4.10 Application: Algorithms

Begin at the beginning . . . and go on till you come to the end: then stop.
—Lewis Carroll, *Alice's Adventures in Wonderland*, 1865

Hulton Archive/Getty Images

Lady Lovelace
(1815–1852)

In this section we will show how the number theory facts developed in this chapter form the basis for some useful computer algorithms.

The word *algorithm* refers to a step-by-step method for performing some action. Some examples of algorithms in everyday life are food preparation recipes, directions for assembling equipment or hobby kits, sewing pattern instructions, and instructions for filling out income tax forms. Part of elementary school mathematics is devoted to learning algorithms for doing arithmetic such as multidigit addition and subtraction, multidigit (long) multiplication, and long division.

The idea of a computer algorithm is credited to Ada Augusta, Countess of Lovelace. Trained as a mathematician, she became very interested in Charles Babbage's design for an "Analytical Engine," a machine similar in concept to a modern computer. Lady Lovelace extended Babbage's explorations of how such a machine would operate, recognizing that its importance lay "in the possibility of using a given sequence of instructions repeatedly, the number of times being either preassigned or dependent on the results of the computation." This is the essence of a modern computer algorithm.

An Algorithmic Language

The algorithmic language used in this book is a kind of pseudocode, combining elements of Python, C, C^{++}, and Java, and ordinary, but fairly precise, English. We will use some of the formal constructs of computer languages—such as assignment statements, loops, and so forth—but we will ignore the more technical details, such as the requirement for explicit end-of-statement delimiters, the range of integer values available on a particular installation, and so forth. The algorithms presented in this text are intended to be precise enough to be easily translated into virtually any high-level computer language.

In high-level computer languages, the term **variable** is used to refer to a specific storage location in a computer's memory. To say that the variable x has the value 3 means that the memory location corresponding to x contains the number 3. A given storage location can hold only one value at a time. So if a variable is given a new value during program execution, then the old value is erased. The **data type** of a variable indicates the set in which the variable takes its values, whether the set of integers, or real numbers, or character strings, or the set $\{0, 1\}$ (for a Boolean variable), and so forth.

An **assignment statement** gives a value to a variable. It has the form

$$x := e,$$

where x is a variable and e is an expression. This is read "x is assigned the value e" or "let x be e." When an assignment statement is executed, the expression e is evaluated (using the current values of all the variables in the expression), and then its value is placed in the memory location corresponding to x (replacing any previous contents of this location).

Ordinarily, algorithm statements are executed one after another in the order in which they are written. **Conditional statements** allow this natural order to be overridden by using the current values of program variables to determine which algorithm statement will be executed next. Conditional statements are denoted either

a. **if** (*condition*) **then** s_1
else s_2

or

b. **if** (*condition*) **then** s_1

where *condition* is a predicate involving algorithm variables and where s_1 and s_2 are algorithm statements or groups of algorithm statements. We generally use indentation to indicate that statements belong together as a unit. When ambiguity is possible, however, we may explicitly bind a group of statements together into a unit by preceding the group with the word **do** and following it with the words **end do**.

Execution of an **if-then-else** statement occurs as follows:

1. The *condition* is evaluated by substituting the current values of all algorithm variables appearing in it and evaluating the truth or falsity of the resulting statement.
2. If *condition* is true, then s_1 is executed and execution moves to the next algorithm statement following the **if-then-else** statement.
3. If *condition* is false, then s_2 is executed and execution moves to the next algorithm statement following the **if-then-else** statement.

Execution of an **if-then** statement is similar to execution of an **if-then-else** statement, except that if *condition* is false, execution passes immediately to the next algorithm statement following the **if-then** statement.

Often *condition* is called a **guard** because it is stationed before s_1 and s_2 and restricts access to them.

Example 4.10.1 **Execution of if-then-else and if-then Statements**

Consider the following algorithm segments:

a. **if** $x > 2$
then $y := x + 1$
else do $x := x - 1$
$y := 3 \cdot x$ **end do**

b. $y := 0$
if $x > 2$ **then** $y := 2^x$

What is the value of y after execution of these segments for the following values of x?
i. $x = 5$ ii. $x = 2$

Solution

a. (i) Because the value of x is 5 before execution, the guard condition $x > 2$ is true at the time it is evaluated. Hence the statement following **then** is executed, and so the value of $x + 1 = 5 + 1$ is computed and placed in the storage location corresponding to y. So after execution, $y = 6$.
(ii) Because the value of x is 2 before execution, the guard condition $x > 2$ is false at the time it is evaluated. Hence the statement following **else** is executed. The value of $x - 1 = 2 - 1$ is computed and placed in the storage location corresponding to x, and the value of $3 \cdot x = 3 \cdot 1$ is computed and placed in the storage location corresponding to y. So after execution, $y = 3$.

b. (i) Since $x = 5$ initially, the condition $x > 2$ is true at the time it is evaluated. So the statement following **then** is executed, and y obtains the value $2^5 = 32$.
(ii) Since $x = 2$ initially, the condition $x > 2$ is false at the time it is evaluated. Execution, therefore, moves to the next statement following the if-then statement, and the value of y does not change from its initial value of 0. ■

Iterative statements are used when a sequence of algorithm statements is to be executed over and over again. We will use two types of iterative statements: **while** loops and **for-next** loops.

A **while** loop has the form

while (*condition*)
[statements that make up the body of the loop]
end while

where *condition* is a predicate involving algorithm variables. The word **while** marks the beginning of the loop, and the words **end while** mark its end.

Execution of a **while** loop occurs as follows:

1. The *condition* is evaluated by substituting the current values of all the algorithm variables and evaluating the truth or falsity of the resulting statement.
2. If *condition* is true, all the statements in the body of the loop are executed in order. Then execution moves back to the beginning of the loop and the process repeats.
3. If *condition* is false, execution passes to the next algorithm statement following the loop.

The loop is said to be **iterated** (IT-a-rate-ed) each time the statements in the body of the loop are executed. Each execution of the body of the loop is called an **iteration** (it-er-AY-shun) of the loop.

Example 4.10.2 **Tracing Execution of a while Loop**

Trace the execution of the following algorithm segment by finding the values of all the algorithm variables each time they are changed during execution:

$i := 1, s := 0$
while $(i \leq 2)$
$s := s + i$
$i := i + 1$
end while

Solution Since i is given an initial value of 1, the condition $i \leq 2$ is true when the **while** loop is entered. So the statements within the loop are executed in order:

$$s = 0 + 1 = 1 \quad \text{and} \quad i = 1 + 1 = 2.$$

Then execution passes back to the beginning of the loop.

The condition $i \leq 2$ is evaluated using the current value of i, which is 2. The condition is true, and so the statements within the loop are executed again:

$$s = 1 + 2 = 3 \quad \text{and} \quad i = 2 + 1 = 3.$$

Then execution passes back to the beginning of the loop.

The condition $i \leq 2$ is evaluated using the current value of i, which is 3. This time the condition is false, and so execution passes beyond the loop to the next statement of the algorithm.

This discussion can be summarized in a table, called a **trace table**, that shows the current values of algorithm variables at various points during execution. The trace table for

a **while** loop generally gives all values immediately following each iteration of the loop. ("After the zeroth iteration" means the same as "before the first iteration.")

Trace Table

Variable Name	Iteration Number 0	1	2
i	1	2	3
s	0	1	3

■

The second form of iteration we will use is a **for-next** loop. A **for-next** loop has the following form:

for *variable* := *initial expression* **to** *final expression*
    [statements that make up
    the body of the loop]
next (*same*) *variable*

A **for-next** loop is executed as follows:

1. The **for-next** loop *variable* is set equal to the value of *initial expression*.
2. A check is made to determine whether the value of *variable* is less than or equal to the value of *final expression*.
3. If the value of *variable* is less than or equal to the value of *final expression*, then the statements in the body of the loop are executed in order, *variable* is increased by 1, and execution returns back to step 2.
4. If the value of *variable* is greater than the value of *final expression*, then execution passes to the next algorithm statement following the loop.

Example 4.10.3 **Trace Table for a for-next Loop**

Convert the **for-next** loop shown below into a **while** loop. Construct a trace table for the loop.

for $i := 1$ **to** 4
    $x := i^2$
next i

Solution The given **for-next** loop is equivalent to the following:

$i := 1$
while $(i \leq 4)$
    $x := i^2$
    $i := i + 1$
end while

Its trace table is as follows:

Trace Table

Variable Name	Iteration Number 0	1	2	3	4
x		1	4	9	16
i	1	2	3	4	5

■

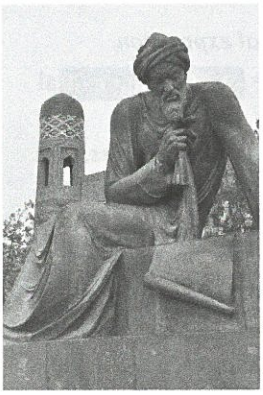
A Notation for Algorithms

Note Programming languages have various terms for subroutines: procedures, functions, routines, and subprograms.

We will express algorithms as subroutines that can be called upon by other algorithms as needed and used to transform a set of input variables with given values into a set of output variables with specific values. The output variables and their values are assumed to be returned to the calling algorithm. For example, the division algorithm specifies a procedure for taking any two positive integers as input and producing the quotient and remainder of the division of one number by the other as output. Whenever an algorithm requires such a computation, the algorithm can just "call" the division algorithm to do the job.

We generally include the following information when describing algorithms formally:

1. The name of the algorithm, together with a list of input and output variables.
2. A brief description of how the algorithm works.
3. The input variable names, labeled by data type (whether integer, real number, and so forth).
4. The statements that make up the body of the algorithm, possibly with explanatory comments.
5. The output variable names, labeled by data type.



Eduard Kim/Shutterstock.com

al-Khowârizmî (ca. 780–850)

You may wonder where the word *algorithm* came from. It evolved from the last part of the name of the Persian mathematician Abu Ja'far Mohammed ibn Mûsâ al-Khowârizmî. During Europe's Dark Ages, the Arabic world enjoyed a period of intense intellectual activity. One of the great mathematical works of that period was a book written by al-Khowârizmî that contained foundational ideas for the subject of algebra. The translation of this book into Latin in the thirteenth century had a profound influence on the development of mathematics during the European Renaissance.

The Division Algorithm

For an integer a and a positive integer d, the quotient-remainder theorem guarantees the existence of integers q and r such that

$$a = dq + r \quad \text{and} \quad 0 \leq r < d.$$

In this section, we give an algorithm to calculate q and r for given a and d where a is nonnegative. (The extension to negative a is left to the exercises at the end of this section.) The following example illustrates the idea behind the algorithm. Consider trying to find the quotient and the remainder of the division of 32 by 9, but suppose that you do not remember your multiplication table and have to figure out the answer from basic

principles. The quotient represents that number of 9's that are contained in 32. The remainder is the number left over when all possible groups of 9 are subtracted. Thus you can calculate the quotient and remainder by repeatedly subtracting 9 from 32 until you obtain a number less than 9:

$$\begin{aligned} 32 - 9 &= 23 \geq 9, \text{ and} \\ 32 - 9 - 9 &= 14 \geq 9, \text{ and} \\ 32 - 9 - 9 - 9 &= 5 < 9. \end{aligned}$$

This shows that 3 groups of 9 can be subtracted from 32 with 5 left over. Thus the quotient is 3 and the remainder is 5.

Algorithm 4.10.1 Division Algorithm

[Given a nonnegative integer a and a positive integer d, the aim of the algorithm is to find integers q and r that satisfy the conditions $a = dq + r$ and $0 \leq r < d$. This is done by subtracting d repeatedly from a until the result is less than d but is still nonnegative.

$$0 \leq a - d - d - d - \cdots - d = a - dq < d.$$

The total number of d's that are subtracted is the quotient q. The quantity $a - dq$ equals the remainder r.]

Input: *a [a nonnegative integer], d [a positive integer]*

Algorithm Body:

$r := a, q := 0$
[Repeatedly subtract d from r until a number less than d is obtained. Add 1 to q each time d is subtracted.]
while $(r \geq d)$
    $r := r - d$
    $q := q + 1$
end while
[After execution of the **while** *loop, $a = dq + r$.]*

Output: *q, r [nonnegative integers]*

Note that the values of q and r obtained from the division algorithm are the same as those computed by the *div* and *mod* functions built into a number of computer languages. That is, if q and r are the quotient and remainder obtained from the division algorithm with input a and d, then the output variables q and r satisfy

$$q = a \text{ div } d \quad \text{and} \quad r = a \bmod d.$$

The next example asks for a trace table for the division algorithm.

Example 4.10.4 **Tracing the Division Algorithm**

Trace the action of Algorithm 4.10.1 on the input variables $a = 19$ and $d = 4$.

Solution Make a trace table as shown on the next page. The column under the kth iteration gives the states of the variables after the kth iteration of the loop.

Variable Name	Iteration Number 0	1	2	3	4
a	19				
d	4				
r	19	15	11	7	3
q	0	1	2	3	4

■

The Euclidean Algorithm

The greatest common divisor of two integers a and b is the largest integer that divides both a and b. For example, the greatest common divisor of 12 and 30 is 6. The Euclidean algorithm provides a very efficient way to compute the greatest common divisor of two integers.

Definition

Let a and b be integers that are not both zero. The **greatest common divisor** of a and b, denoted $\mathbf{gcd}(a, b)$, is that integer d with the following properties:

1. d is a common divisor of both a and b. In other words,
$$d|a \quad \text{and} \quad d|b.$$
2. For every integer c, if c is a common divisor of both a and b, then c is less than or equal to d. In other words,
$$\text{for every integer } c, \text{ if } c|a \text{ and } c|b \text{ then } c \leq d.$$

Example 4.10.5 **Calculating Some gcd's**

a. Find gcd(72, 63).

b. Find gcd(10^{20}, 6^{30}).

c. In the definition of greatest common divisor, gcd(0, 0) is not allowed. Why not? What would gcd(0, 0) equal if it were found in the same way as the greatest common divisors for other pairs of numbers?

Solution

a. $72 = 9 \cdot 8$ and $63 = 9 \cdot 7$. So $9|72$ and $9|63$, and no integer larger than 9 divides both 72 and 63. Hence gcd(72, 63) = 9.

b. By the laws of exponents, $10^{20} = 2^{20} \cdot 5^{20}$ and $6^{30} = 2^{30} \cdot 3^{30} = 2^{20} \cdot 2^{10} \cdot 3^{30}$. It follows that
$$2^{20}|10^{20} \quad \text{and} \quad 2^{20}|6^{30},$$
and by the unique factorization of integers theorem, no integer larger than 2^{20} divides both 10^{20} and 6^{30} (because no more than twenty 2's divide 10^{20}, no 3's divide 10^{20}, and no 5's divide 6^{30}). Hence gcd(10^{20}, 6^{30}) = 2^{20}.

c. Suppose gcd(0, 0) were defined to be the largest common factor that divides 0 and 0. The problem is that *every* positive integer divides 0 and there is no largest integer. So there is no largest common divisor! ■

Calculating gcd's using the approach illustrated in Example 4.10.5 works only when the numbers can be factored completely. By the unique factorization of integers theorem, all numbers can, in principle, be factored completely. But, in practice, even using the highest-speed computers, the process is unfeasibly long for very large integers. Over 2,000 years ago, Euclid devised a method for finding greatest common divisors that is easy to use and is much more efficient than either factoring the numbers or repeatedly testing both numbers for divisibility by successively larger integers.

The Euclidean algorithm is based on the following two facts, which are stated as lemmas.

Lemma 4.10.1

If r is a positive integer, then $\gcd(r, 0) = r$.

Proof: Suppose r is a positive integer. *[We must show that the greatest common divisor of both r and 0 is r.]* Certainly, r is a common divisor of both r and 0 because r divides itself and also r divides 0 (since every positive integer divides 0). Also no integer larger than r can be a common divisor of r and 0 (since no integer larger than r can divide r). Hence r is the greatest common divisor of r and 0.

The proof of the second lemma is based on a clever pattern of argument that is used in many different areas of mathematics: To prove that $A = B$, prove that $A \leq B$ and that $B \leq A$.

Lemma 4.10.2

If a and b are any integers not both zero, and if q and r are any integers such that

$$a = bq + r,$$

then

$$\gcd(a, b) = \gcd(b, r).$$

Proof: *[The proof is divided into two sections: (1) proof that* $\gcd(a, b) \leq \gcd(b, r)$*, and (2) proof that* $\gcd(b, r) \leq \gcd(a, b)$*. Since each* gcd *is less than or equal to the other, the two must be equal.]*

1. **gcd(a, b) ≤ gcd(b, r):**
   a. *[We will first show that any common divisor of a and b is also a common divisor of b and r.]*

      Let a and b be integers, not both zero, and let c be a common divisor of a and b. Then $c \mid a$ and $c \mid b$, and so, by definition of divisibility, $a = nc$ and $b = mc$, for some integers n and m. Substitute into the equation

      $$a = bq + r$$

      to obtain

      $$nc = (mc)q + r.$$

(continued on page 252)

Then solve for r:

$$r = nc - (mc)q = (n - mq)c.$$

Now $n - mq$ is an integer, and so, by definition of divisibility, $c \mid r$. Because we already know that $c \mid b$, we can conclude that c is a common divisor of b and r *[as was to be shown].*

b. *[Next we show that* $\gcd(a, b) \leq \gcd(b, r)$*.]*

Now the greatest common divisor of a and b is defined because a and b are not both zero. Also, by part (a), every common divisor of a and b is a common divisor of b and r, and so the greatest common divisor of a and b is a common divisor of b and r. But then $\gcd(a, b)$ (being one of the common divisors of b and r) is less than or equal to the greatest common divisor of b and r:

$$\gcd(a, b) \leq \gcd(b, r).$$

2. **gcd(b, r) ≤ gcd(a, b):**

The second part of the proof is very similar to the first part. It is left as an exercise.

The Euclidean algorithm can be described as follows:

Euclidean Algorithm Description

1. Let A and B be integers with $A > B \geq 0$.
2. To find the greatest common divisor of A and B, first check whether $B = 0$. If it is, then $\gcd(A, B) = A$ by Lemma 4.10.1. If it isn't, then $B > 0$ and the quotient-remainder theorem can be used to divide A by B to obtain a quotient q and a remainder r:

$$A = Bq + r \quad \text{where } 0 \leq r < B.$$

By Lemma 4.10.2, $\gcd(A, B) = \gcd(B, r)$. Thus the problem of finding the greatest common divisor of A and B is reduced to the problem of finding the greatest common divisor of B and r.

[What makes this information useful is the fact that the larger number of the pair (B, r) is smaller than the larger number of the pair (A, B). The reason is that the value of r found by the quotient-remainder theorem satisfies

$$0 \leq r < B.$$

And, since by assumption $B < A$, we have that

$$0 \leq r < B < A.]$$

3. Now just repeat the process, starting again at (2), but use B instead of A and r instead of B. The repetitions are guaranteed to terminate eventually with $r = 0$ because each new remainder is less than the preceding one and all are nonnegative.

Note Strictly speaking, the fact that the repetitions eventually terminate is justified by the well-ordering principle for the integers, which is discussed in Section 5.4.

By the way, it is always the case that the number of steps required in the Euclidean algorithm is at most five times the number of digits in the smaller integer. This was proved by the French mathematician Gabriel Lamé (1795–1870).

The following example illustrates how to use the Euclidean algorithm.

Example 4.10.6 **Hand-Calculation of gcd's Using the Euclidean Algorithm**

Use the Euclidean algorithm to find gcd(330, 156).

Solution

1. Divide 330 by 156:

$$\begin{array}{r} 2 \leftarrow \text{quotient} \\ 156\overline{)\ 330} \\ \underline{312} \\ 18 \leftarrow \text{remainder} \end{array}$$

Thus $330 = 156 \cdot 2 + 18$, and hence gcd(330, 156) = gcd(156, 18) by Lemma 4.10.2.

2. Divide 156 by 18:

$$\begin{array}{r} 8 \leftarrow \text{quotient} \\ 18\overline{)\ 156} \\ \underline{144} \\ 12 \leftarrow \text{remainder} \end{array}$$

Thus $156 = 18 \cdot 8 + 12$, and hence gcd(156, 18) = gcd(18, 12) by Lemma 4.10.2.

3. Divide 18 by 12:

$$\begin{array}{r} 1 \leftarrow \text{quotient} \\ 12\overline{)\ 18} \\ \underline{12} \\ 6 \leftarrow \text{remainder} \end{array}$$

Thus $18 = 12 \cdot 1 + 6$, and hence gcd(18, 12) = gcd(12, 6) by Lemma 4.10.2.

4. Divide 12 by 6:

$$\begin{array}{r} 2 \leftarrow \text{quotient} \\ 6\overline{)\ 12} \\ \underline{12} \\ 0 \leftarrow \text{remainder} \end{array}$$

Thus $12 = 6 \cdot 2 + 0$, and hence gcd(12, 6) = gcd(6, 0) by Lemma 4.10.2.
Putting all the equations above together gives

$$\begin{aligned} \gcd(330, 156) &= \gcd(156, 18) \\ &= \gcd(18, 12) \\ &= \gcd(12, 6) \\ &= \gcd(6, 0) \\ &= 6 \qquad \text{by Lemma 4.10.1.} \end{aligned}$$

Therefore, gcd(330, 156) = 6. ■

The following is a version of the Euclidean algorithm written using formal algorithm notation.

Algorithm 4.10.2 Euclidean Algorithm

[Given two integers A and B with $A > B \geq 0$, this algorithm computes gcd(A, B)*. It is based on two facts:*

1. gcd(a, b) = gcd(b, r) *if a, b, q, and r are integers with* $a = b \cdot q + r$ *and* $0 \leq r < b$.
2. gcd$(a, 0) = a$.*]*

Input: *A, B [integers with* $A > B \geq 0$*]*

Algorithm Body:

$a := A, b := B, r := B$
[If $b \neq 0$*, compute a mod b, the remainder of the integer division of a by b, and set r equal to this value. Then repeat the process using b in place of a and r in place of b.]*
while $(b \neq 0)$
  $r := a \bmod b$
  [The value of a mod b can be obtained by calling the division algorithm.]
  $a := b$
  $b := r$
end while
[After execution of the **while** *loop,* gcd$(A, B) = a$.*]*
gcd$:= a$

Output: gcd *[a positive integer]*

Example 4.10.7 **A Trace Table for the Euclidean Algorithm**

Construct a trace table for Algorithm 4.10.2 using $A = 330$ and $B = 156$, the same numbers as in Example 4.10.6.

Solution

A	330				
B	156				
a	330	156	18	12	6
b	156	18	12	6	0
r	156	18	12	6	0
gcd					6

■

TEST YOURSELF

1. When an algorithm statement of the form $x := e$ is executed, _____.
2. Consider an algorithm statement of the following form.

   if (*condition*)
   then s_1
   else s_2

When such a statement is executed, the truth or falsity of the *condition* is evaluated. If *condition* is true, _____. If *condition* is false, _____.

3. Consider an algorithm statement of the following form.

   while (*condition*)
   [statements that make up the body of the loop]
   end while

   When such a statement is executed, the truth or falsity of the *condition* is evaluated. If *condition* is true, _____. If *condition* is false, _____.

4. Consider an algorithm statement of the following form.

   for *variable* : = *initial expression* **to** *final expression.*
   [statements that make up the body of the loop]
   next (*same*) *variable*

   When such a statement is executed, *variable* is set equal to the value of the *initial expression*, and a check is made to determine whether the value of *variable* is less than or equal to the value of *final expression*. If so, _____. If not, _____.

5. Given a nonnegative integer a and a positive integer d the division algorithm computes _____.

6. Given integers a and b, not both zero, gcd(a, b) is the integer d that satisfies the following two conditions: _____ and _____.

7. If r is a positive integer, then gcd(r, 0) = _____.

8. If a and b are integers not both zero and if q and r are nonnegative integers such that $a = bq + r$ then gcd(a, b) = _____.

9. Given positive integers A and B with $A > B$, the Euclidean algorithm computes _____.

EXERCISE SET 4.10

Find the value of z when each of the algorithm segments in 1 and 2 is executed.

1. $i := 2$
   if ($i > 3$ or $i \leq 0$)
   then $z := 1$
   else $z := 0$

2. $i := 3$
   if ($i \leq 3$ or $i > 6$)
   then $z := 2$
   else $z := 0$

3. Consider the following algorithm segment:
   if $x \cdot y > 0$ **then do** $y := 3 \cdot x$
   $x := x + 1$ **end do**
   $z := x \cdot y$
   Find the value of z if prior to execution x and y have the values given below.
   a. $x = 2, y = 3$
   b. $x = 1, y = 1$

Find the values of a and e after execution of the loops in 4 and 5 by first making trace tables for them.

4. $a := 2$
   for $i := 1$ **to** 3
   $a := 3a + 1$
   next i

5. $e := 2, f := 0$
   for $k := 1$ **to** 3
   $e := e \cdot k$
   $f := e + f$
   next k

Make a trace table to trace the action of Algorithm 4.10.1 for the input variables given in 6 and 7.

6. $a = 26, d = 7$

7. $a = 59, d = 13$

8. The following algorithm segment makes change; given an amount of money A between 1¢ and 99¢, it determines a breakdown of A into quarters (q), dimes (d), nickels (n), and pennies (p).

   $q := A$ *div* 25
   $A := A$ *mod* 25
   $d := A$ *div* 10
   $A := A$ *mod* 10
   $n := A$ *div* 5
   $p := A$ *mod* 5

   a. Trace this algorithm segment for $A = 69$.
   b. Trace this algorithm segment for $A = 87$.

Find the greatest common divisor of each of the pairs of integers in 9–12. (Use any method you wish.)

9. 27 and 72

10. 5 and 9

11. 7 and 21

12. 48 and 54

Use the Euclidean algorithm to hand-calculate the greatest common divisors of each of the pairs of integers in 13–16.

13. 1,188 and 385

14. 509 and 1,177

15. 832 and 10,933

16. 4,131 and 2,431

Make a trace table to trace the action of Algorithm 4.10.2 for the input variables given in 17–19.

17. 1,001 and 871

18. 5,859 and 1,232

19. 1,570 and 488

Definition: Integers a and b are said to be **relatively prime** if, and only if, their greatest common divisor is 1.

In 20 and 21 trace the action of Algorithm 4.10.2 to determine whether the integers are relatively prime.

20. 4,617 and 2,563 **21.** 34,391 and 6,728.

H 22. Prove that for all positive integers a and b, $a|b$ if, and only if, $\gcd(a, b) = a$. (Note that to prove "A if, and only if, B," you need to prove "if A then B" and "if B then A.")

23. a. Prove that if a and b are integers, not both zero, and $d = \gcd(a, b)$, then a/d and b/d are integers with no common divisor that is greater than 1.

b. Write an algorithm that accepts the numerator and denominator of a fraction as input and produces as output the numerator and denominator of that fraction written in lowest terms. (The algorithm may call upon the Euclidean algorithm as needed.)

24. Complete the proof of Lemma 4.10.2 by proving the following: If a and b are any integers with $b \neq 0$ and q and r are any integers such that

$$a = bq + r.$$

then $\gcd(b, r) \leq \gcd(a, b)$.

H 25. a. Prove: If a and d are positive integers and q and r are integers such that $a = dq + r$ and $0 < r < d$, then

$$-a = d(-(q+1)) + (d - r)$$

and $0 < d - r < d$.

b. Indicate how to modify Algorithm 4.10.1 to allow for the input a to be negative.

26. a. Prove that if a, d, q, and r are integers such that $a = dq + r$ and $0 \leq r < d$, then

$$q = \lfloor a/d \rfloor \quad \text{and} \quad r = a - \lfloor a/d \rfloor \cdot d.$$

b. In a computer language with a built-in-floor function, *div* and *mod* can be calculated as follows:

$$a \text{ div } d = \lfloor a/d \rfloor \quad \text{and} \quad a \text{ mod } d = a - \lfloor a/d \rfloor \cdot d.$$

Rewrite the steps of Algorithm 4.10.2 for a computer language with a built-in floor function but without *div* and *mod*.

27. An alternative to the Euclidean algorithm uses subtraction rather than division to compute greatest common divisors. (After all, division is repeated subtraction.) It is based on the following lemma.

Lemma 4.10.3
If $a \geq b > 0$, then $\gcd(a, b) = \gcd(b, a - b)$.

Algorithm 4.10.3 Computing gcd's by Subtraction
[Given two positive integers A and B, variables a and b are set equal to A and B. Then a repetitive process begins. If $a \neq 0$, and $b \neq 0$, then the larger of a and b is set equal to $a - b$ (if $a \geq b$) or to $b - a$ (if $a < b$), and the smaller of a and b is left unchanged. This process is repeated over and over until eventually a or b becomes 0. By Lemma 4.10.3, after each repetition of the process,

$$\gcd(A, B) = \gcd(a, b).$$

After the last repetition,

$$\gcd(A, B) = \gcd(a, 0) \quad \text{or} \quad \gcd(A, B) = \gcd(0, b)$$

depending on whether a or b is nonzero. But by Lemma 4.10.1,

$$\gcd(a, 0) = a \quad \text{and} \quad \gcd(0, b) = b.$$

Hence, after the last repetition,

$$\gcd(A, B) = a \text{ if } a \neq 0 \quad \text{or} \quad \gcd(A, B) = b \text{ if } b \neq 0.]$$

Input: *A, B [positive integers]*

Algorithm Body:
$a := A, b := B$
while ($a \neq 0$ and $b \neq 0$)
  if $a \geq b$ **then** $a := a - b$
    else $b := b - a$
end while
if $a = 0$ **then** gcd $:= b$
else gcd $:= a$
[After execution of the **if-then-else** *statement,* gcd $= \gcd(A, B)$.*]*

Output: gcd *[a positive integer]*

a. Prove Lemma 4.10.3.

b. Trace the execution of Algorithm 4.10.3 for $A = 630$ and $B = 336$.

c. Trace the execution of Algorithm 4.10.3 for $A = 768$ and $B = 348$.

Exercises 28–32 refer to the following definition.

Definition: The **least common multiple** of two nonzero integers a and b, denoted **lcm(a, b)**, is the positive integer c such that
a. $a|c$ and $b|c$
b. for all positive integers m, if $a|m$ and $b|m$, then $c \leq m$.

28. Find
- **a.** lcm(12, 18)
- **b.** lcm($2^2 \cdot 3 \cdot 5$, $2^3 \cdot 3^2$)
- **c.** lcm(2800, 6125)

29. Prove that for all positive integers a and b, $\gcd(a, b) = \text{lcm}(a, b)$ if, and only if, $a = b$.

30. Prove that for all positive integers a and b, $a \mid b$ if, and only if, $\text{lcm}(a, b) = b$.

31. Prove that for all integers a and b, $\gcd(a, b) \mid \text{lcm}(a, b)$.

32. Prove that for all positive integers a and b, $\gcd(a, b) \cdot \text{lcm}(a, b) = ab$.

ANSWERS FOR TEST YOURSELF

1. the expression e is evaluated (using the current values of all the variables in the expression), and this value is placed in the memory location corresponding to x (replacing any previous contents of the location) **2.** statement s_1 is executed; statement s_2 is executed **3.** all statements in the body of the loop are executed in order and then execution moves back to the beginning of the loop and the process repeats; execution passes to the next algorithm statement following the loop **4.** the statements in the body of the loop are executed in order, *variable* is increased by 1, and execution returns to the top of the loop; execution passes to the next algorithm statement following the loop **5.** integers q and r with the property that $n = dq + r$ and $0 \leq r < d$ **6.** d divides both a and b; if c is a common divisor of both a and b, then $c \leq d$ **7.** r **8.** $\gcd(b, r)$ **9.** the greatest common divisor of A and B (*Or*: $\gcd(A, B)$)