
8.9 Thoughts on the Chapter 152

9 Organizing Mathematical Knowledge 155
9.1 Proofs 155
9.2 The First Theorem 159

9.3 Euclid and the Axiomatic Method 161
9.4 Alternatives to Euclidean Geometry 164
9.5 Hilbert's Formalist Approach 167
9.6 Peano and His Axioms 169
9.7 Building Arithmetic 173
9.8 Thoughts on the Chapter 176

10 Fundamental Programming Concepts 177
10.1 Aristotle and Abstraction 177
10.2 Values and Types 180
10.3 Concepts 181
10.4 Iterators 184
10.5 Iterator Categories, Operations, and Traits 185
10.6 Ranges 188
10.7 Linear Search 190
10.8 Binary Search 191
10.9 Thoughts on the Chapter 196

11 Permutation Algorithms 197
11.1 Permutations and Transpositions 197
11.2 Swapping Ranges 201
11.3 Rotation 204
11.4 Using Cycles 207
11.5 Reverse 212
11.6 Space Complexity 215
11.7 Memory-Adaptive Algorithms 216
11.8 Thoughts on the Chapter 217

12 Extensions of GCD 219
12.1 Hardware Constraints and a More Efficient Algorithm 219
12.2 Generalizing Stein's Algorithm 222
12.3 Bézout's Identity 225
12.4 Extended GCD 229
12.5 Applications of GCD 234
12.6 Thoughts on the Chapter 234

13 A Real-World Application 237
13.1 Cryptology 237
13.2 Primality Testing 240
13.3 The Miller-Rabin Test 243
13.4 The RSA Algorithm: How and Why It Works 245
13.5 Thoughts on the Chapter 248

14 Conclusions 249

Further Reading 251

A Notation 257

B Common Proof Techniques 261
B.1 Proof by Contradiction 261
B.2 Proof by Induction 262
B.3 The Pigeonhole Principle 263

C C++ for Non-C++ Programmers 265
C.1 Template Functions 265
C.2 Concepts 266
C.3 Declaration Syntax and Typed Constants 267
C.4 Function Objects 268
C.5 Preconditions, Postconditions, and Assertions 269
C.6 STL Algorithms and Data Structures 269
C.7 Iterators and Ranges 270
C.8 Type Aliases and Type Functions with `using` in C++11 272
C.9 Initializer Lists in C++11 272
C.10 Lambda Functions in C++11 272
C.11 A Note about `inline` 273

Bibliography 275
Index 281

Acknowledgments

We would like to thank all the people who contributed to making this book a reality. Our management at A9.com actively supported this project from the beginning. Bill Stasior initiated the creation of the course this book is based on, and selected the topic from among several options we offered. Brian Pinkerton not only attended the course, but also strongly encouraged our idea of turning the material into a book. We also would like to thank Mat Marcus, who collaborated with Alex on a similar course at Adobe in 2004–2005.

The other members of the Fundamental Data Structures and Algorithms for Search team played important roles throughout the process. Anil Gangolli helped shape the content of the course, Ryan Ernst provided much of the programming infrastructure, and Paramjit Oberoi gave invaluable feedback during the writing stage. We have enjoyed working with all of them and are grateful for their input.

We are grateful to our editors, Peter Gordon and Greg Doench, and to the team of experts assembled by Addison-Wesley, including managing editor John Fuller, production editor Mary Kesel Wilson, copyeditor Jill Hobbs, and compositor/LaTeX expert Lori Hughes for all their work in turning our rough manuscript into a polished book.

Finally, we'd like to thank the many friends, family members, and colleagues who read earlier drafts of the book and/or gave us comments, corrections, suggestions, advice, or other help: Gašper Ažman, John Banning, Cynthia Dwork, Hernan Epelman, Ryan Ernst, Anil Gangolli, Susan Gruber, Jon Kalb, Robert Lehr, Dmitry Leshchiner, Tom London, Mark Manasse, Paul McJones, Nicolas Nicolov, Gor Nishanov, Paramjit Oberoi, Sean Parent, Fernando Pelliccioni, John Reiser, Robert Rose, Stefan Vargyas, and Adam Young. The book is much better as a result of their contributions.

This page intentionally left blank

About the Authors

Alexander A. Stepanov studied mathematics at Moscow State University from 1967 to 1972. He has been programming since 1972: first in the Soviet Union and, after emigrating in 1977, in the United States. He has programmed operating systems, programming tools, compilers, and libraries. His work on foundations of programming has been supported by GE, Polytechnic University, Bell Labs, HP, SGI, Adobe, and, since 2009, A9.com, Amazon's search technology subsidiary. In 1995 he received the *Dr. Dobb's Journal* Excellence in Programming Award for the design of the C++ Standard Template Library.

Daniel E. Rose is a research scientist who has held management positions at Apple, AltaVista, Xigo, Yahoo, and A9.com. His research focuses on all aspects of search technology, ranging from low-level algorithms for index compression to human–computer interaction issues in web search. Rose led the team at Apple that created desktop search for the Macintosh. He holds a Ph.D. in cognitive science and computer science from University of California, San Diego, and a B.A. in philosophy from Harvard University.

This page intentionally left blank

Authors' Note

The separation of computer science from mathematics greatly impoverishes both. The lectures that this book is based on were my attempt to show how these two activities—an ancient one going back to the very beginnings of our civilization and the most modern one—can be brought together.

I was very fortunate that my friend Dan Rose, under whose management our team was applying principles of generic programming to search engine design, agreed to convert my rather meandering lectures into a coherent book. Both of us hope that our readers will enjoy the result of our collaboration.

—A.A.S.

The book you are about to read is based on notes from an "Algorithmic Journeys" course taught by Alex Stepanov at A9.com during 2012. But as Alex and I worked together to transform the material into book form, we realized that there was a stronger story we could tell, one that centered on generic programming and its mathematical foundations. This led to a major reorganization of the topics, and removal of the entire section on set theory and logic, which did not seem to be part of the same story. At the same time, we added and removed details to create a more coherent reading experience and to make the material more accessible to less mathematically advanced readers.

While Alex comes from a mathematical background, I do not. I've tried to learn from my own struggles to understand some of the material and to use this experience to identify ideas that require additional explanation. If in some cases we describe something in a slightly different way than a mathematician would, or using slightly different terminology, or using more simple steps, the fault is mine.

—D.E.R.

This page intentionally left blank

This page intentionally left blank

3 three

Ancient Greek Number Theory

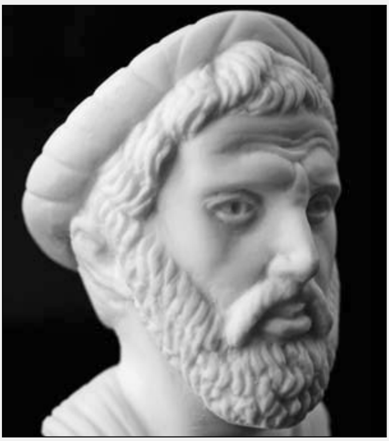
Pythagoreans applied themselves to the study of mathematics....
They thought that its principles must be the principles of all existing things.
Aristotle, *Metaphysics*

In this chapter, we're going to look at some of the problems studied by ancient Greek mathematicians. Their work on patterns and "shapes" of numbers led to the discovery of prime numbers and the beginnings of a field of mathematics called *number theory*. They also discovered paradoxes that ultimately produced some mathematical breakthroughs. Along the way, we'll examine an ancient algorithm for finding primes, and see how to optimize it.

3.1 Geometric Properties of Integers

Pythagoras, the Greek mathematician and philosopher who most of us know only for his theorem, was actually the person who came up with the idea that understanding mathematics is necessary to understand the world. He also discovered many interesting properties of numbers; he considered this understanding to be of great value in its own right, independent of any practical application. According to Aristotle's pupil Aristoxenus, "He attached supreme importance to the study of arithmetic, which he advanced and took out of the region of commercial utility."

Pythagoras (ca. 570 BC–ca. 490 BC)



Pythagoras was born on the Greek island of Samos, which was a major naval power at the time. He came from a prominent family, but chose to pursue wisdom rather than wealth. At some point in his youth he traveled to Miletus to study with Thales, the founder of philosophy (see Section 9.2), who advised him to go to Egypt and learn the Egyptians' mathematical secrets.

During the time Pythagoras was studying abroad, the Persian empire conquered Egypt. Pythagoras followed the Persian army eastward to Babylon (in what is now Iraq), where he learned Babylonian mathematics and astronomy. While there, he may have met travelers from India; what we know is that he was exposed to and began espousing ideas we typically associate with Indian religions, including the transmigration of souls, vegetarianism, and asceticism. Prior to Pythagoras, these ideas were completely unknown to the Greeks.

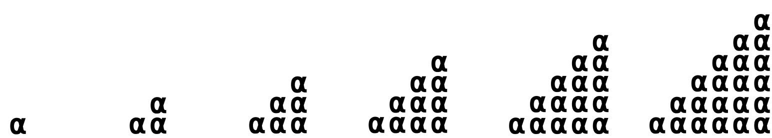
After returning to Greece, Pythagoras started a settlement in Croton, a Greek colony in southern Italy, where he gathered followers—both men and women—who shared his ideas and followed his ascetic lifestyle. Their lives were centered on the study of four things: astronomy, geometry, number theory, and music. These four subjects, later known as the *quadrivium*, remained a focus of European education for 2000 years. Each of these disciplines was related: the motion of the stars could be mapped geometrically, geometry could be grounded in numbers, and numbers generated music. In fact, Pythagoras was the first to discover the numerical structure of frequencies in musical octaves. His followers said that he could "hear the music of the celestial spheres."

After the death of Pythagoras, the Pythagoreans spread to several other Greek colonies in the area and developed a large body of mathematics. However, they kept their teachings secret, so many of their results may have been lost. They also eliminated competition within their ranks by crediting all discoveries to Pythagoras himself, so we don't actually know which individuals did what.

Although the Pythagorean communities were gone after a couple of hundred years, their work remains influential. As late as the 17th century, Leibniz (one of the inventors of calculus) described himself as a Pythagorean.

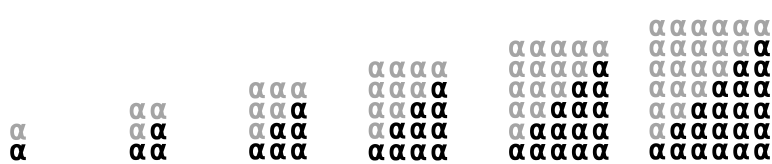
Unfortunately, Pythagoras and his followers kept their work secret, so none of their writings survive. However, we know from contemporaries what some of his discoveries were. Some of these come from a first-century book called *Introduction to Arithmetic* by Nicomachus of Gerasa. These included observations about geometric properties of numbers; they associated numbers with particular shapes.

Triangular numbers, for example, which are formed by stacking rows representing the first n integers, are those that formed the following geometric pattern:



1 3 6 10 15 21

Oblong numbers are those that look like this:



2 6 12 20 30 42

It is easy to see that the nth oblong number is represented by an $n \times (n + 1)$ rectangle:

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

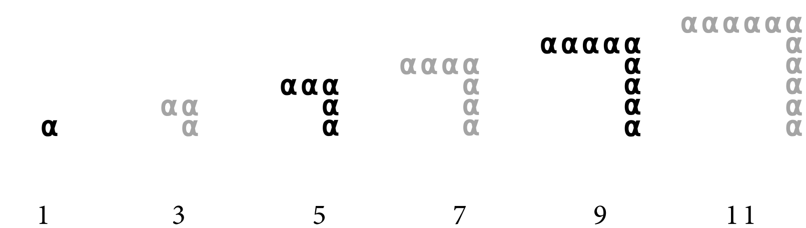
It's also clear geometrically that each oblong number is twice its corresponding triangular number. Since we already know that triangular numbers are the sum of the first n integers, we have

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

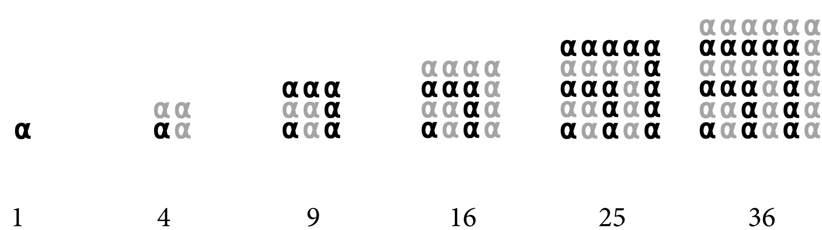
So the geometric representation gives us the formula for the sum of the first n integers:

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

Another geometric observation is that the sequence of odd numbers forms the shape of what the Greeks called *gnomons* (the Greek word for a carpenter's square; a gnomon is also the part of a sundial that casts the shadow):



Combining the first n gnomons creates a familiar shape—a square:



This picture also gives us a formula for the sum of the first n odd numbers:

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

Exercise 3.1. Find a geometric proof for the following: take any triangular number, multiply it by 8, and add 1. The result is a square number. (This problem comes from Plutarch's *Platonic Questions*.)

3.2 Sifting Primes

Pythagoreans also observed that some numbers could not be made into any nontrivial rectangular shape (a shape where both sides of the rectangle are greater than 1). These are what we now call *prime numbers*—numbers that are not products of smaller numbers:

$$2, 3, 5, 7, 11, 13, \ldots$$

("Numbers" for the Greeks were always whole numbers.) Some of the earliest observations about primes come from Euclid. While he is usually associated with geometry, several books of Euclid's *Elements* actually discuss what we now call number theory. One of his results is this theorem:

Theorem 3.1 (Euclid VII, 32): *Any number is either prime or divisible by some prime.*

The proof, which uses a technique called "impossibility of infinite descent," goes like this:[1]

Proof. Consider a number A. If it is prime, then we are done. If it is composite (i.e., nonprime), then it must be divisible by some smaller number B. If B is prime, we are done (because if A is divisible by B and B is prime, then A is divisible by a prime). If B is composite, then it must be divisible by some smaller number C, and so on. Eventually, we will find a prime or, as Euclid remarks in his proof of the previous proposition, "an infinite sequence of numbers will divide the number, each of which is less than the other; and this is impossible." □

This Euclidean principle that *any descending sequence of natural numbers terminates* is equivalent to the induction axiom of natural numbers, which we will encounter in Chapter 9.

* * *

Another result, which some consider the most beautiful theorem in mathematics, is the fact that there are infinitely many primes:

Theorem 3.2 (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

$$q = 1 + \prod_{i=1}^{n} p_i$$

[1] Euclid's proof of VII, 32 actually relies on his proposition VII, 31 (any composite number is divisible by some prime), which contains the reasoning shown here.

where p_i is the ith prime in the sequence. Because of the way we constructed q, we know it is not divisible by any p_i. Then either q is prime, in which case it is itself a prime not in the sequence, or q is divisible by some new prime, which by definition is not in the sequence. Therefore, there are infinitely many primes. $\square$

One of the best-known techniques for finding primes is the *Sieve of Eratosthenes*. Eratosthenes was a 3rd-century Greek mathematician who is remembered in part for his amazingly accurate measurement of the circumference of the Earth. Metaphorically, the idea of Eratosthenes' sieve is to "sift" all the numbers so that the nonprimes "fall through" the sieve and the primes remain at the end. The actual procedure is to start with a list of all the candidate numbers and then cross out the ones known not to be primes (since they are multiples of primes found so far); whatever is left are the primes. Today the Sieve of Eratosthenes is often shown starting with all positive integers up to a given number, but Eratosthenes already knew that even numbers were not prime, so he didn't bother to include them.

Following Eratosthenes' convention, we'll also include only odd numbers, so our sieve will find primes greater than 2. Each value in the sieve is a candidate prime up to whatever value we care about. If we want to find primes up to a maximum of $m = 53$, our sieve initially looks like this:

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

In each iteration, we take the first number (which must be a prime) and cross out all the multiples except itself that have not previously been crossed out. We'll highlight the numbers being crossed out in the current iteration by boxing them. Here's what the sieve looks like after we cross out the multiples of 3:

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

Next we cross out the multiples of 5 that have not yet been crossed out:

3	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

And then the remaining multiples of 7:

3	5	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

We need to repeat this process until we've crossed out all the multiples of factors less than or equal to $\lfloor\sqrt{m}\rfloor$, where m is the highest candidate we're considering.

In our example, $m = 53$, so we are done. All the numbers that have not been crossed out are primes:

$$
\begin{array}{ccccccccccccc}
\mathbf{3} & \mathbf{5} & \mathbf{7} & \cancel{\mathbf{9}} & \mathbf{11} & \mathbf{13} & \cancel{\mathbf{15}} & \mathbf{17} & \mathbf{19} & \cancel{\mathbf{21}} & \mathbf{23} & \cancel{\mathbf{25}} & \cancel{\mathbf{27}} \\
\mathbf{29} & \mathbf{31} & \cancel{\mathbf{33}} & \cancel{\mathbf{35}} & \mathbf{37} & \cancel{\mathbf{39}} & \mathbf{41} & \mathbf{43} & \cancel{\mathbf{45}} & \mathbf{47} & \cancel{\mathbf{49}} & \cancel{\mathbf{51}} & \mathbf{53}
\end{array}
$$

Before we write our implementation of the algorithm, we'll make a few observations. Let's go back to what the sieve looked like in the middle of the process (say, when we were crossing out multiples of 5) and add some information—namely, the index, or position in the list, of each candidate being considered:

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	…
values:	**3**	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	$\cancel{39}$	…

Notice that when we're considering multiples of factor 5, the *step size*—the number of entries between two numbers being crossed out, such as 25 and 35—is 5, the same as the factor. Another way to say this is that the difference between the *indexes* of any two candidates being crossed out in a given iteration is the same as the factor being used. Also, since the list of candidates contains only odd numbers, the difference between two values is twice as much as the difference between two indexes. So the difference between two numbers being crossed out in a given iteration (e.g., between 25 and 35) is twice the step size or, equivalently, twice the factor being used. You'll see that this pattern holds for all the factors we considered in our example as well.

Finally, we observe that the first number crossed out in each iteration is the square of the prime factor being used. That is, when we're crossing out multiples of 5, the first one that wasn't previously crossed out is 25. This is because all the other multiples were already accounted for by previous primes.

3.3 Implementing and Optimizing the Code

At first glance it seems like our algorithm will need to maintain two arrays: one containing the candidate numbers we're sifting—the "values"—and another containing Boolean flags indicating whether the corresponding number is still there or has been crossed out. However, after a bit of thought it becomes clear that we don't actually need to store the values at all. Most of the values (namely, all the nonprimes) are never used. When we do need a value, we can compute it from its position; we know that the first value is 3 and that each successive value is 2 more than the previous one, so the ith value is $2i + 3$.

So our implementation will store just the Boolean flags in the sieve, using `true` for prime and `false` for composite. We call the process of "crossing out" nonprimes *marking* the sieve. Here's a function we'll use to mark all the non-primes for 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;
    }
}
```

We are using the convention of "declaring" our template arguments with a description of their requirements. We will discuss these requirements, known as *concepts*, in detail later on in Chapter 10; for now, readers can consult Appendix C as a reference. (If you are not familiar with C++ templates, these are also explained in this appendix.)

As we'll see shortly, we'll call this function with `first` pointing to the Boolean value corresponding to the first "uncrossed-out" multiple of `factor`, which as we saw is always `factor`'s square. For `last`, we'll follow the STL convention of passing an iterator that points just past the last element in our table, so that `last - first` is the number of elements.

* * *

Before we see how to sift, we observe the following sifting lemmas:

- The square of the smallest prime factor of a composite number c is less than or equal to 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 $p^2 \geq m$.

We will use the following formulas in our computation:

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

step between multiple k and multiple $k+1$ of value at i:

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

index of square of value at i:

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

We can now make our first attempt at implementing the 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 candidate is prime
            mark_sieve(first + index_square,
                       first + n,  // last
                       i + i + 3); // factor
        }
        ++i;
        index_square =  2*i*(i + 3) + 3;
    }
}
```

It might seem that we should pass in a reference to a data structure containing the Boolean sequence, since the sieve works only if we sift the whole thing. But by instead passing an iterator to the beginning of the range, together with its length, we don't constrain which kind of data structure to use. The data could be in an STL container or in a block of memory; we don't need to know. Note that we use the size of the table n rather than the maximum value to sift m.

The variable `index_square` is the index of the first value we want to mark—that is, the square of the current factor. One thing we notice is that we're computing the factor we use to mark the sieve ($i + i + 3$) and other quantities (shown in *slanted text*) every time through the loop. We can hoist common subexpressions out of the loop; the changes are shown in **bold**:

```
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;
    }
}
```

The astute reader will notice that the `factor` computation is actually slightly worse than before, since it happens every time through the loop, not just on iterations when the `if` test is true. However, we shall see later why making `factor` a separate variable makes sense. A bigger issue is that we still have a relatively expensive operation—the computation of `index_square`, which involves two multiplications. So we will take a cue from compiler optimization and use a technique known as *strength reduction*, which was designed to replace more expensive operations like multiplication with equivalent code that uses less expensive operations like addition.[2] If a compiler can do this automatically, we can certainly do it manually.

Let's look at these computations in more detail. Suppose we replaced

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

with

`factor +=` δ_{factor}`;`

[2]While multiplication is not necessarily slower than addition on modern processors, the general technique can still lead to using fewer operations.

`index_square +=` δ_{index_square}`;`

where δ_{factor} and δ_{index_square} are the differences between successive (ith and $i+1$st) values of `factor` and `index_square`, respectively:

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

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

δ_{factor} is easy; the variables cancel and we get the constant 2. But how did we simplify the expression for δ_{index_square}? We observe that by rearranging the terms, we can express it using something we already have, `factor(i)`, and something we need to compute anyway, `factor(i + 1)`. (When you know you need to compute multiple quantities, it's useful to see if one can be computed in terms of another. This might allow you to do less work.)

With these substitutions, we get our final version of `sift`; again, our improvements are shown in bold:

```
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;
    }
}
```

Exercise 3.2. Time the sieve using different data sizes: bit (using `std::vector<bool>`), `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t`.

Exercise 3.3. Using the sieve, graph the function

$$\pi(n) = \text{number of primes} < n$$

for n up to 10^7 and find its analytic approximation.

We call primes that read the same backward and forward *palindromic primes*. Here we've highlighted the ones up to 1000:

$\boxed{\textbf{2}}$ $\boxed{\textbf{3}}$ $\boxed{\textbf{5}}$ $\boxed{\textbf{7}}$ $\boxed{\textbf{11}}$ 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
83 89 97 $\boxed{\textbf{101}}$ 103 107 109 113 127 $\boxed{\textbf{131}}$ 137 139 149 $\boxed{\textbf{151}}$ 157 163
167 173 179 $\boxed{\textbf{181}}$ $\boxed{\textbf{191}}$ 193 197 199 211 223 227 229 233 239 241 251
257 263 269 271 277 281 283 293 307 311 $\boxed{\textbf{313}}$ 317 331 337 347 349
$\boxed{\textbf{353}}$ 359 367 $\boxed{\textbf{373}}$ 379 $\boxed{\textbf{383}}$ 389 397 401 409 419 421 431 433 439
443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547
557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643
647 653 659 661 673 677 683 691 701 709 719 $\boxed{\textbf{727}}$ 733 739 743 751
$\boxed{\textbf{757}}$ 761 769 773 $\boxed{\textbf{787}}$ 797 809 811 821 823 827 829 839 853 857 859
863 877 881 883 887 907 911 $\boxed{\textbf{919}}$ $\boxed{\textbf{929}}$ 937 941 947 953 967 971 977
983 991 997

Interestingly, there are no palindromic primes between 1000 and 2000:

1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087
1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181
1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279
1283 1289 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373
1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471
1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559
1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637
1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747
1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867
1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951 1973
1979 1987 1993 1997 1999

Exercise 3.4. Are there palindromic primes > 1000? What is the reason for the lack of them in the interval [1000, 2000]? What happens if we change our base to 16? To an arbitrary n?

3.4 Perfect Numbers

As we saw in Section 3.1, the ancient Greeks were interested in all sorts of properties of numbers. One idea they came up with was that of a *perfect* number—

a number that is the sum of its proper divisors.[3] They knew of four perfect numbers:

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

Perfect numbers were believed to be related to nature and the structure of the universe. For example, the number 28 was the number of days in the lunar cycle.

What the Greeks really wanted to know was whether there was a way to predict other perfect numbers. They looked at the prime factorizations of the perfect numbers they knew:

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

and noticed the following pattern:

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

The result of this expression is perfect when the the second term is prime. It was Euclid who presented the proof of this fact around 300 BC.

Theorem 3.3 (Euclid IX, 36):

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

[3] A proper divisor of a number n is a divisor of n other than n itself.

Useful Formulas

Before we look at the proof, it is useful to remember a couple of algebraic formulas. The first is the *difference of powers*:

$$\begin{aligned} x^2 - y^2 &= (x-y)(x+y) \\ x^3 - y^3 &= (x-y)(x^2 + xy + y^2) \\ &\vdots \\ x^{n+1} - y^{n+1} &= (x-y)(x^n + x^{n-1}y + \cdots + xy^{n-1} + y^n) \end{aligned} \tag{3.1}$$

This result can easily be derived using these two equations:

$$x(x^n + x^{n-1}y + \cdots + xy^{n-1} + y^n) = x^{n+1} + x^n y + x^{n-1}y^2 + \cdots + xy^n \tag{3.2}$$

$$y(x^n + x^{n-1}y + \cdots + xy^{n-1} + y^n) = x^n y + x^{n-1}y^2 + \cdots + xy^n + y^{n+1} \tag{3.3}$$

The left and right sides of 3.2 and 3.3 are equal by the distributive law. If we then subtract 3.3 from 3.2, we get 3.1.

The second useful formula is for the *sum of odd powers*:

$$x^{2n+1} + y^{2n+1} = (x+y)(x^{2n} - x^{2n-1}y + \cdots - xy^{2n-1} + y^{2n}) \tag{3.4}$$

which we can derive by converting the sum to a difference and relying on our previous result:

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

We can get away with this because -1 to an odd power is still -1. We will rely heavily on both of these formulas in the proofs ahead.

Now we know that for $n > 0$

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1 \tag{3.5}$$

by the difference of powers formula:

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

(or just think of the binary number you get when you add powers of 2).

Exercise 3.5. Using Equation 3.1, prove that if $2^n - 1$ is prime, then n is prime.

We are going to prove Euclid's theorem the way the great German mathematician Carl Gauss did. (We'll learn more about Gauss in Chapter 8.) First, we will use Equation 3.5, substituting $2^n - 1$ for both occurrences of $\sum_{i=0}^{n-1} 2^i$ in Euclid's theorem, to restate the theorem like this:

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

Next, we define $\sigma(n)$ to be the sum of the divisors of n. If the prime factorization of n is

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

then the set of all divisors consists of every possible combination of the prime divisors raised to every possible power up to a_i. For example, $24 = 2^3 \cdot 3^1$, so the divisors are $\{2^0 \cdot 3^0,\ 2^1 \cdot 3^0,\ 2^2 \cdot 3^0,\ 2^0 \cdot 3^1,\ 2^1 \cdot 3^1,\ 2^2 \cdot 3^1,\ 2^3 \cdot 3^1\}$. Their sum is

$$2^0 \cdot 3^0 + 2^1 \cdot 3^0 + 2^2 \cdot 3^0 + 2^0 \cdot 3^1 + 2^1 \cdot 3^1 + 2^2 \cdot 3^1 + 2^3 \cdot 3^1 = (2^0 + 2^1 + 2^2 + 2^3)(3^0 + 3^1)$$

That is, we can write the sum of the divisors for any number n as a product of sums:

$$\begin{aligned}
\sigma(n) &= \prod_{i=1}^{m}(1 + p_i + p_i^2 + \cdots + p_i^{a_i}) \\
&= \prod_{i=1}^{m} \frac{p_i - 1}{p_i - 1}(1 + p_i + p_i^2 + \cdots + p_i^{a_i}) \\
&= \prod_{i=1}^{m} \frac{(p_i - 1)(1 + p_i + p_i^2 + \cdots + p_i^{a_i})}{p_i - 1} \\
&= \prod_{i=1}^{m} \frac{p_i^{a_i+1} - 1}{p_i - 1} \qquad (3.6)
\end{aligned}$$

where the last line relies on using the difference of powers formula to simplify the numerator. (In this example, and for the rest of the book, when we use p as an integer variable in our proofs, we assume it's a prime, unless we say otherwise.)

Exercise 3.6. Prove that if n and m are *coprime* (have no common prime factors), then

$$\sigma(nm) = \sigma(n)\sigma(m)$$

(Another way to say this is that σ is a *multiplicative function*.)

We now define $\alpha(n)$, the *aliquot sum*, as follows:

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

In other words, the aliquot sum is the sum of all *proper* divisors of n—all the divisors except n itself.

Now we're ready for the proof of Theorem 3.3, also known as 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 is prime, and the theorem's condition is that $2^n - 1$ is prime, so $2^{n-1}(2^n - 1)$ is already a prime factorization of the form $n = p_1^{a_1} p_2^{a_2} \dots p_m^{a_m}$, where $m = 2, p_1 = 2, a_1 = n - 1, p_2 = 2^n - 1$, and $a_2 = 1$. Using the sum of divisors formula (Equation 3.6):

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

Then

$$\alpha(q) = \sigma(q) - q = 2q - q = q$$

That is, q is perfect. □

We can think of Euclid's theorem as saying that if a number has a certain form, then it is perfect. An interesting question is whether the converse is true: if a number is perfect, does it have the form $2^{n-1}(2^n - 1)$? In the 18th century, Euler proved that if a perfect number is even, then it has this form. He was not able to prove the more general result that *every* perfect number is of that form. Even today, this is an unsolved problem; we don't know if any odd perfect numbers exist.

Exercise 3.7. Prove that every even perfect number is a triangular number.

Exercise 3.8. Prove that the sum of the reciprocals of the divisors of a perfect number is always 2. Example:

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{6} = 2$$

3.5 The Pythagorean Program

For Pythagoreans, mathematics was not about abstract symbol manipulation, as it is often viewed today. Instead, it was the science of numbers and space—the two fundamental perceptible aspects of our reality. In addition to their focus on understanding *figurate* numbers (such as square, oblong, and triangular numbers), they believed that there was discrete structure to space. Their challenge, then, was to provide a way to ground geometry in numbers—essentially, to have a unified theory of mathematics based on positive integers.

To do this, they came up with the idea that one line segment could be "measured" by another:

Definition 3.1. A segment V is a **measure** of a segment A if and only if A can be represented as a finite concatenation of copies of V.

A measure must be small enough that an exact integral number of copies produces the desired segment; there are no "fractional" measures. Of course, different measures might be used for different segments. If one wanted to use the same measure for two segments, it had to be a *common measure*:

Definition 3.2. A segment V is a **common measure** of segments A and B if and only if it is a measure of both.

For any given situation, the Pythagoreans believed there is a common measure for all the objects of interest. Therefore, space could be represented discretely.

* * *

Since there could be many common measures, they also came up with the idea of the *greatest common measure*:

Definition 3.3. A segment V is the **greatest common measure** of A and B if it is greater than any other common measure of A and B.

The Pythagoreans also recognized several properties of greatest common measure (GCM), which we represent in modern notation as follows:

$$\mathrm{gcm}(a, a) = a \tag{3.7}$$

$$\mathrm{gcm}(a, b) = \mathrm{gcm}(a, a + b) \tag{3.8}$$

$$b < a \implies \mathrm{gcm}(a, b) = \mathrm{gcm}(a - b, b) \tag{3.9}$$

$$\mathrm{gcm}(a, b) = \mathrm{gcm}(b, a) \tag{3.10}$$

Using these properties, they came up with the most important procedure in Greek mathematics—perhaps in all mathematics: a way to compute the greatest common measure of two segments. The computational machinery of the Greeks consisted of ruler and compass operations on line segments. Using C++ notation, we might write the procedure like this, using `line_segment` as a type:

```
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);
}
```

This code makes use of the *trichotomy law*: the fact that if you have two values a and b of the same totally ordered type, then either $a = b$, $a < b$, or $a > b$.

Let's look at an example. What's `gcm(196, 42)`?

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

So we're done: gcm(196, 42) = 14.

Of course, when we say gcm(196, 42), we really mean GCM of segments with length 196 and 42, but for the examples in this chapter, we'll just use the integers as shorthand.

We're going to use versions of this algorithm for the next few chapters, so it's important to understand it and have a good feel for how it works. You may want to try computing a few more examples by hand to convince yourself.

3.6 A Fatal Flaw in the Program

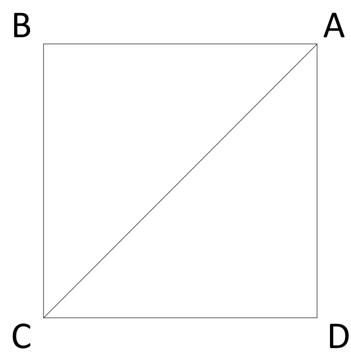
Greek mathematicians found that the *well-ordering principle*—the fact that any set of natural numbers has a smallest element—provided a powerful proof technique. To prove that something does not exist, prove that if it did exist, a smaller one would also exist.

Using this logic, the Pythagoreans discovered a proof that undermined their entire program.[4] We're going to use a 19th-century reconstruction of this proof by George Chrystal.

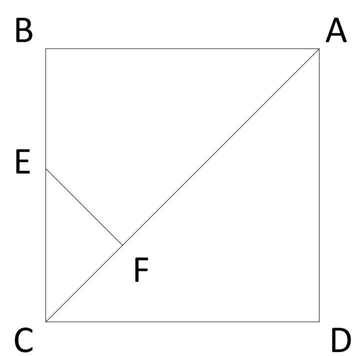
Theorem 3.4: *There is no segment that can measure both the side and the diagonal of a square.*

[4] We don't know if Pythagoras himself made this discovery, or one of his early followers.

Proof. Assume the contrary, that there were a segment that could measure both the side and the diagonal of some square.[5] Let us take the smallest such square for this segment:



Using a ruler and compass,[6] we can construct a segment $\overline{AF}$ with the same length as $\overline{AB}$, and then create a segment starting at F and perpendicular to $\overline{AC}$.

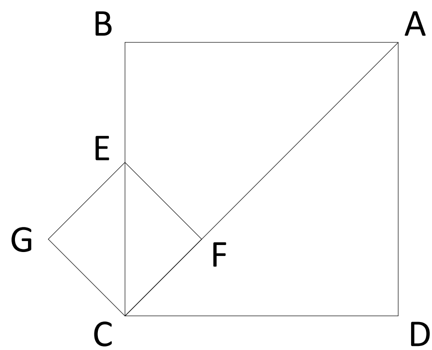


$$\overline{AB} = \overline{AF} \quad \wedge \quad \overline{AC} \perp \overline{EF}$$

Now we construct two more perpendicular segments, $\overline{CG}$ and $\overline{EG}$:

[5]This is an example of proof by contradiction. For more about this proof technique, see Appendix B.1.

[6]Although modern readers may think of a ruler as being used to measure distances, for Euclid it was only a way to draw straight lines. For this reason, some people prefer the term *straightedge* to describe Euclid's instrument. Similarly, although a modern compass can be fixed to measure equal distances, Euclid's compass was used only to draw circles with a given radius; it was collapsible, so it did not preserve distances once lifted.

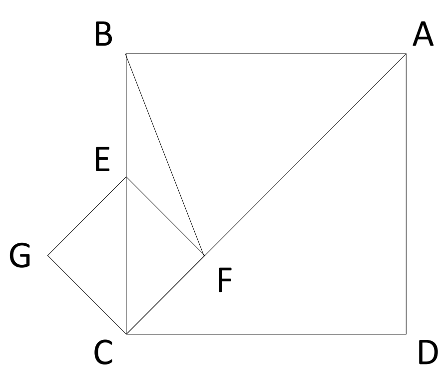


$$\overline{AC} \perp \overline{CG} \quad \wedge \quad \overline{EG} \perp \overline{EF}$$

We know that $\angle CFE = 90°$ (by construction) and that $\angle ECF = 45°$ (since it's the same as $\angle BCA$, which is the angle formed by the diagonal of a square, and therefore is half of $90°$). We also know that the three angles of a triangle sum to $180°$. Therefore

$$\angle CEF = 180° - \angle CFE - \angle ECF = 180° - 90° - 45° = 45°$$

So $\angle CEF = \angle ECF$, which means CEF is an isosceles triangle, so the sides opposite equal angles are equal—that is, $\overline{CF} = \overline{EF}$. Finally, we add one more segment $\overline{BF}$:



Triangle ABF is also isosceles, with $\angle ABF = \angle AFB$, since we constructed $\overline{AB} = \overline{AF}$. And $\angle ABC = \angle AFE$, since both were constructed with perpendiculars. So

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

Now, we know $\overline{AC}$ is measurable since that's part of our premise, and we know $\overline{AF}$ is measurable, since it's the same as $\overline{AB}$, which is also measurable by our premise. So their difference $\overline{CF} = \overline{AC} - \overline{AF}$ is also measurable. Since we just showed that $\triangle CEF$ and $\triangle BEF$ are both isosceles,

$$\overline{CF} = \overline{EF} = \overline{BE}$$

we know $\overline{BC}$ is measurable, again by our premise, and we've just shown that $\overline{CF}$, and therefore $\overline{BE}$, is measurable. So $\overline{EC} = \overline{BC} - \overline{BE}$ is measurable.

We now have a smaller square whose side ($\overline{EF}$) and diagonal ($\overline{EC}$) are both measurable by our common unit. But our original square was chosen to be the smallest for which the relationship held—a contradiction. So our original assumption was wrong, and *there is no segment that can measure both the side and the diagonal of a square.* If you try to find one, you'll be at it forever—our `line_segment_gcm(a, b)` procedure will not terminate. □

To put it another way, the ratio of the diagonal and the side of a square cannot be expressed as a rational number (the ratio of two integers). Today we would say that with this proof, the Pythagoreans had discovered irrational numbers, and specifically that $\sqrt{2}$ is irrational.

The discovery of irrational numbers was unbelievably shocking. It undermined the Pythagoreans's entire program; it meant that geometry could not be grounded in numbers. So they did what many organizations do when faced with bad news: they swore everyone to secrecy. When one of the order leaked the story, legend has it that the gods punished him by sinking the ship carrying him, drowning all on board.

* * *

Eventually, Pythagoras' followers came up with a new strategy. If they couldn't unify mathematics on a foundation of numbers, they would unify it on a foundation of geometry. This was the origin of the ruler-and-compass constructions still used today to teach geometry; no numbers are used or needed.

Later mathematicians came up with an alternate, number-theoretic proof of the irrationality of $\sqrt{2}$. One version was included as proposition 117 in some editions of Book X of Euclid's *Elements*. While the proof predates Euclid, it was added to *Elements* some time after the book's original publication. In any case, it is an important proof:

Theorem 3.5: *$\sqrt{2}$ is irrational.*

Proof. Assume $\sqrt{2}$ is rational. Then it can be expressed as the ratio of two integers m and n, where m/n is irreducible:

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

m^2 is even, which means that m is also even,[7] so we can write it as 2 times some number u, substitute the result into the preceding equation, and do a bit more algebraic manipulation:

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

n^2 is even, which means that n is also even. But if m and n are both even, then m/n is not irreducible—a contradiction. So our assumption is false; there is no way to represent $\sqrt{2}$ as the ratio of two integers. □

3.7 Thoughts on the Chapter

The ancient Greeks' fascination with "shapes" of numbers and other properties such as prime and perfect were the basis of the mathematical field of number theory. Some of the algorithms they used, such as the Sieve of Eratosthenes, are still very elegant, though we saw how to improve their efficiency further by using some modern optimization techniques.

* * *

Toward the end of the chapter, we saw two different proofs that $\sqrt{2}$ is irrational, one geometric and one algebraic. The fact that we have two completely different proofs of the same result is good. It is actually essential for mathematicians to look for multiple proofs of the same mathematical fact, since it increases their confidence in the result. For example, Gauss spent much of his career coming up with multiple proofs for one important theorem, the quadratic reciprocity law.

[7] This is easily shown: The product of two odd numbers is an odd number, so if m were not even, m^2 could not be even. Euclid proved this and many other results about odd and even numbers earlier in *Elements*.

The discovery of irrational numbers emerged from the Pythagoreans' attempts to represent continuous reality with discrete numbers. While at first glance we might think they were naive to believe that they could accomplish this, computer scientists do the same thing today—we approximate the real world with binary numbers. In fact, the tension between continuous and discrete has remained a central theme in mathematics through the present day, and will probably be with us forever. But rather than being a problem, this tension has actually been the source of great progress and revolutionary insights.

This page intentionally left blank

This page intentionally left blank

Index

$*$ (operator), mathematical convention for, 115
$+$ (plus sign), mathematical convention for, 115
α. *See* Aliquot sum
φ. *See* Euler totient function
σ (sum of divisors) formula, 31

A

Abelian group, 86, 108, 153
Abstract algebra
- birth of, 85, 140–145
- Euclidean domains, 150–151, 153
- fields, 151–153
- groups, 85–88, 92–95, 108, 152
- ideals, 226–228
- modules, 151, 154
- monoids, 89, 108–109, 152, 154
- and programming, 2, 141, 249
- principal ideals, 227–228
- rings, 142–145, 153
- semigroups, 90–91, 108–109, 152
- semirings, 145–149, 153
- vector spaces, 152, 154

Abstraction
- Aristotle, 177, 180, 196
- in mathematics, 84, 85–109
- and programming, 2, 5, 249

Academy (Plato's), 41–44, 178
Addition
- associativity of, 9, 156, 174
- commutativity of, 155–156, 174–175
- definition, 173

Addition chains, 11
Additive groups, 86
Additive monoids, 89, 109
Additive semigroups, 90, 109
Address, 181
Adleman, Len, 239
`advance`, 190
Agrawal, Manindra, 244
Ahmes, 8–9, 57
Ahmes algorithm. *See* Egyptian multiplication, Egyptian division
AKS primality test, 244–245
Alexander the Great, 43, 178–179
Alexandria, 43–44
Algebraic integers, 140
Algebraic structures, 85. *See also* Abstract algebra
Algorithms
- in ancient Egypt, 7–11
- definition, 7
- domain or setting, 150
- first recorded, 8
- generalizing, 111, 119–123, 126–127, 151
- history of, 7–11
- in-place, 215–216
- memory adaptive, 216–217
- polylog space, 215–216
- space complexity, 215–216
- performance in practice, 211

Aliases, 272
Aliquot sum, 31
Amicable numbers, 63–64
***Analytical Mechanics*, 99**
APL, 124
***Apology*, 43**
Archimedes
  on acquiring mathematical knowledge, 176
  axiom of, 47
  place in history, 50
Aristophanes, 42
Aristotle, 17, 177–180
Aristoxenus, 17
***Arithmétique*, 132**
***The Art of Computer Programming*, 9**
Aryabhata, 51
***Aryabhatiya*, 51**
Assertions, 269
Associative binary operation (○), 108
  in groups, 85–86
  in monoids, 89
  in semigroups, 90
Associativity axiom, semigroups, 91
Associativity of addition, 9, 113
  definition, 174
  visual proof, 156
Associativity of multiplication, visual proof, 157
Asymmetric keys, 238
Athens, 41–43
Automorphism, 104
Averroes. *See* Ibn Rushd
Axiom of Archimedes, 47
Axiomatic method, 161–162
Axioms
  definition, 163
  Euclid's, 162–163
  Hilbert's, 167
  Peano's, 170–171

B

Bachet de Méziriac, Claude Gaspar, 67, 235
  profile, 225–226
Backus, John, 124
Bacon, Roger, 1, 249
Bartels, Martin, 165
Bernoulli, Johann, 69
Bézout's identity, 225–229
Bidirectional iterators, 185
Binary search, 191–196. *See also* Partition points
Binary search lemma, 194–195
Bletchley Park, 238
Bolyai, Farkas, 166
Bolyai, János, 166
Bolzano-Cauchy Theorem. *See* IVT (Intermediate Value Theorem)
Boolean semirings, 148
Bounded ranges, 189, 203–204
`bsearch`, 192

C

C++, 3, 265–273
C++11, 57, 187, 195, 265, 272–273
***The C++ Programming Language*, 265, 270**
C++ Standard Template Library. *See* STL
Caesar cipher, 237
Cancellation
  Cancellation Law, 74–75
  definition, 72–73
  inverse numbers, 73
  and modular arithmetic, 72–76
  Self-Canceling Law, 75–76
Cancellation Law, 74–75
Carmichael numbers, 242
Cartesian coordinates, 131, 138
Cataldi, Peter, 64
Categorical theories
  vs. STL,104
  definition, 104
  examples of, 104–106
Category dispatch, 188, 190, 196, 213
Cayley's Theorem, 198
Chinese mathematics, 51
Chrystal, George, 34
Cicero, 50
Ciphertext, 238

Closed ranges, 188
Clouds, 42
Cocks, Clifford, 240
Codes, definition, 237
***Cogitata Physico Mathematica*, 64**
Colossus machine, 238
Common Lisp, 116, 124, 190
Common measure of segments, 33
Common notions, Euclid's axiomatic method, 162
Commutative algebra, rings, 143–144
Commutativity of addition, 155–156, 174–175
Commutativity of multiplication, visual proof, 156
Commutativity of powers, semigroups, 91
Compile-time dispatch. *See* **Category dispatch**
Completeness, law of, 203–204
Completeness, theories, 102
Complex numbers, 137–138
Composite numbers. *See also* **Prime numbers**
- definition, 21
- distinguishing from prime, 240–245

Concepts
- and abstract algebra, 141
- definition, 181
- choosing, 250
- examples, 116–117, 181
- naming conventions, 183
- overview, 181–184, 266–267
- `Regular`, 183–184
- requirements on types, 24, 182
- `Semiregular`, 184
- type attributes, 182–183
- type functions, 182–183

Consistency, theories, 102, 104
Constructivists, 229
Contradiction, proof by, 35, 261–262
Contrapositive, 259
Coprime, 31, 78, 80–81, 246–247
Cosets, 97. *See also* **Lagrange's Theorem**
Counted ranges, 189, 203–204
Cryptanalysis, 237–238
Cryptography, 233–234, 237
Cryptology
- asymmetric keys, 238
- Bletchley Park, 238
- Caesar cipher, 237
- ciphertext , 238
- codes, definition, 237
- Colossus machine, 238
- cryptanalysis, 237–238
- cryptography, 237
- cryptosystems, 238
- Enigma machine, 238
- keys, 238
- Lorenz machine, 238
- plaintext, 238
- public-key cryptosystems, 239–240
- RSA algorithm, 239–240, 245–247
- symmetric keys, 238
- trapdoor one-way functions, 239

Cryptosystems, 238
Cycles, of permutations, 200, 207–211
Cyclic groups, 96, 109
- generator, 96

Cyclic subgroups, 96

D

Datum, 180
Decimal fractions, 129–131
Declaration syntax, 267
Dedekind, Richard, 140, 171
Degree of polynomials, 133
Dereferencing, iterators, 184–185
Descartes, René, 64, 131
Difference of powers formula, 30
Difference type, iterators, 187
`difference_type` iterator trait, 187
***Differential Calculus*, 70**
Diffie, Whitfield, 239
Diophantus, 67, 225
Dirichlet, Peter Gustav Lejeune, 41, 139–140, 156
Dirichlet principle. *See* **Pigeonhole principle**
***Disme: The Art of Tenths*, 129–131**

Disquisitiones Arithmeticae ("Investigations of Arithmetic"), 136–137
`distance`, 186-188
`divides`, 240
Dividing polynomials, 133
Domain of algorithm, 150
Domain of definition, 113
Doubly linked lists, 185

E

Egyptian division, 57
Egyptian multiplication, 8–11
  requirements, 111–118
  generalizing to power, 120
***Elements* (of Euclid), 2, 21, 43–45, 161–163**
  Proposition [VII, 30], 70
  Proposition [VII, 32], 21
  Proposition [IX, 36], 29, 31–32
  Proposition [X, 2], 45
  Proposition [X, 3], 45–46
  Proposition [X, 117], 37
***Elements of Programming*, 3, 113–114, 183, 185, 208**
Enigma machine, 238
Equational reasoning, 114
Equivalence, 114
Eratosthenes, 22
Euclid. ***See also Elements***
  the axiomatic method, 161–163
  GCM (greatest common measure) algorithm, 45–49
  incommensurable quantities, 45–49
  on number theory, 21
  profile, 44–45
Euclidean domains (ED), 150–151, 153
Euclidean geometry
  alternatives to, 164–167
  fifth postulate, 163–164
  vs. hyperbolic geometry, 164–167
  vs. non-Euclidean, 166–167
Euclidean rings. ***See*** **Euclidean domains (ED)**
Euclid's algorithm, 45–47
Euler, Leonhard, 84, 85
  Euler's theorem, 79–83
  and Lagrange, 99
  perfect numbers, 32, 63–64
  prime numbers, 63, 68
  profile, 69–70
Euler totient function, 80, 245
Euler's Theorem, 79–83, 246
  proof using Lagrange's Theorem, 101
Even and odd numbers, 9–10, 117
  in GCD, 219–220, 224, 234
Existence of zero axiom, 172
Extended GCD algorithm, 229–235, 245, 247
`extended_gcd`, 233

F

Fast-multiplication algorithm. ***See*** **Egyptian multiplication**
Fermat, Pierre de, 63, 65–69
  profile of, 67–68
  proofs, 65–66, 68
Fermat primes, 63–68, 137
Fermat's Last Theorem, 67
Fermat's Little Theorem
  converse of, 77–79
  description, 69
  non-invertibility lemma, 79
  proof by Lagrange's Theorem, 101
  proof, 77
  testing for prime numbers, 241–242
  restatement using modular arithmetic, 84
Fermat test, 241–242
`fermat_test`, 242
Fibonacci. ***See*** **Leonardo Pisano**
Fibonacci numbers, computing, 124–127
Fibonacci sequence, 58–59
Fields
  characteristic of, 151
  definition, 151, 153
  extensions, 151
  prime, 151, 154

Fifth postulate of Euclidean geometry, 163–164
Figurate numbers
gnomons, 20
oblong numbers, 19
overview, 17, 19–20
triangular numbers, 19
square numbers, 20
`find_if`, 190–191
`find_if_n`, 191
Finite axiomatizability of theories, 102
Flowers, Tommy, 238
Floyd, Robert, 58
Formalist philosophy of mathematics, 167–169
***Formulario Mathematico*, 170–172**
Forward iterators, 185
FP, 124
Function objects, 123–124, 268, 270
Functors. *See* Function objects

G

Galois, Évariste
discovery of groups, 85–88
profile of, 88–89
Gauss, Carl Friedrich, 31, 72, 136–140, 166, 240
profile of, 136–137
Gaussian integers, 138–139, 224
GCD (greatest common divisor)
applications of, 234
of polynomials, 134
description, 59
computing, 59
Euclid's algorithm, 45–46
extended GCD, 229–235, 245, 247
historical milestones, 222
and rational arithmetic, 234
and ring structures, 225–229
rotation algorithms, 234
Stein's algorithm, 219–225
symbolic integration, 234
validating, 59–60
`gcd`, 150, 230
GCM (greatest common measure) 33, 41
Euclid's algorithm, 45–49
properties, 33
Generator elements in subgroups, 96
Generic programming
in C++, 265–266, 270
concepts, 181
essence, 127, 249–250
history, 124, 134, 141, 180
and mathematics, 84
overview, 1–2, 5
`get_temporary_buffer`, 217
Gnomons, 20
Gödel, Kurt, 169
Göttingen, University of
Carl Gauss, 136–140
David Hilbert, 168–169
Emmy Noether, 140–145
profile, 135–136
Granville, Andrew, 244
Grassman, Hermann, 171
Greatest common divisor (GCD). *See* GCD (greatest common divisor)
Greatest common measure (GCM). *See* GCM (greatest common measure)
Gries, David, 205
Gries-Mills algorithm, 204–208
Groups
abelian, 86, 108, 153
additive, 86
binary operations, 86
cyclic, 96, 109
definition, 85
discovery of, 85
examples of, 86–88
identity elements, 86
inverse operations, 86
Klein group, 106
order of elements, 94
summary description, 108, 152

Groups *(continues)*
symmetric, 198
theorems about, 92–95

H

`half`, 118
Heath, Thomas, 9, 45
Hegel, G.W.F., 111
Hellman, Martin, 239
Hilbert, David, 141, 167–169, 229
profile, 168–169
Hilbert spaces, 168–169
Hilbert's problems, 169
Hilbert's program, 169
***History of Algebra*, 129**
Horner's rule, 132
Hyperbolic geometry, 164–167

I

Ibn Rushd, 180
Ideals. *See also* **Rings**
definition, 226
ideals in Euclidean domains lemma, 227
linear combination ideal lemma, 227
PID (principal ideal domains), 228
principal ideals, 227–228
principal elements, 227
Ideals in Euclidean domains lemma, 227
Identity element, 108–109, 121
in groups, 86
in monoids, 89
in rings, 143
`identity_element`, 123, 241
Immutable objects, 181
Impossibility of infinite descent, 21
Inclusion-exclusion principle, 82–83
Incommensurable quantities, 45–49
Independence, theories, 102
Indian mathematics, 51
Induction, proof by, 262–263
Induction axiom, 21, 170, 172–173
Inman, Bobby Ray, 240
Inner product of two vectors, 145–146
In-place algorithms, 215–216
Input iterators, 185
***Integral Calculus*, 70**
Integral domains, 145, 153
Interface refinement, law of, 215
Interfaces, designing, 215
Interlingua, 171
Intermediate Value Theorem (IVT), 131, 192
***Introduction to Analysis of the Infinite*, 70**
***Introduction to Arithmetic*, 10, 19**
Intuitionist philosophy of mathematics, 229
Inverse numbers, 73
`inverse_operation`, 123
Inverse operation, 86, 119, 121
in groups, 86
Invertibility lemma, 229
Invertibility of successor axiom, 173
Invertible elements. *See* **Units**
Irrational numbers, 38–39
`is_prime`, 241
Isomorphism, models, 103–104
Iterator categories
bidirectional, 185
forward, 185
input, 185
output, 186
random-access, 185
Iterator traits, 187
`iterator_category` iterator trait, 187
Iterators
in arrays, 185
bidirectional, 185
definition, 184
dereferencing, 184–185
difference type, 187
finding the distance between, 186–187
forward, 185
input, 185
in noncontiguous data segments, 186
linked, 186
output, 186
overview, 184–185
random access, 185

- segmented, 186
- successors, 184

Iverson, Kenneth, 124
IVT (Intermediate Value Theorem), 131, 192

J

Jefferson, Thomas, 44, 130

K

Kapur, Deepak, 124
Kayal, Neeraj, 244
Keys, cryptography, 238
Khayyam, Omar, 164
Kleene, Stephen, 115–116
Klein, Felix, 106–107, 141
Klein group, 106–107
Knuth, Donald E., 9, 58, 197
Kovalevskaya, Sofia, 141

L

Lagrange, Joseph-Louis, 99–100, 192
Lagrange's Theorem, 97–99, 100–101
Lambda expressions, 195, 272–273
Laplace, Pierre-Simon, 70
`largest_doubling`, 54
***Latine sine Flexione*, 171**
Law of completeness, 203–204
Law of interface refinement, 215
Law of separating types, 202–203
Law of useful return, 57–58, 201–202, 213
***Lectures on Number Theory* (*Vorlesungen über Zahlentheorie*), 140**
Legendre, Adrien-Marie, 155
Lehmer, D. H., 192
Leonardo Pisano
- Fibonacci sequence, 58–59
- introduction of zero, 52
- profile, 52–53

***Letters to a German Princess*, 70**
***Liber Abaci*, 52**
***Liber Quadratorum*, 52**
Library of Alexandra, 43
Lincoln, Abraham, 44
Linear algebra
- inner product, 145–146
- matrix-matrix product, 146
- matrix-vector product, 146
- review, 145–147

Linear combination ideal lemma, 227
Linear recurrence functions, 127
Linear recurrence sequences, 127
Linear search, 190–191
Linked iterators, 186
Liu, Hui, 51
Lobachevsky, Nikolai, 164–166
Lorenz machine, 238
`lower_bound`, 195–196
Lyceum, 179

M

Magmas, 91, 108
`mark_sieve`, 24
Math notation in this book, 257–259
Matrix multiplication, 145–147
Matrix-matrix product, 146
Matrix-vector product, 146
Mauchly, John, 192
McJones, Paul, 3
Measure of a segment, 33
Memory-adaptive algorithms, 216–217
***Meno*, 43**
Mersenne, Marin, 64–65
Mersenne primes, 63–68
***Metaphysics*, 179**
Miller-Rabin test, 243–245
`miller_rabin_test`, 243
Mills, Harlan, 205
Models. *See also* Theories
- definition, 103
- isomorphism, 103–104

***Modern Algebra*, 142**
Modular arithmetic, 72–74, 83–84
- Fermat's Little Theorem, 83–84
- Wilson's Theorem, 83

Modules, definition, 151, 154
`modulo_multiply`, 241

Monoids. *See also* **Groups**
additive, 89, 109, 154
definition, 89
examples of, 89
multiplicative, 89, 154
summary description, 108, 152
Mouseion, 43
Multiplication
definition, 8, 173–174
Egyptian, 8–11
Russian Peasant Algorithm. 9
Multiplicative functions, 31
`multiplicative_inverse`, 121, 247
`multiplicative_inverse_fermat`, 241
Multiplicative monoids, 89
Multiplicative semigroups, 90
Multiply-accumulate function, 11–14
Musser, David R., 124
Mutable objects, 181

N

Naming conventions, concepts, 183
Naming principle, 115–116
Natural numbers, 147, 170, 172, 175, 258
***Nicomachean Ethics*, 179**
Nicomachus of Gerasa, 10, 19
***Nine Chapters on the Mathematical Art*, 51**
Noether, Emmy, 129, 140–145
profile, 141–142
Non-categorical theories, 106–107
Noncommutative additive monoids, 119
Noncommutative additive semigroups, 115
Noncommutative algebra, rings, 143–144
Nonconstructive proofs, 229
Noncontiguous data segments, iterators, 186
Non-Euclidean geometry, 164–167
Non-invertibility lemma, 79
Notation in this book, 257–259
Number line, 131
Number of assignments theorem, 200–201
Number systems, ancient Egypt, 8
Number theory 2 , 41, 43
in ancient Greece, 17–39
Bezout's identity, 225–229
Euler's Theorem, 79–83, 101
Fermat's Little Theorem 69–78, 101
figurate numbers, 17–20, 33
Gauss, 136–137
and GCD, 140
Liber Quadratorum, 53
modular arithmetic, 72–74
perfect numbers, 28–32
primality testing 240–245
prime numbers, 21–28
17th and 18th century, 63–72, 74–84
sieve of Eratosthenes, 22–23
Wilson's Theorem, 76, 83

O

Object types, definition, 181
Objects
definition, 180
immutable, 181
mutable, 181
remote parts, 181
unrestricted, 181
Oblong numbers, 19
Octonions, 151
`odd`, 118
Odd numbers. *See* **Even and odd numbers**
One-to-one correspondence, 92
Open ranges, 188
"Operators and Algebraic Structures," 124
Order of group elements, 94
***Organon*, 180**
Output iterators, 186

P

Palindromic primes, 28
Parallel postulate. *See* **Fifth postulate of Euclidean geometry**
Partition points, 193
`partition_point`, 194

partition_point_n, 193
Peano, Giuseppe, 169–175
  profile, 171–172
Peano arithmetic, 170–171, 173–175
Peano axioms, 170–173
Peano curve, 171
Perfect numbers
  in ancient Greece, 28–32, 38
  definition, 28–29
  mathematicians' interest in, 63
Permutation of remainders lemma, 71–72
Permutations, 197–201
Phaedo, 43
Philo of Alexandria, 7
PID (principal ideal domains), 228
Pigeonhole principle, 95, 263
Pisano, Leonardo *See* Leonardo Pisano
Plaintext, 238
Plato, 41–43, 177–179
  profile, 42–43
Platonic Questions, 20
Platonic solids, 41, 44
Playfair's axiom, 163
Plus sign (+), mathematical convention for, 115
Plutarch, 20
Poincaré, Jules Henri, 85, 229–230, 248
  profile, 229–230
pointer iterator trait, 187
Politics, 179
Polylog space, 215–216
Polynomials
  computing GCD for, 134
  degree of, definition, 133
  division with remainder, 133
  history of, 132–135
  Horner's rule, 132
  treating as numbers, 133–135
polynomial_value, 132
Population count, 10
Postconditions, 269
Postulates, Euclid's axiomatic method, 162, 163
Power algorithm, 119–123, 249
  computing Fibonacci numbers, 126
  computing linear recurrence, 127
  use in cryptology, 241–243, 246
  use in graph applications, 148–149
power_accumulate_semigroup, 121
power_group, 123
power_monoid, 122
power_semigroup, 122
Primality testing, 240–245
Prime factorization, 29, 31–32, 65, 136, 139–140
Prime fields, 151, 154
Prime numbers
  in ancient Greece, 21–28
  definition, 21
  distinguishing from composite, 240–245
  Fermat primes, 63–68
  finding. *See* sieve of Eratosthenes
  infinite number of, 21
  Mersenne primes, 63–68
  primality testing, 240–245
Principal element, 227
Principal ideal domains (PID), 228
Principal ideals, 227–228
Problèmes Plaisants, 225–226
Proof
  by contradiction, 35, 261–262
  definition, 158–159
  by induction, 262–263
  nonconstructive, 229
  pigeonhole principle, 95, 263
  visual, 155–159
Proper divisor, 32
Ptolemy, 164
Public-key cryptosystems, 239–240
Pythagoras, 17
  profile, 18–19
Pythagorean program, 33–38

Pythagorean Theorem, 44
Pythagorean triples, 50–51

Q

Quadrivium, 18
Quaternions, 151
Quotient, 55–57, 150, 153, 202
 for polynomials, 133
`quotient_remainder`, 57

R

Random-access iterators, 185
Ranges
 bounded, 189, 203–204
 closed, 188
 counted, 189, 203–204
 definition, 188
 open, 188
 overview, 188–189
 partition points, 193
 semi-open, 188
 swapping, 201–204
Rational arithmetic, GCD applications, 234
Rational numbers, 151, 258
Real numbers, 131, 258
`reciprocal`, 124
Recreational mathematics, 225–226
Recursive remainder lemma, 48–49
Reduction algorithm, 124
`reference` iterator trait, 187
Regius, Hudalricus, 64
Regular concepts, 183–184
Regular functions, 183
`Regular` types, 114
Rejewski, Marian, 238
Remainder, 47–49, 53–55, 57–59, 150, 153, 222
 Floyd-Knuth algorithm, 58
 permutation of remainders, 71–72
 in modular arithmetic, 73–75
 of Gaussian integers, 138–139
 of polynomials, 133–134
`remainder`, 54–55
`remainder_fibonacci`, 58
Remote parts of objects, 181
Requirements on algorithm, 111–119
`reverse`, 212–215
`reverse_copy`, 216
`reverse_n`, 214
`reverse_n_adaptive`, 217
`reverse_n_with_buffer`, 216
`reverse_recursive`, 214
Reverse permutation, 201, 212–215
Rewriting code, 14–15
Rhind Mathematical Papyrus, 8, 57
Rings. *See also* Ideals; Semirings
 definition, 142–143
 and the GCD, 225–229
 integral domains, 145
 summary description, 153
 unitary, 143
 units, 144
 zero divisors, 145
Rivest, Ron, 239
Rotate algorithms, 204–213
`rotate`, 207, 210, 213
`rotate_cycle_from`, 208
`rotate_transform`, 210
`rotate_unguarded`, 206
Rotation, 204–207
Rotation algorithms, GCD applications, 234
RSA algorithm, 239–240, 245–247
Russell, Bertrand, 171
Russian Peasant Algorithm, 9. *See also* Egyptian multiplication

S

Saccheri, Giovanni Girolamo, 164
Saxena, Nitin, 244
Scheme, 116
The School of Athens, 177–178
Searches
 binary, 191–196
 linear, 190–191
Segmented iterators, 186
Self-Canceling Law, 75–76
Semantic requirements for generic algorithms, 113

Semigroups. ***See also*** **Groups**
additive, 90, 109
associativity axiom, 91
commutativity of powers, 91
definition, 90, 109
examples, 90
multiplication algorithm, 115
multiplicative, 90
summary description, 108, 152
Semi-open ranges, 188
`Semiregular` concepts, 184
Semirings. ***See also*** **Rings**
Boolean, 148
description, 145–147
matrix multiplication, 146
shortest path, 148–149
summary description, 153
tracing social networks, 147–148
transitive closures, 147–148
tropical, 149
weak, 147
Separating types, law of, 202–203
Setting of algorithm, 150
Shamir, Adi, 239
Shortest path, finding, 148–149
Sieve of Eratosthenes, 22–23
implementation 23–28
`sift`, 27
`smallest_divisor`, 240
Social network connections, tracing, 147–148
Socrates, 42
Socratic method, 42
Sophists, 42
Space complexity, 215–216
Square root of 2, an irrational number, 37–38
Standard Template Library. ***See*** **STL**
Stein, Josef, 219–222
Stein's algorithm, 219–225
`stein_gcd`, 220
Stepanov, Alexander A., 3, 124
Stevin, Simon, 129–135, 192
profile, 130–131
STL (Standard Template Library)
algorithms, 195–196, 215, 217
application of generic programming, 1, 186
containers, 190–191
conventions, 24
non-categorical, 104
Strength reduction, 26
Stroustrup, Bjarne, 265
Subgroups. ***See also*** **Groups**
cyclic, 96
definition, 95, 109
generator elements, 96
trivial, 95
Successors, 170, 184
Sum of odd powers formula, 30
`swap`, 199
`swap_ranges`, 201–203
Symbolic integration, GCD applications, 234
Symmetric groups, 198
Symmetric keys, 238
***Symposium*, 43**
Syntactic requirements for generic algorithms, 113

T

Tail-recursive functions, 12–14
Template functions, 265–266
Thales of Miletus, 18, 159–161
Thales' Theorem, 160–161
Theories. ***See also*** **Models**
categorical, 104–106
characteristics of, 102
completeness, 102
consistency, 102, 104
definition, 102
determining truth of, 167
finite axiomatizability, 102
independence, 102
non-categorical, 106–107
univalent, 104
Totality of successor axiom, 172
Totient of an integer, 80
***A Tour of C++*, 265**
Transfinite ordinals, 172–173

Transformation group, 92
Transitive closures, finding, 147–148
Transposition lemma, 199
Transpositions, 197, 199–201
Trapdoor one-way functions, 239
Triangular numbers, 19
Trichotomy Law, 34
Trip count, 204, 213
Trivial cycles, 200, 208
Trivial subgroups, 95
Tropical semirings, 149
Turing, Alan, 169, 238
Tusculan Disputations, 50
Type attributes, 182–183
Type dispatch. *See* Category dispatch
Type functions, 182–183

U

Unitary rings, 143
Units, rings, 144
Univalent theories, 104
Univariate polynomials. *See* Polynomials
University of Göttingen. *See* Göttingen, University of
Unreachable numbers, 172–173
Unrestricted objects, 181
`upper_bound`, 195
Useful return, law of, 57–58, 201–202, 213

V

Value types, definition, 180
Values, definition, 180
`value_type` iterator trait, 187
van der Waerden, Bartel, 129, 142
Veblen, Oswald, 104
`vector` container, 116,
Vector space, 152, 154
Visual proofs, 155–159
Vorlesungen über Zahlentheorie (*Lectures on Number Theory*), 140

W

Waring, Edward, 76
Weak semirings, 147
Weilert, Andre, 224
Well-ordering principle, 34
Whitehead, Alfred North, 43
Wiles, Andrew, 67
Wilson, John, 76
Wilson's Theorem
- description, 76
- using modular arithmetic, 83

Witnesses, primality testing, 242

Z

Zero
- in Egyptian number system, 8
- introduction of, 52
- origins of, 51–53

Zero divisors, rings, 145, 154