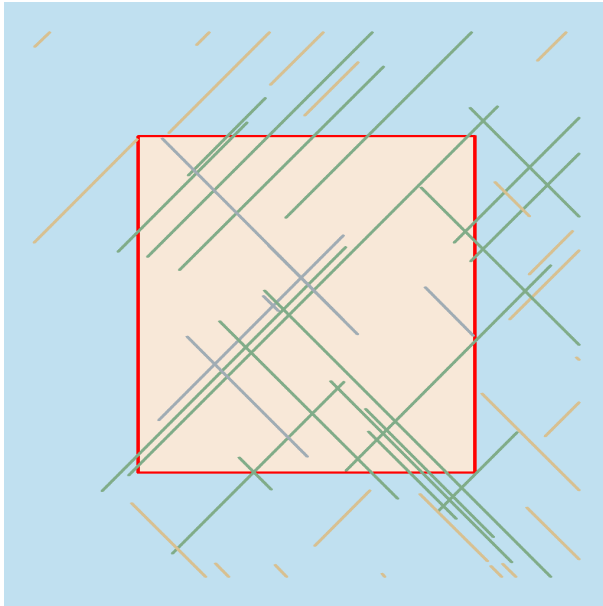


The Purpose of Computing

Leon Hannah Tabak



MIT License

Copyright ©2017 Leon Hannah Tabak

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Creative Commons License

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit...

<http://creativecommons.org/licenses/by-sa/4.0/>

...or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Contact the author:

Leon Hannah Tabak
Cornell College
600 First Street SW
Mount Vernon, Iowa 52314
Voice: (319) 895-4294
E-mail: l.tabak@ieee.org

For Ariel and Avidan

Contents

I Day	1
1	3
2	7
3	11
3.1 Video clips	11
3.2 Binary numbers	12
3.3 Exercises	13
4	17
4.1 Video clips	17
4.2 A young field?	18
4.3 Exercises	18

5	23
5.1 Review	23
5.2 Programming languages	25
5.3 Exercises	28
II Day	29
6	31
6.1 Classes that have constructors.	31
6.2 Class that have no constructors.	33
6.3 A complete example and an exercise.	34
7	37
7.1 Video clip	37
7.2 Exercises	37
8	41
8.1 Exercises	42
III Day	43
9	45

CONTENTS

9.1 Exercises 47

10 **53**

11 **61**

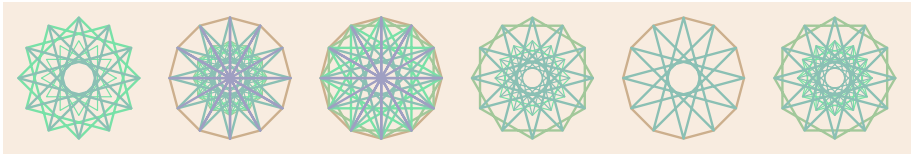
IV Day **69**

12 **71**

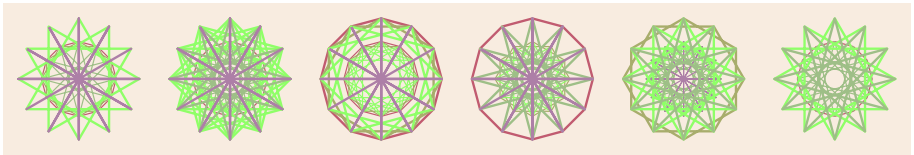
13 **77**

14 **83**

Part I



Foundations



Lesson 1

We will discuss reasons for using the Java programming language rather than another in our course. We will compare programming languages to the languages that people speak.

1. Benjamin Whorf promoted the idea of linguistic relativity.
 - Who was he?
 - What is “linguistic relativity?”
2. Wilhelm von Humboldt also developed ideas related to linguistic relativity. We will learn the Java programming language. Von Humboldt studied a language of the people of the island of Java. Tell us something more about Wilhelm von Humboldt.
3. Das Weltbild, die Weltanschauung, and die Weltsicht are three German nouns with similar meanings. The meanings are related to the ideas developed by Whorf and von Humboldt. Look up the definition of one of these words. (If you are feeling more ambitious, try to find how their meanings differ.) Can you see the connection between the meanings of these words and the meaning of linguistic relativism?
4. What do the following German words mean?
 - das Erklärungsnot

- das Fremdschämen
 - das Torschlusspanik
 - das Treppenwitz
 - verschlimmbessern
5. Think of how the German words in the previous question might find application in a translation of the following story: You have found in the company that you and a partner have created a once in a lifetime opportunity. You have invested everything you have. For a year, you have been leading an effort to develop software on which the company has bet its future. You are racing to get to market before bigger companies that have more resources can do the same. You are close to finishing an excellent design. Now you worry that more work might compromise the simplicity, soundness, and elegance of your design. You have promised delivery to your most important client by the end of the month. Yesterday, your team met with the client. Your partner had insisted on presenting the product to this client, but bungled the demonstration and could not answer the clients questions. Your partner did not know enough to feel embarrassed. You squirmed. Then, without warning, your partner suddenly called you to the stage. You saved the day. On the drive home, you thought of what you should have said to your ill-prepared partner after the nearly disastrous meeting.
- Do you think it is possible to produce a German version that conveys not just the same facts, but also the same emotions, assumptions, and values?
 - Does the language we use limit what we can say?
 - Are there ideas that we can express in one language but not another?
6. The Imitation Game is a motion picture that tells the story of Alan Turing. Who was Alan Turing?
7. What is a Turing machine?
8. What is the Church-Turing Thesis?

9. Why should we use the Java programming language? Search on the Web. What do other people have to say?
10. Learn a little about the origins, applications, and popularity of these other programming languages.
 - C#
 - JavaScript
 - Python
 - Ruby

Lesson 2

Step	R0C0	R0C1	R0C2	R0C3	Best guess
1:	R1C0	R1C1	R1C2	R1C3	R1C4
2:	R2C0	R2C1	R2C2	R2C3	R2C4
3:	R3C0	R3C1	R3C2	R3C3	R3C4
4:	R4C0	R4C1	R4C2	R4C3	R4C4

Figure 2.1: Sequential search for minimum value.

Step	R0C0	R0C1	R0C2	R0C3	Best guess
1:	R1C0	R1C1	R1C2	R1C3	R1C4
2:	R2C0	R2C1	R2C2	R2C3	R2C4
3:	R3C0	R3C1	R3C2	R3C3	R3C4
4:	R4C0	R4C1	R4C2	R4C3	R4C4
					R4C4

Figure 2.2: Sequential search for index of minimum value.

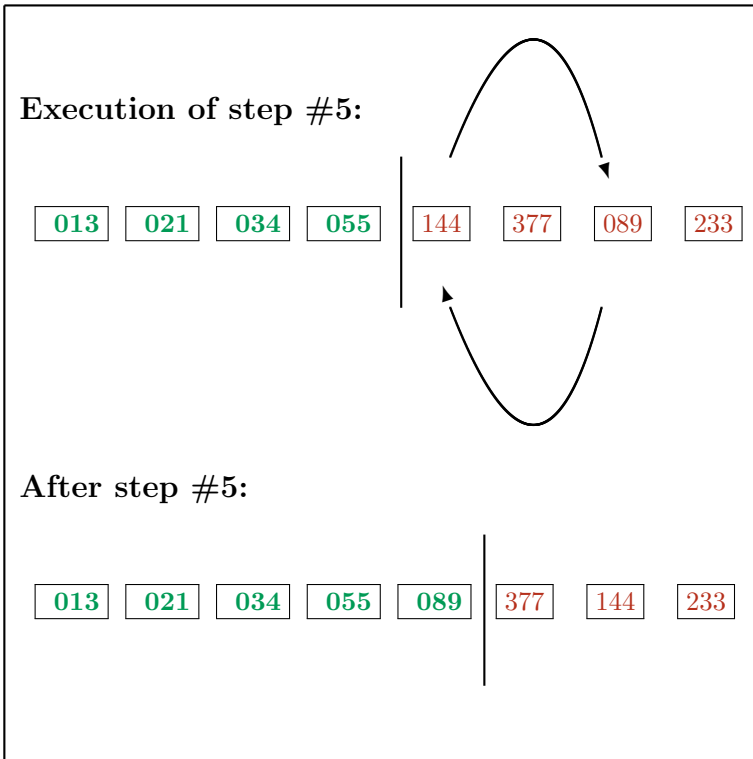


Figure 2.3: Selection sort.

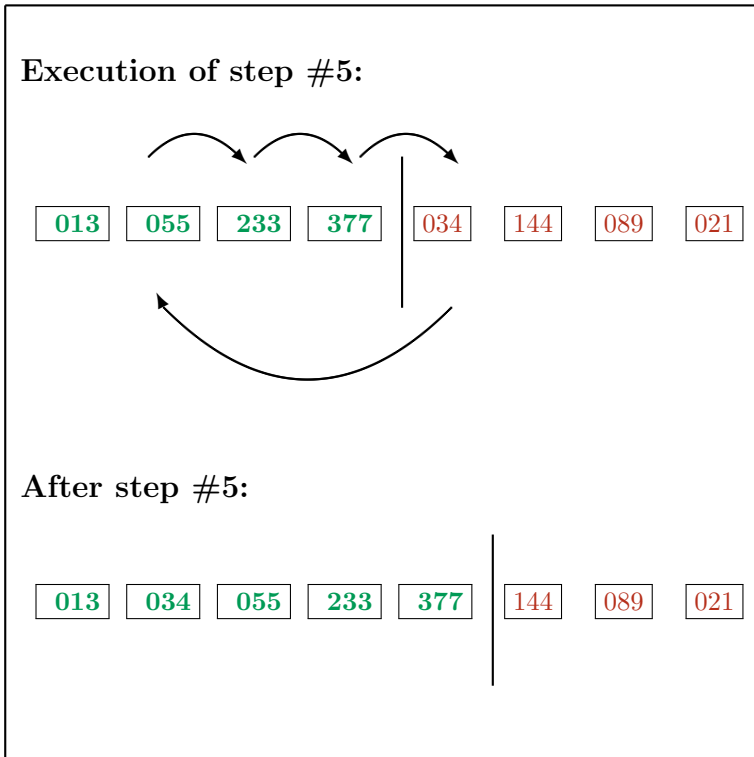


Figure 2.4: Insertion sort.

Step	<	>	50	73	61	65	84	89	31	19	>
1	<	50	>	73	61	65	84	89	31	19	>
2	<	50	73	>	61	65	84	89	31	19	>
3	<	50	61	73	>	65	84	89	31	19	>
4	<	50	61	65	73	>	84	89	31	19	>
5	<	50	61	65	73	84	>	89	31	19	>
6	<	50	61	65	73	84	89	>	31	19	>
7	<	31	50	61	65	73	84	89	>	19	>
8	<	19	31	50	61	65	73	84	89	>	>
	<	19	31	50	61	65	73	84	89	>	>

Figure 2.5: Selection sort.

Step	<	>	58	29	52	67	33	79	76	97	>
1	<	29	>	58	52	67	33	79	76	97	>
2	<	29	33	>	52	67	58	79	76	97	>
3	<	29	33	52	>	67	58	79	76	97	>
4	<	29	33	52	58	>	67	79	76	97	>
5	<	29	33	52	58	67	>	79	76	97	>
6	<	29	33	52	58	67	76	>	79	97	>
7	<	29	33	52	58	67	76	79	>	97	>
8	<	29	33	52	58	67	76	79	97	>	>
	<	29	33	52	58	67	76	79	97	>	>

Figure 2.6: Insertion sort.

Lesson 3

“As I have always held it a crime to anticipate evils, I will believe it a good comfortable road until I am compelled to believe differently.”

—Meriwether Lewis [1]

“All programmers are optimists. Perhaps this modern sorcery especially attracts those who believe in happy endings and fairy godmothers. Perhaps the hundreds of nitty frustrations drive away all but those who habitually focus on the end goal.”

—Frederick P. Brooks [2, page 35]

3.1 Video clips

- “Forbidden Planet” (1956)
- “Desk Set” (1957)
- “2001: A Space Odyssey” (1968)
- “Bicentennial Man” (1999)
- “Her” (2013)

3.2 Binary numbers

Engineers have found it convenient to build computers that represent integers in binary (base 2) form. This is not an essential feature of computers. It is possible to build computers that work with decimal numbers. It is just that binary arithmetic makes designing digital electronics easier.

Different levels of voltage represent different digits. In designing a computer whose native language is binary numbers, engineers only have to distinguish between two levels of voltage. That makes the engineers' job easier. It allows simpler circuits.

You know that...

$$\begin{aligned} 1905 &= 1 \cdot 10^3 + 9 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0 \\ &= 1000 + 900 + 0 + 5 \end{aligned}$$

This is place value arithmetic. In decimal numbers, each digit represents a multiple of a power of ten. In binary numbers, each digit represents a multiple of a power of two.

$$\begin{aligned} 101010_2 &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= 32 + 8 + 2 \\ &= 42_{10} \end{aligned}$$

To compute the product of two powers of the same number, add the exponents:

$$\begin{aligned}2^2 &= 2 \cdot 2 \\ &= 4 \\ 2^3 &= 2 \cdot 2 \cdot 2 \\ &= 8 \\ 2^5 &= 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \\ &= (2 \cdot 2 \cdot 2) \cdot (2 \cdot 2) \\ &= 2^3 \cdot 2^2 \\ &= 8 \cdot 4 \\ &= 32 \\ 2^5 &= 2^3 \cdot 2^2 = 2^{3+2}\end{aligned}$$

Every computer scientist should know some powers of two by heart and be able to produce others. The table in figure 3.1 will help you get started.

Commit $2^8 = 256$ and $2^{10} = 1024$ to memory. Learn how to calculate and estimate the values of other powers of two. Figure 3.1 includes models that show you how to do this. For example, $2^{12} = 2^{10} \cdot 2^2 = 1024 \cdot 4 \approx 1000 \cdot 4$.

3.3 Exercises

1. Look at the keyboard on your computer. Estimate the total number of different letters (count uppercase and lowercase!), digits, punctuation marks. Round up to the nearest power of two. The exponent on that power of two is the minimum number of bits required to specify a character.

If we want to construct a code that assigns a binary number to each character that we can type, how big (how many bits) must that number be?

$$\begin{aligned}2^0 &= 1 \\2^1 &= 2 \\2^2 &= 4 \\2^3 &= 8 \\2^4 &= 2^2 \cdot 2^2 = 4 \cdot 4 = 16 \\2^5 &= 32 \\2^6 &= 2^4 \cdot 2^2 = 16 \cdot 4 = 64 \\2^7 &= 128 \\2^8 &= 2^4 \cdot 2^4 = 16 \cdot 16 = 256 \\2^9 &= 512 \\2^{10} &= 1024 \\2^{11} &= 2048 \\2^{12} &= 2^{10} \cdot 2^2 = 1024 \cdot 4 = 4096 \\2^{13} &= 8192 \\2^{14} &= 16,384 \\2^{15} &= 32,768 \\2^{16} &= 2^{10} \cdot 2^6 = 1024 \cdot 64 = 65,536 \\&\vdots \\2^{20} &= 2^{10} \cdot 2^{10} = 1024 \cdot 1024 = 1,048,576 \\&\vdots \\2^{32} &= 4,294,967,296 \\&\vdots \\2^{64} &= 18,446,744,073,709,551,616\end{aligned}$$

Figure 3.1: Powers of 2.

2. How many meters are in a kilometer? How many bytes are in a kilobyte? Be careful!
3. 2^{10} is approximately one thousand. Therefore, $2^{20} = 2^{10} \cdot 2^{10}$ is approximately equal to what number?
4. Which powers of two correspond to the prefixes “mega,” “giga,” and “tera?” Approximately which powers of ten correspond to the same prefixes?
5. The Java platform includes a class named `Color`. That class defines a constructor that has three integer parameters—to create a color, the programmer specifies the amount of red, green, and blue in the color with three integers. The values must lie within the range 0 to 255—there are 2^8 different possible values of red, 2^8 possible values of green, and 2^8 possible values of blue. The total number of possible colors is then $2^8 \cdot 2^8 \cdot 2^8$.

Approximately how many millions of different colors are possible?

6. The arithmetic that you see in a course in computer science will sometimes differ from the arithmetic you see in courses on mathematics!

Copy this Java source code into a file on your computer. Compile and execute it.

Which results surprise you?

```

package arithmetic ;

public class Arithmetic {
    private static final int BILLION = 1000000000;

    public static void main( String [] arguments ) {
        String label = "1 billion + 1 billion = ";
        int sum = BILLION + BILLION;
        System.out.println( label + sum );

        label= "1 billion + 1 billion + 1 billion = ";
        sum = BILLION + BILLION + BILLION;
    }
}

```

```

System.out.println( label + sum );

double x = 8.0/0.0;
String s = "(8.0/0.0)";
System.out.println( s + " = " + x);
System.out.println( "-" + s + " = " + -x);
System.out.println( s + " + " + s + " = " + (x + x));
System.out.println( s + " - " + s + " = " + (x - x));
System.out.println( s + " * " + s + " = " + (x * x));
System.out.println( s + " / " + s + " = " + (x / x));

System.out.println( "5 / 3 = " + (5/3));
System.out.println( "5.0 / 3.0 = " + (5.0/3.0));
System.out.println( "5 % 3 = " + (5%3));
System.out.println( "3 * (5/3) + (5%3) = " +
                    (3*(5/3) + (5%3)));

System.out.println( "3.0 * (5.0/3.0) + (5.0%3.0) = " +
                    (3.0*(5.0/3.0) + (5.0%3.0)));

    } // main( String [] )
} // Arithmetic

```


Lesson 4

“I think plans are highly overrated. Plans are a baseline, in my opinion, a model of a life that you depart from as you go on. Very rarely do they take you to the place you envisioned at the time you started.”

—Andrew S. Grove [10]

“I say ‘innovation’ rather than simply ‘invention,’ because innovation, to me, means invention implemented. And I have grudgingly come to realize that invention is often the easy part of innovation. The hard part is usually the implementation.”

—John Seely Brown [5]

4.1 Video clips

- “Big Ideas”
- “Our Co-workers”
- “Our Doodles”
- “Our Parties”
- “Rock Star”

4.2 A young field?

Computer scientists continue to regard their field as a young field. For how much longer can we call computer science young? To what else are people comparing computer science when they call it young?

Progress has been very rapid in some respects. Moore's Law may be the best known example and description of fast-paced developments in computing. (You do not know Moore's Law? Look it up!) At the same time, it has sometimes taken decades for important ideas to move from laboratories to widespread practice in industry. RISC microprocessors, object-oriented programming, and support for functional programming in the most widely used programming languages are cases in point.

Some perspective will help us understand how innovation proceeds. With an understanding of the past, we can form realistic expectations of the future.

Computer science has produced a lot of jargon.

4.3 Exercises

1. Construct a timeline of the history of computing. Associate with each point the name of one of the people listed in Figure 4.1 together with an achievement of that person. An achievement could be the publication of an influential article or book, an invention or discovery, the creation of a company or product, an award, a prediction, or an appointment to an important office.

You do not have to refer to every person in this list on your timeline—let's divide the work among everyone in the class.

Find the information that you need on the Internet.

2. Define the terms listed in Figure 4.2. Short statements will suffice in some cases. In other cases, you might find a story about

the origin of the term to add to the definition.

You do not each have to define every term in this list—let's divide the work among everyone in the class.

Find the information that you need on the Internet.

Hal Abelson	Marc Andreessen	Isaac Asimov
John Atanasoff	Charles Babbage	John Backus
Steve Balmer	Gordon Bell	Leo Beranek
Fred Brooks, Jr.	Vannevar Bush	Vint Cerf
Edgar F. (Ted) Codd	Stephen Cook	Seymour Cray
Peter J. Denning	Edsger Dijkstra	John Doerr
J. Presper Eckert	Richard Feynman	Carly Fiorina
Bill Gates	Adele Goldberg	Herman Goldstine
Shafi Goldwasser	James Gosling	Andy Grove
Richard Hamming	John Hennessy	Danny Hillis
C.A.R. (Tony) Hoare	Herman Hollerith	Grace Murray Hopper
Steve Jobs	Bob Kahn	Richard Karp
Gary Kasparov	John Kemeny	Vinod Khosla
Jack Kilby	Donald E. Knuth	Ray Kurzweil
John Lasseter	Gottfried Wilhelm Leibniz	J.C.R. Licklider
Barbara Liskov	Ada Lovelace	John Mauchly
Marissa Mayer	John McCarthy	Marvin Minsky
Gordon Moore	Peter Naur	Robert Noyce
Ken Olsen	David Patterson	Radia Perlman
Eric Raymond	Ginni Rometty	Ken Sakamura
Claude Shannon	William Shockley	Balaji Srinivasan
Richard Stallman	Cliff Stoll	Bjarne Stroustrup
Ivan Sutherland	Andy Tanenbaum	Linus Torvalds
Alan Turing	John von Neumann	Thomas Watson, Jr.
Meg Whitman	Norbert Wiener	Maurice Wilkes
Jeanette Wing	Niklaus Wirth	Steve Wozniak
Konrad Zuse		

Figure 4.1: People

ALGOL	ALU	ASCII	BASIC	bit
byte	CAD	CAE	CFG	CISC
CMOS	COBOL	CPU	CP/M	CSS
DBMS	DEC	DOS	ECL	ENIAC
FTP	GaAs	GNU	GPL	GUI
HAL 9000	HDL	HTML	IBM	IC
ICAAN	JAR	JPEG	JSON	LAMP
Lisp	MANIAC	MIME	MIPS	MP3
NFS	NP	OOP	PCB	PDP-11
PKE	PL/I	PNG	RAID	RAM
RISC	ROM	RSA	SMTP	SPARC
SUN	SQL	TCP/IP	TLD	TTL
UNIVAC	URL	VAX	VLSI	VME
VMS	WNT	WYSIWYG	XM	

Figure 4.2: Acronyms, initialisms, invented words, and portmanteaus.

Lesson 5

“It’s the way I study—to understand something by trying to work it out or, in other words, to understand something by creating it. Not creating it one hundred percent, of course; but taking a hint as to which direction to go but not remembering the details. These you work out for yourself.”

—Richard P. Feynman [7, page 15]

“It is not a question of creativity versus discipline; creative work is simply not possible without discipline.”

—Watts S. Humphrey [12, page 26]

5.1 Review

1. Create a Web page that displays the words “Heuristically programmed ALgorithmic computer” in a top level headline. Use just the *body*, *h1*, *head*, *html*, and *title* tags.
2. Beginning in the 1940s, this prolific author established his reputation by writing science fiction. In many stories, he explored dilemmas that arise when intelligent robots try to obey the three laws that he invented.

The three laws are:

- A robot may not injure a human being or, through inaction, allow a human being to come to harm.
- A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
- A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Who is he?

3. He shared what his experience leading a large programming project for IBM in the early 1960s had taught him in *The Mythical Man-Month*. He later joked that the book had become “The Bible of Software Engineering” because, like the Holy Book, “everybody quotes it, some people read it, and a few people go by it.”

Who is he?

4. She studied computer science at Northwestern University. After a short stint with General Motors Institute, she joined IBM. She contributed in engineering, consulting, sales, and marketing roles. She led the effort to find commercial applications for IBM’s *Jeopardy* playing computer, Watson. Last year, she became IBM’s president and chief executive officer.

Who is she?

5. A professor emeritus at Stanford University, he pioneered methods of analyzing algorithms, wrote the enormously influential *The Art of Computer Programming*, and created software to typeset the book. Scientists and mathematicians around the world have subsequently adopted that software (called `TEX`). They use it to write dissertations, articles, and books.

He pays readers who find errors in his books, but no one cashes the checks.

Who is he?

6. While employed at the Digital Employment Corporation, she invented a network protocol that greatly improved Ethernet, the very widely used local area network technology. She also invented an important algorithm that is used to route packets through the Internet.

You can find references to her as the “Mother of the Internet.” However, please do not call her that if you meet her. She does not like that accolade.

Who is she?

7. He studied physics at Cornell College, earned a doctoral degree at Harvard University, taught at the Massachusetts Institute of Technology, and created a company that built the first packet switches for the Internet.

His journey from Cornell College to Harvard University began when he offered to change the tire of a stranger who was passing through Mount Vernon.

Who is he?

8. A journalist, he once wrote for *Rolling Stone*, but then developed a fascination with the fast developing personal computers that he used for his writing. He has written books about the hacker ethic, Apple’s creation of the Macintosh, Google, and cryptography.

Who is he?

5.2 Programming languages

Every programming language is like every other programming language in this respect: if it is possible to write a program in language A, it is possible to write a program in language B that produces exactly the same outputs when given the same inputs.

Every computer is like every other computer in this respect: it is possible to write a program on computer B that perfectly simulates the

functions of computer A. Again, “perfect simulation” means that the same sequence of words and numbers fed into the two computers will cause the two computers to emit the same output (which might be signals, images, text, sounds, movements of a robot’s arm, or something else).

Of course, there are also many important differences that distinguish the many programming languages and models of computers. We will give special attention to features of a language that make it easier for human beings to read and write programs.

Here is an outline of elements of the Java programming language:

Not all of the features listed here appear in all other languages because not all features are independent of one another. For example, if a programming language includes a means of adding positive and negative numbers, it does not need a separate operator for subtraction—a subtraction is just the addition of a negative number.

- data types
 - primitive types: **boolean** (**true** or **false**), **double** (numbers that can have a fractional part), **int** (whole numbers)
 - and others that we will mostly ignore
 - arrays (refer to individual elements with an index—for example, `student[12]`)
 - reference types—pointers to instances of classes (e.g., `Color`, `PriorityQueue`, `Rectangle2D`) (refer to individual elements with a period and the name of the field—for example, `student.firstName`)
- declarations—instruct the computer to allocate space in its memory for the storage of a value, give that location a name, and specify the type of value that will be stored
- expressions
 - operands

- * literals (numbers, the words **true** and **false**)
- * variables (names of locations in computer's memories that hold values)
- * calls to functions
- operators
 - * binary arithmetic operators: +, -, % (remainder), *, /
 - * unary arithmetic operator(s): -
 - * binary logical operators: && (“and”), || (“or”)
 - * unary logical operator(s): ! (“not”)
- statements
 - assignment—instruct the computer to evaluate an expression and then store the expression's value in a named location in the computer's memory
 - **if**—instruct the computer to conditionally execute a statement
 - (**switch**)
 - **for**—instruct the computer to execute a statement repeatedly (a specified number of times)
 - **while**—instruct the computer to execute a statement repeatedly (so long as some condition still holds true)
 - (**do while**)
- divide a big program into units of manageable size
 - methods—a sequence of statements that has a single, well-defined purpose
 - classes—a blueprint for the construction of objects
 - objects—a collection of related data and the methods for operating on that data (where “operating” might mean changing the data, retrieving that data, or combining that data with other data)
- compose small units—treat the assembly as a single unit

- compound statement—an assembly of statements within a method
- methods—an assembly of statements
- class—an assembly of variables and methods
- package—an assembly of related classes

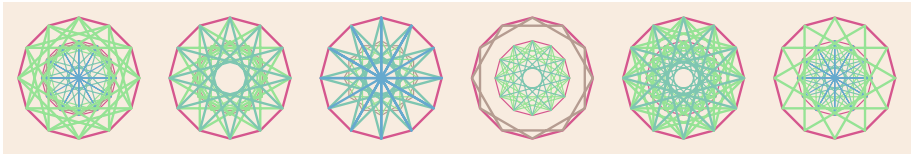
5.3 Exercises

1. The name of which one of Java's primitive types honors an English mathematician of the nineteenth century?
2. Java makes a distinction between whole numbers and numbers that may contain a fractional part (for example, 3.14159265) or an exponent (for example, Avogadro's number $6.022 \cdot 10^{23}$ and the Planck constant $6.62 \cdot 10^{-34}$).

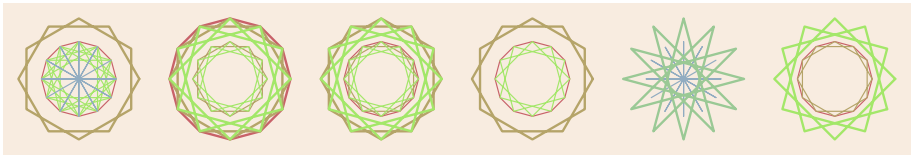
Which of the keywords of the Java programming language do programmers use to label variables to which they intend to assign values that might include a fractional part or an exponent?

3. Order the words “**class**,” “method,” and “**package**” so that the first item in your list is a something that can contain the second thing named in your list, and the second item is something that can contain the third thing named in your list.
4. Which of these delimiters do Java programmers use to mark the beginning and end of a sequence of statements: braces {}, parentheses (), or brackets [] ?

Part II



Classes



Lesson 6

“If we perceive our role aright, we then see more clearly the proper criterion for success: a toolmaker succeeds as, and only as, the *users* of his tool succeed with his aid. However shining the blade, however jeweled the hilt, however perfect the heft, a sword is tested only by cutting. That swordsmith is successful whose clients die of old age.”

—Frederick P. Brooks [4, page 62]

“The real significance of computing was to be found not in this gadget or that gadget, but in how the technology was woven into the fabric of human life—how computers could change the way people thought, the way they created, the way they communicated, the way they worked together, the way they organized themselves, even the way they apportioned power and responsibility.”

—M. Mitchell Waldrop [20, page 342]

6.1 Classes that have constructors.

One kind of a class is a blueprint the we use to construct objects. Each object is an “instance of the class.”

For example, Java provides a `Color` class. This class has a constructor that requires three integer arguments. The integers specify the amount of red, green, and blue in a color. A programmer can create any number of colors with the `Color`.

```
// Colors of the flag of the United States of America.
Color red = new Color( 255, 0, 0 );
Color white = new Color( 255, 255, 255 );
Color blue = new Color( 0, 0, 255 );
```

The `Color` class has many methods. These methods includes accessor methods that enable a programmer to retrieve the values of a color's red, green, and blue fields.

```
Color peach = new Color( 248, 206, 160 );
int howMuchRed = peach.getRed();
```

Every Java application has at least one class. It may contain many classes. At least one class in an application must contain a `main()` method.

A `main()` method looks like this:

```
public static void main( String [] args ) {
    // Statements go here.
} // main( String [] )
```

Although there is a way to define classes that do not need constructors, the classes that we will write will have constructors. (A class may have more than one constructor if the constructors have different numbers and types of parameters.)

If the name of a class is `MyClass`, then its constructor might look like this:

```
public MyClass() {
    // Statements go here.
} // MyClass()
```


We will begin our definition of classes by writing something like this:

```
public class MyClass {
    public MyClass () {
        // Statements go here.
    } // MyClass ()

    // Other methods go here.

    public static void main( String [] args ) {
        MyClass myClass = new MyClass ();
    } // main( String [] )
} // MyClass
```

6.2 Class that have no constructors.

The `Math` differs from the `Color` class in this important way: we might want to create many colors (each with different amounts of red, green, and blue) but we have no need to create multiple `Math` objects. Mathematics is the same everywhere and always.

The `Math` class is not a blueprint. It is a bundle of related methods (square root, sine, cosine, logarithm, and others) and constants (the numbers `pi` and `e`). All of the methods in the `Math` class have the **static** attribute (just like the `main()`). The **static** qualifier makes these methods callable even before we create an instance of a class.

To call a **static** method, write the name of the class that contains the method, a period, and then the name of the method:

```
double logarithm = Math.log( 1.0 );
```

Contrast this with calls to methods that are not **static**:

```
Color leaves = new Color( 128, 204, 192 );
int green = leaves.getGreen ();
```

The call to `getGreen()` follows the name of the object (`leaves`) and not the name of the class (`Color`). Each `Color` object carries around its own means of reporting how much green it contains.

6.3 A complete example and an exercise.

Define a class that models lengths that are measured in feet and inches. Use the following class as a model.

```

package weight;

public class Weight {
    public static final int OUNCES_PER_POUND = 16;
    private int pounds;
    private int ounces;

    public Weight( int pounds, int ounces ) {
        this.pounds = pounds;
        this.ounces = ounces;
    } // Weight( int, int )

    public int getPounds() {
        return this.pounds;
    } // getPounds()

    public int getOunces() {
        return this.ounces;
    } // getOunces()

    public Weight add( Weight w ) {
        int lbs = this.pounds + w.pounds +
            (this.ounces + w.ounces)/OUNCES_PER_POUND;
        int oz = (this.ounces + w.ounces) % OUNCES_PER_POUND;

        return new Weight( lbs, oz );
    }
}

```

```
    } // add( Weight )

    @Override
    public String toString() {
        return this.pounds + " lbs., " + this.ounces + " oz.";
    } // toString()

    public static void main( String [] args ) {
        Weight kiwis = new Weight( 1, 8 );
        Weight plums = new Weight( 2, 12 );

        Weight sum = kiwis.add( plums );

        System.out.println( kiwis + " + " + plums + " = " + sum );
    } // main( String [] )
} // Weight
```


Lesson 7

“Who cares about design unless people want to use it?”

—Linus Torvalds [16, page 80]

“Thus computing is, or at least should be, intimately bound up with both the source of the problem and the use that is going to be made of the answers—it is not a step to be taken in isolation from reality.”

—Richard W. Hamming [11, page 3]

7.1 Video clip

- [IBM Watson: Final Jeopardy! and the Future of Watson](#)

7.2 Exercises

1. In which city in the United States are the largest and second largest airports named, respectively, after a hero of the Second World War and after a great battle of the Second World War?

2. What is the relevance of the previous question about airports to our study of computer science?
3. The world is full of opportunities for curious and energetic people. Most careers are zig-zags, not straight lines. Accidents as much as plans determine most careers. You just never know where you might wind up.

What do Jeff Bates, Matt Harding, Rob Malda, and Eric Weinstein have in common? (Look up the names on the Web!)

4. The Java compiler will reject this code. Fix the code.

```

if( 70 < temp < 75 ) {
    System.out.println( "The room is comfortable." );
} // if

```

5. Computer scientists count from zero. This means that the index of the last element in an array of N items has what value?
6. An array is a homogeneous data structure. A personnel record will probably be a heterogeneous data structure.

What does the word “homogeneous” mean in this context?

7. What is missing in the following code?

```

int [] data;
data[0] = 42;

```

8. Suppose that this next example is an excerpt from a program that computes and reports statistics related to the weather. Suppose that the program in its current form analyzes data for just one year.

A programmer might later adapt the program so that it analyzes the data in a leap year. There are 366 days, not 365 days, in a leap year. Or a programmer might modify the program to work with a measurements collected over decades instead of over a single year.

Replace $i < 365$ in the following code with an expression so that the program will still print the high temperature for every day

on which data was recorded even if the programmer changes the size of the data set.

```

double [] temps = new double[365];
for( int i = 0; i < 365; i++ ) {
    System.out.println( "high temperature on day " + i
        " = " + temps[i] );
} // for

```

9. Most **for** loops have a very similar appearance. However, the Java programming language allows many variations.

What is the output of this fragment of a program?

```

for( int i = 0; i < 20; i += 2 ) {
    System.out.print( i + " " );
} // for
System.out.println();

```

10. When they need to sort data, software engineers most often make use of methods that they find in libraries of software. They rarely write their own methods for sorting data. Still, there are good reasons for studying sorting algorithms. What are they?
11. The number of instructions that a computer executes when sorting a list of N items with the selection sort algorithm is proportional to what function of N ?
12. The number of instructions that a computer executes when sorting a list of N items with the insertion sort algorithm is comparable to the number executed when using the selection sort algorithm in the average and worst cases. In the best case, the performance of the insertion sort can be much better than the performance of the selection sort. What is the best case?
13. What is the unit of time that is most relevant to a discussion of events on the microprocessor in your computer?
14. What is the unit of length that is most relevant to a discussion of the transistors and the conducting paths that connect transistors on the microprocessor in your computer?

15. Write code that creates an array of floating point numbers.
16. Write code that assigns a random value in the range 0.0 to 1.0 to each element of an array of floating point numbers.
17. Write code that finds the value of the smallest value in an array of floating point numbers.
18. Write code that finds the position (that is, the index) of the smallest value in an array of floating point numbers.
19. Write code that finds the length of the longest sequence of ascending values in an array of floating point numbers.
20. Write code that prints all pairs of integers (a, b) , where $0 \leq a < N$, $0 \leq b < N$, and $a \neq b$.

Lesson 8

“Seeing things from a different angle was another strength, and he sometimes made dramatic demonstrations of this. A gymnast in college, Faget liked to leap over chairs in conference rooms or to stand on his head ‘to improve blood circulation in my brain,’ as he put it. With keys and coins falling out of his pockets, Faget would calmly discuss the engineering questions on the agenda.”

—James Oberg [18, page 62]

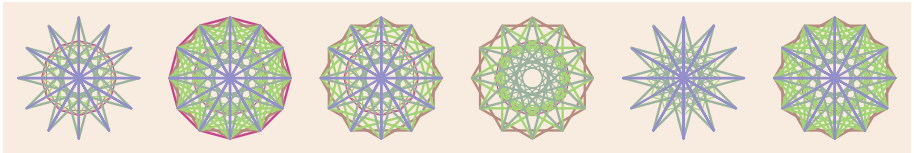
“Karl Heimburg, chief of the Testing Lab, explained how von Braun went around looking for new ideas, which he would then take to his associates. Heimburg would often listen unimpressed and explain to von Braun why something wouldn’t work. And then Heimburg would find himself saying, ‘But we could do it in this other way,’ and an innovation that he had not considered would have opened up before him.”

—Bruce Murray [17, page 52]

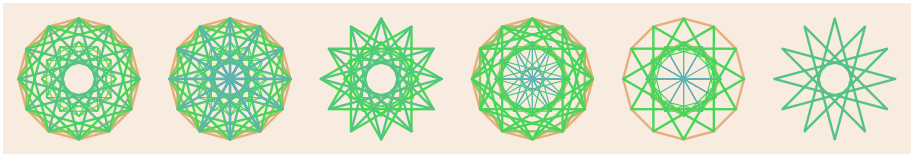
8.1 Exercises

1. Write a method that, given an array of floating point values, returns the sum of those values.
2. Write a method that, given an array of floating point values and a floating point value x , returns an array in which the i^{th} element equals the i^{th} element of the given array divided by x .
3. Write a method that, given an array of floating point values, returns an array in which the i^{th} element equals the sum of the first i elements of the given array.
4. Write a method that, given an array of floating point values and a floating point value x , returns the index of the first element in the array whose value is greater than x . If the array does not contain an element whose value is greater than x , then the array should return the length of the array.

Part III



Algorithms



Lesson 9

“Because there is a certain inevitability about inventions of this sort, which have often been anticipated by futurists, the real contribution lies, first, in making the idea work. What is needed next is believing in it passionately enough to supply the emotional energy to carry on the inevitable struggle until the new idea is firmly rooted in the world and has taken on a life of its own. It is a work of intellect and love.”

—Federico Faggin [citesources](#)[page 102]gilder

“When I finally quit, I felt weary in my bones. I was actually sweating; my shirt stuck to my back. Things around me kept going in and out of focus. I looked at Alsing, and the rims of his eyes were red. He said he could remember experiencing weariness like this during his midnight-programming days, but he had been younger then. Weariness had been a badge and a part of the fun.”

—Tracy Kidder [[32](#), page 127]

In this lesson, we will learn how to simulate random events on a computer. Random events arise in games. For example, players may draw cards from a shuffled deck or throw dice to determine their next move.

A discrete probability distribution function associates a probability with each event in a finite set of possible events. Each probability is a number in the range 0.0 to 1.0. The sum of all probabilities must equal 1.0. This is so because on each trial exactly one of the events must occur. A tossed coin will either land heads up or tails up. The probability of heads plus the probability of tails must equal one—we are certain that one of the two possible outcomes will occur.

Let us think about what can happen when we throw a pair of dice.

Each die has six sides. If the dice are fair, the probabilities of landing with any given side facing up are equal. When the dice land, any one of six numbers could be on the top of the first die and any of six numbers could be on top of the second die—there are 36 possible combinations. Again, if the dice are fair, each of these 36 possible combinations is equally likely.

The sum of numbers on top of the two dice will be in the range 2 to 12. The probability of getting a given sum will depend upon how many combinations can produce the sum. There is only one combination that can produce a sum of 2 (both dice must land with one on top) but there are six combinations that will sum to 7. The probability of rolling a 7 exceeds the probability of rolling a 2 (“snake eyes”).

Figure 9.1 shows the combinations and their sums. In this figure, (i, j) means that the first die lands with i showing on the top face and the second die lands with j showing on its top.

Figure 9.2 shows the probability distribution function. That is, it gives the probabilities of sum with a throw.

The probability that a throw of the dice will yield 2 is about 0.0277. The probability that a throw will yield 3 is about 0.0555. It follows that the probability of getting 3 or less is $0.0277 + 0.0555 = 0.0833$.

Similarly, the probability that a throw will yield 4 or less is $0.0277 + 0.0555 + 0.0833 = 0.1666$.

sum	combinations that produce sum					
2	(1,1)					
3	(1,2)	(2,1)				
4	(1,3)	(3,1)	(2,2)			
5	(1,4)	(4,1)	(2,3)	(3,2)		
6	(1,5)	(5,1)	(2,4)	(4,2)	(3,3)	
7	(1,6)	(6,1)	(2,5)	(5,2)	(3,4)	(4,3)
8	(2,6)	(6,2)	(3,5)	(5,3)	(4,4)	
9	(3,6)	(6,3)	(4,5)	(5,4)		
10	(4,6)	(6,4)	(5,5)			
11	(5,6)	(6,5)				
12	(6,6)					

Figure 9.1: Ways to get each possible result in a throw of a pair of dice

Figure 9.3 tabulates the probabilities of outcomes less than or equal to k for all values of k from 2 to 12. It is, of course, certain that a throw of dice will produce an outcome that is less than or equal to one.

9.1 Exercises

These questions refer to the program that follows.

1. The `makeArray()` method returns an array of floating point numbers.
 - (a) How many numbers will the array that `makeArray` returns to its caller contain?
 - (b) What is the range of possible values for each element of the array?

```
import java.util.Random;
```

sum	# combinations that produce sum	probability of sum
2	1	$\frac{1.0}{36.0} \approx 0.0277$
3	2	$\frac{2.0}{36.0} \approx 0.0555$
4	3	$\frac{3.0}{36.0} \approx 0.0833$
5	4	$\frac{4.0}{36.0} \approx 0.1111$
6	5	$\frac{5.0}{36.0} \approx 0.1388$
7	6	$\frac{6.0}{36.0} \approx 0.1666$
8	5	$\frac{5.0}{36.0} \approx 0.1388$
9	4	$\frac{4.0}{36.0} \approx 0.1111$
10	3	$\frac{3.0}{36.0} \approx 0.0833$
11	2	$\frac{2.0}{36.0} \approx 0.0555$
12	1	$\frac{1.0}{36.0} \approx 0.0277$

Figure 9.2: Probabilities of each possible combination

event	probability
$sum \leq 2$	$\frac{1}{36} \approx 0.0277$
$sum \leq 3$	$\frac{1}{36} + \frac{2}{36} \approx 0.0833$
$sum \leq 4$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} \approx 0.1666$
$sum \leq 5$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} + \frac{4}{36} \approx 0.2777$
$sum \leq 6$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} + \frac{4}{36} + \frac{5}{36} \approx 0.4166$
$sum \leq 7$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} + \frac{4}{36} + \frac{5}{36} + \frac{6}{36} \approx 0.5833$
$sum \leq 8$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} + \frac{4}{36} + \frac{5}{36} + \frac{6}{36} + \frac{5}{36} \approx 0.7222$
$sum \leq 9$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} + \frac{4}{36} + \frac{5}{36} + \frac{6}{36} + \frac{5}{36} + \frac{4}{36} \approx 0.8333$
$sum \leq 10$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} + \frac{4}{36} + \frac{5}{36} + \frac{6}{36} + \frac{5}{36} + \frac{4}{36} + \frac{3}{36} \approx 0.9166$
$sum \leq 11$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} + \frac{4}{36} + \frac{5}{36} + \frac{6}{36} + \frac{5}{36} + \frac{4}{36} + \frac{3}{36} + \frac{2}{36} \approx 0.9722$
$sum \leq 12$	$\frac{1}{36} + \frac{2}{36} + \frac{3}{36} + \frac{4}{36} + \frac{5}{36} + \frac{6}{36} + \frac{5}{36} + \frac{4}{36} + \frac{3}{36} + \frac{2}{36} + \frac{1}{36} = 1.0$

Figure 9.3: Cumulative probability distribution for throw of a pair of dice.

```

public class Loops {
    public static final int N = 12;
    public Loops() {
        double [] pdf = makeArray( N );
        normalizeArray( pdf );
        printArray( pdf );
        double [] cdf = accumulate( pdf );
        printArray( cdf );
        int [] histogram = new int[ cdf.length ];
        for( int i = 0; i < 10000; i++ ) {
            int event = randomEvent( cdf );
            histogram[event]++;
        } // for
        for( int i = 0; i < histogram.length; i++ ) {
            System.out.print( histogram[i] + " " );
        } // for
        System.out.println();
    } // Loops()

    private double [] makeArray(int size) {
        double [] result = new double[ size ];
        for( int i = 0; i < result.length; i++ ) {
            result[i] = Math.random();
        } // for
        return result;
    } // makeArray( int )

    private double sumArray( double [] data ) {
        double sum = 0.0;
        for( int i = 0; i < data.length; i++ ) {
            sum += data[i];
        } // for
        return sum;
    } // sumArray( double [] )

    private void normalizeArray( double [] data ) {
        double sum = sumArray( data );
        for( int i = 0; i < data.length; i++ ) {

```

```

        data[i] /= sum;
    } // for
} // normalizeArray( double [] )

private double [] accumulate( double [] data ) {
    double [] result = new double[ data.length ];
    double sum = 0.0 ;
    for( int i = 0; i < data.length; i++ ) {
        result[i] = data[i] + sum;
        sum += data[i];
    } // for
    return result;
} // accumulate( double [] )

private int positionOfFirstLargerValue( double [] values ,
    double threshold ) {
    int index = 0;
    while( index < values.length && values[index] < threshold )
        index++;
    } // while
    return index;
} // positionOfFirstLargerValue( double [], double )

private int randomEvent( double [] cdf ) {
    double randomNumber = Math.random();
    return positionOfFirstLargerValue( cdf, randomNumber );
} // randomEvent( double [] )

private void printArray( double [] data ) {
    for( double x : data ) {
        System.out.format( "%5.2f ", x );
    } // for
    System.out.println();
} // printArray( double [] )

public static void main( String [] args ) {
    Loops loops = new Loops();
} // main( String [] )

```

} // *Loops*

Lesson 10

“No experiment is a failure—even one that doesn’t work. You learn something from them all.”

—Harold E. Edgerton

“Success is the ability to go from failure to failure with no loss of enthusiasm.”

—Winston Churchill

1. IBM built and programmed a remarkable computer that beat the best human players at the game of *Jeopardy!* The company has high hopes for the technology it demonstrated in that game. IBM is using the same system now to solve problems that arise in medical care.

The company named the computer after a person who was very important in the company’s history. What is the name of the machine?

2. Let’s sum the first 8 non-negative integers. We can make the work easier by rearranging and pairing the numbers:

$$\begin{aligned}
 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 &= (0 + 7) + (1 + 6) + (2 + 5) + (3 + 4) \\
 &= 4 \cdot 7 \\
 &= \frac{8}{2} \cdot (8 - 1) \\
 &= 28
 \end{aligned}$$

What is the sum of the first N non-negative integers?

3. Here is an 8×8 array. The elements on the diagonal are labeled “D.” The elements in the lower triangular part of the array (below the diagonal) are labeled “L” and those in the upper triangular array (above the diagonal) are labeled “U.”

D	U	U	U	U	U	U	U
L	D	U	U	U	U	U	U
L	L	D	U	U	U	U	U
L	L	L	D	U	U	U	U
L	L	L	L	D	U	U	U
L	L	L	L	L	D	U	U
L	L	L	L	L	L	D	U
L	L	L	L	L	L	L	D

Let’s count the number of L’s in the array.

Here are two observations that we can use to make the counting easier:

- The array contains $8 \times 8 = 64$ elements. Subtract the 8 diagonal elements and divide what is left by 2 (because what is left contains equal numbers of L’s and U’s) to get the number of L’s.
- The array contains one row that has 0 L’s, one row that has 1 L, one row that has 2 L’s, and so on to the bottom row, which contains 7 L’s.

How many L’s are in the array?

4. Here is a trace of a selection sort. The i^{th} row in this table shows the list that is being sorted after the i^{th} step in the execution of the algorithm. After the i^{th} step, i elements of the list are in order—those elements are printed in boldface type. At each step, the algorithm searches through the unsorted part of the list (the part that is not printed in boldface type) to find the smallest number in that unsorted part. In each search, the algorithm examines all elements in the unsorted part of the list.

How many times does the algorithm examine an element of the list? (Count the numbers in this table that are not printed in boldface type.)

87	32	80	39	15	26	58	70
15	32	80	39	87	26	58	70
15	26	80	39	87	32	58	70
15	26	32	39	87	80	58	70
15	26	32	39	87	80	58	70
15	26	32	39	58	80	87	70
15	26	32	39	58	70	87	80
15	26	32	39	58	70	80	87
15	26	32	39	58	70	80	87

5. Here is a trace of an insertion sort. The i^{th} row in this table shows the list that is being sorted after the i^{th} step in the execution of the algorithm. After the i^{th} step, i elements of the list are in order—those elements are printed in boldface type. At each step, the algorithm searches through the sorted part of the list (the part that is printed in boldface type) to find where it must place the next number from the unsorted part. In the worst case, in each search the algorithm must examine all elements in the sorted part of the list.

How many times does the algorithm examine an element of the list? (Count the numbers in this table that are printed in boldface type.)

42 64 23 43 17 67 19 68
42 64 23 43 17 67 19 68
42 **64** 23 43 17 67 19 68
23 **42** **64** 43 17 67 19 68
23 **42** **43** **64** 17 67 19 68
17 **23** **42** **43** **64** 67 19 68
17 **23** **42** **43** **64** **67** 19 68
17 **19** **23** **42** **43** **64** **67** 68
17 **19** **23** **42** **43** **64** **67** **68**

6. My favorite definition of computer science answers the question of “What is computer science?” with four more questions:
- What kinds of problems can be solved by writing computer programs?
 - Given a solvable problem, how do we write the program?
 - Given several programs that solve the same problem, how do we choose the best program?
 - Given a claim that a program solves a problem, how can we assure ourselves that it really does solve the problem in all cases?

How does an exercise in counting the number of operations required for sorting a list with the selection sort or insertion sort algorithms relate to this definition?

7. You learned that Eric Weisstein created *www.mathworld.com*, an on-line encyclopedia of mathematics. You learned that Rob Malda and Jeff Bates created *www.slashdot.com* to share news about developments in science and technology. You learned that Matt Harding danced with people in many countries and posted videorecordings of the dancing on *www.wherethehellismatt.com*.

What lesson did you draw from these discoveries?

8. The arithmetic mean of two numbers a and b is $(a + b)/2$. The geometric mean of a and b is $\sqrt{ab} = (ab)^{1/2}$. The harmonic mean of a and b is $2/(1/a + 1/b)$.

It is possible to compute arithmetic, geometric, and harmonic means not only for two numbers, but also for three, four, or many numbers.

Here are methods that compute arithmetic, geometric, and harmonic means. Label each method with the name of the mean it computes and with the value that it returns when $\text{data} = \{2, 4, 8\}$.

- (a) **public double** f1(**double** [] data) {
 double product = 1.0;

 for(**int** i = 0; i < data.length; i++) {
 product = product * data[i];
 } // for

 return Math.pow(product , 1.0/data.length);
} // f1(double [])
- (b) **public double** f0(**double** [] data) {
 double sum = 0.0;

 for(**int** i = 0; i < data.length; i++) {
 sum = sum + data[i];
 } // for

 return sum / data.length;
} // f0(double [])
- (c) **public double** f2(**double** [] data) {
 double sum = 0.0;

 for(**int** i = 0; i < data.length; i++) {
 sum = sum + 1.0 / data[i];
 } // for

 return data.length / sum;
} // f2(double [])

9. Let $\text{data} = \{ 17, 11, 19, 41, 43, 13, 31, 29 \}$.

- (a) What value does $f(\text{data})$ return?
 (b) What value does $g(\text{data})$ return?

```

public double f( double [] data ) {
    double bestGuessSoFar = data[0];

    for( int i = 0; i < data.length; i++ ) {
        if( data[i] < bestGuessSoFar ) {
            bestGuessSoFar = data[i];
        } // if
    } // for

    return bestGuessSoFar;
} // f( double [] )

public int g( double [] data ) {
    int bestGuessSoFar = 0;

    for( int i = 0; i < data.length; i++ ) {
        if( data[i] < data[bestGuessSoFar] ) {
            bestGuessSoFar = i;
        } // if
    } // for

    return bestGuessSoFar;
} // g( double [] )

```

10. Complete this stub.

```

// Count the number of perfect scores in
// a list of grades.
// Each grade is a number in the inclusive
// range 0 to 100.
// A perfect score is, of course, 100.
public int countPerfectScores( int [] grades ) {
    return grades.length;
} // countPerfectScores( int [] )

```

11. Suppose that a program includes code that creates a 4×4 array m , initializes it with the values shown, and then calls $f(m)$. The method will change the array. What will the array look like after the call to $f(m)$?

$$m = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

```
public void f( int [][] m ) {
    for( int i = 0; i < m.length; i++ ) {
        for( int j = 0; j < i; j++ ) {
            int temp = m[i][j];
            m[i][j] = m[j][i];
            m[j][i] = temp;
        } // for
    } // for
} // f( int [] )
```

12. Write code that prints the value of $a^3 + b^3$ for all $0 \leq a < N$ and $0 \leq b < N$.

Lesson 11

“You will never remember the many times the launch slipped, but the on-time failures are with you always.”

—Walt Williams [15, page 100]

“Checking doesn’t take place at any one point in a problem-solving situation. Checking permeates the whole procedure.”

—John R. Dixon [6, page 158]

1. Write code that creates an array of integers that is just big enough to hold all of the integers in the arrays a and b.

```
int [] a = {0, 2, 4, 6};  
int [] b = {1, 3, 5, 7};
```

2. Change this code so that it prints the contents of the array on a single line with a space between each pair of consecutive values.

```
int [] c = {0, 1, 2, 3, 4, 5, 6, 7};  
  
for( int i = 0; i < c.length; i++ ) {  
    System.out.println( c[i] );  
} // for
```

3. We can represent a probability distribution function (pdf) with an array. Similarly, we can represent a cumulative distribution function (cdf) with an array.

The values of the elements of the *pdf* are the probabilities of various events. In this example, there are four possible events: Event 0, Event 1, Event 2, and Event 3.

```
double [] pdf = {0.1, 0.2, 0.3, 0.4};
double [] cdf = {0.1, 0.3, 0.6, 1.0};
```

Here is the relationship between the cumulative distribution function and the probability distribution function.

$$\begin{aligned}cdf[0] &= pdf[0] \\cdf[1] &= pdf[1] + pdf[0] \\cdf[2] &= pdf[2] + pdf[1] + pdf[0] \\cdf[3] &= pdf[3] + pdf[2] + pdf[1] + pdf[0]\end{aligned}$$

The rule can be generalized for larger arrays. The value of the i^{th} element of *cdf* is the sum of the values of all elements of *pdf* whose index is less than or equal to i .

$$cdf[i] = \sum_{j=0}^i pdf[j]$$

Complete this stub.

```
// Make a cdf from a pdf.
public double [] makeCDF( double [] pdf ) {
    double [] cdf = new double[ pdf.length ];

    // Add code here.

    return cdf;
} // makeCDF( double [] )
```

4. Let *cdf* represent a cumulative distribution function.

$$cdf = \{0.1, 0.3, 0.6, 1.0\}$$

In this example...

- The probability that the next event will be Event 0 is 0.1.
- The probability that the next event will be either Event 0 or Event 1 is 0.3.
- The probability that the next event will be either Event 0, Event 1, or Event 2 is 0.6.
- The probability that the next event will be either Event 0, Event 1, Event 2, or Event 3 is 1.0. (There are only four possible outcomes in this example—a probability of 1.0 signifies the certainty of choosing one of the events.)

Complete this stub.

```
// Generate a random event with probabilities
// described by a given function.
public int randomEvent( double [] cdf ) {
    int index = 0;
    double r = Math.random();

    // Find the index of the first element
    // of cdf whose value is greater than r.

    return index;
} // randomEvent( double [] )
```

5. It is possible to simulate the behavior of people browsing on the Web with a Markov process. At the heart of the algorithm is the repeated multiplication of a matrix *m* times a vector *v*.

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} m_{00} \cdot v_0 + m_{01} \cdot v_1 + m_{02} \cdot v_2 + m_{03} \cdot v_3 \\ m_{10} \cdot v_0 + m_{11} \cdot v_1 + m_{12} \cdot v_2 + m_{13} \cdot v_3 \\ m_{20} \cdot v_0 + m_{21} \cdot v_1 + m_{22} \cdot v_2 + m_{23} \cdot v_3 \\ m_{30} \cdot v_0 + m_{31} \cdot v_1 + m_{32} \cdot v_2 + m_{33} \cdot v_3 \end{bmatrix}$$

Each element of the product is a sum of products. Each of the products in the sum is obtained by multiplying corresponding elements in a row of m and those in \vec{v} .

Complete this stub.

```
// Compute the sum of the products of corresponding
// elements in two arrays.
public double multiply( double [] row, double [] col
} // multiply( double [], double [] )
```

6. In the following code...

- (a) Circle and label an instance variable.
- (b) Circle and label a local variable.
- (c) Circle and label a constant.
- (d) Circle and label a constructor.
- (e) Circle and label a method that overrides the definition of an inherited method.
- (f) Circle and label an accessor method.
- (g) Circle and label the return type of a method.
- (h) Circle and label the formal parameter of a method.

```
public class Time {
    public static final int MINUTES_IN_AN_HOUR = 60;

    private int hours;
    private int minutes;

    public Time( int hours, int minutes ) {
        this.hours = hours;
        this.minutes = minutes;
    } // Time( int, int )

    public Time add( Time anotherTime ) {
        int h = this.getHours() + anotherTime.getHours()
        int m = this.getMinutes() + anotherTime.getMinu
```



```

        h = h + m / MINUTES_IN_AN_HOUR;
        m = m % MINUTES_IN_AN_HOUR;
        return new Time( h, m );
    } // add( Time )

    public int getHours() {
        return this.hours;
    } // getHours()

    public int getMinutes() {
        return this.minutes;
    } // getMinutes()

    public String toString() {
        String result = "";
        result += this.getHours() + " hours, ";
        result += this.getMinutes() + " minutes";
        return result;
    } // toString()
} // Time

```

7. Write code that adds 3 hours, 40 minutes to 4 hours, 30 minutes. Use the Time class.
8. Describe what is missing in the following code. It is supposed to merge the contents of a and b but does not finish the job.

It will be sufficient to describe what the missing code must do. You do not have to write the missing code.

```

// This code illustrates the merge operation
// that is at the heart of a merge sort.

// Declare, create, and initialize an array.
// Fill it with an ordered sequence of integers.
int [] a = { 2, 5, 19, 37, 42, 49, 60, 99 };

// Declare, create, and initialize an array.
// Fill it with an ordered sequence of integers.

```

```

int b = { 11, 13, 17, 24, 36, 61, 72, 108 };

// Declare and create an array that
// is just big enough to hold all of the
// elements of a and all of the elements of b.
int [] c = new int[ a.length + b.length ];

// i is an index to an element in a.
int i = 0;
// j is an index to an element in b.
int j = 0;
// k is an index to an element in c.
int k = 0;

// Repeatedly select one element from
// either a or b and copy its value into c.
while( i < a.length && j < b.length ) {
    // Compare the next element in a that
    // has not been copied into c with the
    // next element in b that has not been
    // copied into c.
    if( a[i] < b[j] ) {
        // The element in a is the smaller
        // of the 2 elements—copy it into c.
        c[k] = a[i];
        // Increment the index for a—that is,
        // point to the next element in a.
        i++;
    } // if
    else {
        // The element in b is the smaller
        // of the 2 elements—copy it into c.
        c[k] = b[j];
        // Increment the index for b—that is,
        // point to the next element in b.
        j++;
    }
    k++;
}

```

```

} // while

// More code needed here...

```

9. How can you tell that this method is recursive?

```

private void flood(int x, int y,
                  int xMin, int yMin, int xMax, int yMax) {

    boolean xInBounds = false;
    if (x > xMin && x < xMax) {
        xInBounds = true;
    } // if

    boolean yInBounds = yMin < y && y < yMax;

    if (xInBounds && yInBounds && !cells[x][y]) {
        cells[x][y] = true;

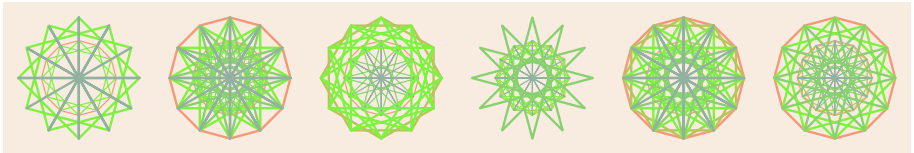
        flood(x + 1, y, xMin, yMin, xMax, yMax);
        flood(x, y + 1, xMin, yMin, xMax, yMax);
        flood(x - 1, y, xMin, yMin, xMax, yMax);
        flood(x, y - 1, xMin, yMin, xMax, yMax);
    } // if
} // flood( int, int, int, int, int, int )

```

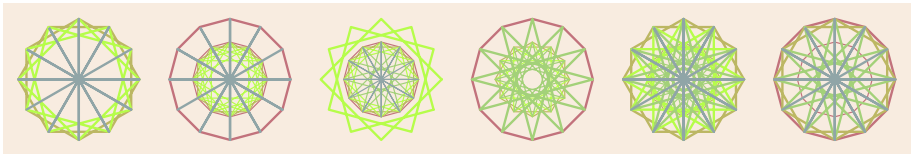
10. Suppose that you have written a collection of Java classes to help other programmers with their projects. How might you distribute your software?

(Hint: In what form did the College Board distribute the classes that Advanced Placement students need for exercises with the GridWorld Case Study? What kind of file did you download from our course's Moodle site so that you could complete exercises that Sedgewick and Wayne included in our textbook?)

Part IV



Images



Lesson 12

“Whereas the difference between poor conceptual designs and good ones may lie in the soundness of design method, the difference between good designs and great ones surely does not. Great designs come from great designers. Software construction is a creative process. Sound methodology can empower and liberate the creative mind; it cannot inflame or inspire the drudge.”

—Frederick P. Brooks [3, page 18]

“To make it work they’d have to hire the very best new engineers they could find, ones who would know more about the state of the art in computers than they did. They told each other that they’d have to be sure not to turn away candidates just because the youngsters made them feel old and obsolete; on the contrary, those were the candidates they’d have to welcome.”

—Tracy Kidder [32, page 78]

1. Supply the missing **if** statement to complete the definition of this method.

```
public double findMinimum( double [] data ) {
```

```

double bestGuessSoFar = data[0];

for( int i = 1; i < data.length; i++ ) {

    // Need if statement here.

} // for

return bestGuessSoFar;
} // findMinimum( int [] )

```

2. Supply the missing assignment statement to complete the definition of this method.

```

public int findPositionOfMinimum( double [] data ) {
    int bestGuessSoFar = 0;

    for( int i = 1; i < data.length; i++ ) {
        if( data[i] < data[bestGuessSoFar] ) {

            // Need assignment statement here.

        } // if
    } // for

    return bestGuessSoFar;
} // findPositionOfMinimum( double [] )

```

3. Write a method that returns **true** if a given array of integers contains at least one element with a specified value and returns **false** otherwise.

Use this method as a model. You can solve this problem by making small changes to this method.

```

public int countMatches( int [] data, int valueToMatch
    int count = 0;

    for( int i = 0; i < data.length; i++ ) {

```



```

        if( data[i] == valueToMatch ) {
            count++;
        } // if
    } // if

    return count;
} // countMatches( int [], int )

```

4. Complete the **for** loop in this method. Be careful! The answer does not begin with **int** `i = 0`.

```

// Return true if each element in data
// is less than or equal to the element
// that follows.
// Return false otherwise.
public boolean inOrder( int [] data ) {
    boolean result = true;

    for(
        ) {
        result = result && data[i - 1] <= data[i];
    } //

    return result;
} // inOrder( int [] )

```

5. Write a method that computes the weighted average of two floating point numbers. This method will have three parameters: the two numbers to be averaged and the weight. The weight will be a value in the range 0.0 to 1.0. The method will return a floating point value to its caller.

If a and b are the numbers to be averaged and t is the weight, then the weighted average is...

$$\text{weighted - average} = (1 - t) \cdot a + t \cdot b$$

6. I wrote a class that models fractions and then scrambled the order of the lines of code. Reorder these lines to make the class named `Rational`.

```

line # 0:    } // Rational( int, int )
line # 1:  public class Rational {
line # 2:    private int num;
line # 3:      return a;
line # 4:      if( b == 0 ) {
line # 5:      } // if
line # 6:    } // gcd( int, int )
line # 7:    private int den;
line # 8:      return new Rational( n, d );
line # 9:    } // toString()
line #10:      this.den = den/gcd;
line #11:  } // Rational
line #12:  public Rational( int num, int den ) {
line #13:  public String toString() {
line #14:  public Rational add( Rational r ) {
line #15:    } // else
line #16:      int d = this.den * r.den;
line #17:      this.num = num/gcd;
line #18:  private int gcd( int a, int b ) {
line #19:  } // add( Rational )
line #20:    else {
line #21:      int gcd = gcd( num, den );
line #22:      int n = this.num * r.den + this.den * r.n
line #23:      return gcd( b, a % b );
line #24:      return num + "/" + den;

```

7. This code creates two instances of the Rational classes, computes the sum that the two objects represents, and then prints two identical lines. Explain why the two lines are identical? What is the toString() method?

```

Rational a = new Rational( 1, 2 );
Rational b = new Rational( 1, 6 );
Rational sum = a.add( b );
System.out.println( a.toString() + " + " + b.toString()
    + " = " + sum.toString() );
System.out.println( a + " + " + b + " = " + sum );

```

Write your answer here.

8. We can prove that...

$$\begin{aligned}\gcd(a, b) &= \gcd(b, a \bmod b) \\ \gcd(a, 0) &= a\end{aligned}$$

Here is an example of how Euclid's Algorithm can be used to find the greatest common divisor of two integers.

$$\begin{aligned}\gcd(102, 68) &= \gcd(68, 102 \bmod 68) \\ &= \gcd(68, 34) \\ &= \gcd(34, 68 \bmod 34) \\ &= \gcd(34, 0) \\ &= 34\end{aligned}$$

This algorithm is recursive.

We know that...

$$\begin{aligned}a^n &= a^{\frac{n}{2}} \cdot a^{\frac{n}{2}} && \text{when } n \text{ is even} \\ a^n &= a \cdot a^{\frac{n}{2}} \cdot a^{\frac{n}{2}} && \text{when } n \text{ is odd} \\ a^1 &= a \\ a^0 &= 1\end{aligned}$$

Here is an example of how to raise a number to a power recursively.

$$2^{16} = 2^8 \cdot 2^8$$

$$2^8 = 2^4 \cdot 2^4$$

$$2^4 = 2^2 \cdot 2^2$$

$$2^2 = 2^1 \cdot 2^1$$

$$2^1 = 2$$

What is gained by using recursion in these two cases?

9. Which of the three sorting algorithms that we examined is recursive?
10. Write a paragraph or two in response to the question: Is computer science a science? Provide an argument in support of your answer.

You might choose to refer to problems you encountered in this course or exercises you completed. Was our work like work that scientists do?

You might find evidence to support your answer in the lives and work of one or more of the computer scientists whom you met through your reading and our discussions. Were their methods or goals like those of scientists?

Lesson 13

“It is especially nice when the things we regard as beautiful are also regarded by other people as useful.”

—Donald E. Knuth [14, page 671]

“Beauty is important in engineering terms because software is so complicated. Complexity makes programs hard to build and potentially hard to use; beauty is the ultimate defense against complexity.”

—David Gelernter [27, page 22]

1. My fourth grade teacher asked me and my classmates to write book reports. She asked us to include in our reports definitions of new words that we had encountered in our reading. She forbade us from defining a word using that same word.

Now many years later I am a grown-up computer scientist and know that I can define a method by using the same method. What word names this problem solving technique?

2. Here is a recursive method:

```
public int factorial( int n ) {  
    if( n == 0 ) {
```

```

        return 1;
    } // if
    else {
        return n * factorial( n - 1 );
    } // else
} // factorial( int )

```

- (a) What is the value of `factorial(4)`?
- (b) For which values of n will the method return $n!$?
- (c) Every recursive method contains an **if** statement (or something equivalent). Why?

3. Here is an iterative method:

```

public int factorial( int n ) {
    int product = 1;
    for( int i = 1; i <= n; i++ ) {
        product *= i;
    } // for
    return product;
} // factorial( int )

```

How does the number of multiplications in an execution of the recursive method compare to the number in an execution of the iterative method?

4. Here are two ways that a teacher of mathematics might present a definition of factorial to students. Which do you prefer? Why?

- (a) “To compute factorial of n when n is a positive integer, multiply together all of the integers from 1 to n . Let’s also agree that the factorial of 0 will be 1.”
- (b)

$$\begin{aligned}
 0! &= 1 \\
 1! &= 1 \\
 n! &= n \cdot (n-1)! \quad (\text{for } n > 1)
 \end{aligned}$$

5. Why might you prefer the second of the two versions of `powerOf2()` shown here?

```

// Version #1
public int powerOf2( int exponent ) {
    int power = 1;

    for( int i = 0; i < exponent; i++ ) {
        power *= 2;
    } // for

    return power;
} // powerOf2( int )

// Version #2
public int powerOf2( int exponent ) {
    if( exponent == 0 ) {
        return 1;
    } // if
    else {
        int product = powerOf2( exponent/2 );
        if( exponent % 2 == 0 ) {
            return product * product;
        } // if
        else {
            return 2 * product * product;
        } // else
    } // else
} // powerOf2( int )

```

6. Refer again to version #2 of `powerOf2()`.
- (a) In English, what is the meaning and purpose of the expression `exponent % 2 == 0` ?
7. Write a method that computes x^n for any floating point number x and any non-negative integer n ?
8. Write a method that computes x^n for any floating point number x and **any** integer n ?

9. Let a and b be non-negative integers. Let q be the quotient obtained by dividing a by b . Let r be the remainder obtained by dividing a by b .

- (a) Write an expression with q , b , and r that completes this equation:

$$a =$$

- (b) a and b have a greatest common divisor. Call that divisor d . To say that d is a divisor of a is to say that d divides into a without remainder. Similarly d divides into b without remainder. Let us call the quotients m and n :

$$a = m \cdot d$$

$$b = n \cdot d$$

Rewrite the equation that you wrote for the first part of this question using md in place of a and nd in place of b . Regroup terms to show that if d is a divisor of a and b then it is also a divisor of r .

- (c) Write a method that finds the greatest common divisor of two non-negative integers a and b by testing every possible divisor from 1 up to the smaller a and b .
10. Compute $\text{gcd}(192, 42)$ by using the following definition of the greatest common divisor function and showing the values computed at each step.

$$\text{gcd}(a, 0) = a$$

$$\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$$

11. The Fibonacci sequence begins: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

After we get past the first two elements, each element is the sum of the two previous elements.

$$\begin{aligned}
 fib(0) &= 1 \\
 fib(1) &= 1 \\
 fib(n) &= fib(n-1) + fib(n-2) \quad (\text{for } n > 1)
 \end{aligned}$$

Now let's compute $fib(5)$.

$$\begin{aligned}
 fib(5) &= fib(4) + fib(3) \\
 &= (fib(3) + fib(2)) + (fib(2) + fib(1)) \\
 &= ((fib(2) + fib(1)) + (fib(1) + fib(2))) + ((fib(1) + fib(0)) + fib(1))
 \end{aligned}$$

Imagine computing $fib(n)$ for some large value of n . What will happen? What's the problem here?

12. We can use Newton's Method to find the roots of equations. In particular, we can find the square root of x by finding the positive root of $x - root^2 = 0$. (Here x is a given number and $root$ is the unknown value that we are trying to find.)

Newton's Method invites us to guess the answer. Then it gives us a means of using that guess to construct an even better guess. By repeating the process, we can generate successively better approximations of the root we seek.

Let's use a variable named *approx* to hold the value of our current approximation of the square root of a number x .

- If *approx* is greater than the square root of x , then x/\textit{approx} is smaller than the square root.
- If *approx* is less than the square root of x , then x/\textit{approx} is greater than the square root of x .

It follows that the square root lies between *approx* and x/\textit{approx} .

Explain what happens in the execution of this method:

```
private static final double EPSILON = 1E-8;
```

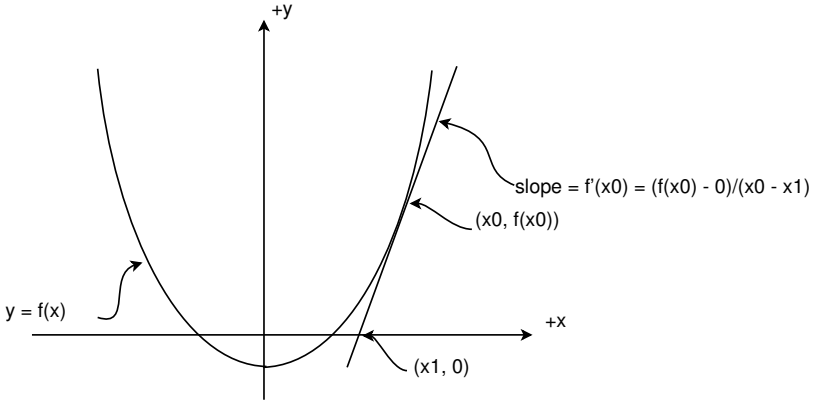


Figure 13.1: Newton's method.

```

public double squareRoot( double x, double approx ) {
    if( Math.abs( x - approx * approx ) < EPSILON ) {
        return approx;
    } // if
    else {
        double average = (approx + x/approx)/2;
        return squareRoot( x, average );
    } // else
} // squareRoot( double, double, double )

```

13. Here is another derivation of Newton's method. Figure 13.1 shows a plot of a function $f(x)$, a line that is tangent to that curve, the point of tangency, and the point at which the tangent intersects the x axis.

Let us suppose that $f(x) = x^2 - 2$. Then the derivative of $f(x)$ is $f'(x) = 2x$. We will begin by guessing that a root of $f(x)$ is at $x = x_0$. We will produce a better guess and call it x_1 .

Express the x_1 in terms of x_0 , $f(x_0)$, and $f'(x_0)$.

Lesson 14

“I did not distinguish between ‘art’ and ‘science’ and still don’t.”

—Alan Kay [19, page 39]

“The geniuses of the computer field, on the other hand, are the people with the keenest aesthetic senses, the ones who are capable of creating beauty.”

—David Gelernter [27, page 50]

1. Design, write, and test a class that models a vector in two dimensions. Include within your class methods for producing...
 - the sum of 2 vectors: $\vec{u} + \vec{v}$
 - the difference of 2 vectors: $\vec{u} - \vec{v}$
 - the product of a number and a vector: $s\vec{u}$
 - the dot product of two vectors: $\vec{u} \cdot \vec{v}$
 - the magnitude of a vector: $|\vec{u}|$

Also include a method a `toString()` method for producing a string that will give us a printable representation of a vector.

Here’s the mathematics that you need to know.

$$\begin{aligned}\vec{u} &= [u_x, u_y] \\ \vec{v} &= [v_x, v_y] \\ \vec{u} + \vec{v} &= [u_x + v_x, u_y + v_y] \\ \vec{u} - \vec{v} &= [u_x - v_x, u_y - v_y] \\ s\vec{u} &= [su_x, su_y] \\ \vec{u} \cdot \vec{v} &= u_x v_x + u_y v_y \\ |\vec{u}| &= \sqrt{\vec{u} \cdot \vec{v}}\end{aligned}$$

Define two constructors.

- A constructor with no parameters will set both elements of a newly created vector equal to zero.
 - A constructor with two parameters will set the elements of a newly created vector equal to the supplied values.
2. Design, write, and test a class that models a 2x2 matrix. Include within your class methods for the producing...
- the sum of 2 matrices: $A + B$
 - the difference of 2 matrices: $A - B$
 - the product of a number and a matrix: sA
 - the product of 2 matrices: AB
 - the determinant of a matrix: $|A|$
 - the inverse of a matrix (defined only when the determinant is not zero): A^{-1}
 - the product of a matrix and a vector: $A\vec{u}$

Also include a method a `toString()` method for producing a string that will give us a printable representation of a matrix.

Here's the mathematics that you need to know.

$$\begin{aligned}
 A &= \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \\
 B &= \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} \\
 A + B &= \begin{bmatrix} (a_{00} + b_{00}) & (a_{01} + b_{01}) \\ (a_{10} + b_{10}) & (a_{11} + b_{11}) \end{bmatrix} \\
 A - B &= \begin{bmatrix} (a_{00} - b_{00}) & (a_{01} - b_{01}) \\ (a_{10} - b_{10}) & (a_{11} - b_{11}) \end{bmatrix} \\
 sA &= \begin{bmatrix} sa_{00} & sa_{01} \\ sa_{10} & sa_{11} \end{bmatrix} \\
 A \times B &= \begin{bmatrix} (a_{00} b_{00} + a_{01} b_{10}) & (a_{00} b_{01} + a_{01} b_{11}) \\ (a_{10} b_{00} + a_{11} b_{10}) & (a_{10} b_{01} + a_{11} b_{11}) \end{bmatrix} \\
 |A| &= a_{00} a_{11} - a_{10} a_{01} \\
 A^{-1} &= \frac{1}{|A|} \begin{bmatrix} a_{11} & -a_{01} \\ -a_{10} & a_{00} \end{bmatrix} \\
 A\vec{u} &= \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \\
 &= \begin{bmatrix} a_{00}u_x + a_{01}u_y \\ a_{10}u_x + a_{11}u_y \end{bmatrix}
 \end{aligned}$$

Define two constructors.

- A constructor with no parameters will set all four elements of a newly created matrix equal to zero.
- A constructor with four parameters will set the elements of a newly created matrix equal to the supplied values.

Finally, it will be convenient to have methods that set the values

Vector2D
-x: double -y: double
+Vector2D() +Vector2D(x:double,y:double) +add(otherVector:Vector2D): Vector2D +subtract(otherVector:Vector2D): Vector2D +scale(factor:double): Vector2D +dot(otherVector:Vector2D): double +magnitude(): double +toString(): String

Figure 14.1: Vector2D class.

of the four elements of a matrix in special ways.

$$\begin{aligned}
 \textit{makeIdentity}() &\rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
 \textit{makeRotation}(\theta) &\rightarrow \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \\
 \textit{makeScaling}(\textit{scaleFactor}) &\rightarrow \begin{bmatrix} \textit{scaleFactor} & & 0 \\ & 0 & \textit{scaleFactor} \end{bmatrix}
 \end{aligned}$$

Matrix2x2
-m: double []
+Matrix2x2() +Matrix2x2(m00:double,m01:double,m10:double,m11:double) +makeIdentity(): void +makeRotation(angle:double): void +makeScaling(factor:double): void +add(otherMatrix:Matrix2x2): Matrix2x2 +subtract(otherMatrix:Matrix2x2): Matrix2x2 +multiply(otherMatrix:Matrix2x2): Matrix2x2 +multiply(factor:double): Matrix2x2 +multiply(u:Vector2D): Vector2D +determinant(): double +inverse(): Matrix2x2 +toString(): String

Figure 14.2: Matrix2x2 class.

Sources

- [1] Stephen Ambrose. *Undaunted Courage: Meriwether Lewis, Thomas Jefferson, and the Opening of the American West*. Simon & Schuster, New York, 1996.
- [2] Frederick P. Brooks, Jr. The mythical man-month. In Peter Freeman and Anthony I. Wasserman, editors, *Tutorial on Software Design Techniques*. IEEE Computer Society Press, Silver Spring, MD, 1983. This article originally appeared in the December 1974 issue of *Datamation* and in Brooks' 1975 book *The Mythical Man-Month*, published by Addison-Wesley.
- [3] Frederick P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987.
- [4] Frederick P. Brooks, Jr. The computer scientist as toolsmith ii. *Communications of the ACM*, 39(3):61–68, March 96.
- [5] John Seely Brown. Foreward—understanding silicon valley. In Martin Kenney, editor, *Understanding Silicon Valley: The Anatomy of an Entrepreneurial Region*. Stanford University Press, 2000.
- [6] John R. Dixon. *Design Engineering: Inventiveness, Analysis, and Decision*. McGraw-Hill, New York, 1966.
- [7] Richard P. Feynman. Introduction to computers. In Anthony J.G. Hey and Robin W. Allen, editors, *Feynman Lectures on Computation*. Perseus Books, Reading, MA, 1996.

- [8] David Gelernter. *Machine Beauty: Elegance and the Heart of Technology*. Basic Books/Perseus Books Group, New York, 1998.
- [9] George Gilder. *Microcosm: The Quantum Revolution in Economics and Technology*. Touchstone/Simon and Schuster Trade Paperbacks, 1990.
- [10] Andrew S. Grove. Intel international science and engineering fair keynote address, May 9 2001.
- [11] Richard W. Hamming. *Numerical Methods for Scientists and Engineers*. McGraw-Hill, New York, second edition, 1973.
- [12] Watts S. Humphrey. *A Discipline for Software Engineering*. SEI Series in Software Engineering. Addison-Wesley, Reading, MA, 1995.
- [13] Tracy Kidder. *The Soul of a New Machine*. Modern Library/Random House, New York, 1981.
- [14] Donald E. Knuth. Computer programming as an art. *Communications of the ACM*, 17(12):667–673, December 1974. This is Donald Knuth’s Turing Award Lecture.
- [15] Gene Kranz. *Failure is Not An Option: Mission Control from Mercury to Apollo 13 and Beyond*. Simon and Schuster, New York, 2000.
- [16] Glyn Moody. *Rebel Code: Inside Linux and the Open Source Revolution*. Perseus Books, Cambridge, MA, 2001.
- [17] Bruce Murray. *Journey Into Space: The First Three Decades of Space Exploration*. W.W. Norton & Company, New York, 1989.
- [18] James Oberg. Mag faget: Master builder. *OMNI Magazine*, April 1995.
- [19] Dennis Sasha and Cathy Lazere. *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Copernicus/Springer-Verlag, New York, 1995.

- [20] M. Mitchell Waldrop. *The Dream Machine: J.C.R. Licklider and the Revolution that Made Computing Personal*. Viking/Penguin Putnam, New York, 2001.

Recommended Reading

- [21] Isaac Asimov. *I, Robot*. Random House/Spectra, New York, 2008. (first published in 1950).
- [22] Jon Bentley. *Programming Pearls*. Addison-Wesley, Reading, MA, 1999. (first published in 1986).
- [23] Nathaniel S. Borenstein. *Programming as if People Mattered: Friendly Programs, Software Engineering, and Other Noble Delusions*. Princeton University Press, Princeton, NJ, 1991.
- [24] Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Boston, MA, 1995. (first published in 1975).
- [25] Martin Davis. *The Universal Computer: The Road from Leibniz to Turing*. W.W. Norton & Company, New York, 2000.
- [26] A.K. Dewdney. *The Turing Omnibus: 61 Excursions in Computer Science*. W.H. Freeman and Company/Computer Science Press, Rockville, MD, 1989.
- [27] David Gelernter. *Machine Beauty: Elegance and the Heart of Technology*. Perseus Books/Basic Books, New York, 1998.
- [28] Jon Gertner. *The Idea Factory: Bells Labs and the Great Age of American Innovation*. Penguin Press, New York, 2012.

- [29] George Gilder. *Microcosm: The Quantum Revolution in Economics and Technology*. Simon & Schuster/Touchstone, New York, 1990.
- [30] Fred Hapgood. *Up the Infinite Corridor: MIT and the Technical Imagination*. Perseus Books, New York, 1993.
- [31] Mark Kac, Gian-Carlo Rota, and Jacob T. Schwartz. *Discrete Thoughts: Essays on Mathematics, Science, and Philosophy*. Birkhäuser, Boston, MA, 1992.
- [32] Tracy Kidder. *The Soul of a New Machine*. Back Bay Books, 2000. (first published in 1981).
- [33] Steven Levy. *Insanely Great: The Life and Times of Macintosh, the Computer that Changed Everything*. Penguin Books, New York, 2000. (first published in 1994).
- [34] Steven Levy. *Hackers: Heroes of the Computer Revolution—25th Anniversary Edition*. O'Reilly Media, Sebastopol, CA, 2010.
- [35] Henry Petroski. *To Engineer is Human: The Role of Failure in Successful Design*. Random House/Vintage, New York, 1992.
- [36] Clifford A. Pickover. *Computers, Pattern, Chaos and Beauty: Graphics from an Unseen World*. St. Martin's Press, New York, 1990.
- [37] Clifford A. Pickover. *Computers and the Imagination: Visual Adventures Beyond the Edge*. Alan Sutton Publishing, Stroud, Gloucestershire, United Kingdom, 1991.
- [38] William Poundstone. *Prisoner's Dilemma: John von Neumann, Game Theory, and the Puzzle of the Bomb*. Random House/Anchor Books, New York, 1992. (This book includes a biographical sketch of John von Neumann.).
- [39] William Poundstone. *How Would You Move Mount Fuji?: Microsoft's Cult of the Puzzle—How the World's Smartest Companies Select the Most Creative Thinkers*. Little, Brown, and Company, New York, 2003.

- [40] William Poundstone. *Fortune's Formula: The Untold Story of the Scientific Betting System that Beat the Casinos and Wall Street*. Farrar, Straus and Giroux/Hill and Wang, New York, 2005. (This book includes a biographical sketch of Claude Shannon.).
- [41] William Poundstone. *Are You Smart Enough to Work at Google?* Little, Brown, and Company, New York, 2012.
- [42] Dennis Shasha and Cathy Lazere. *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Springer-Verlag/Copernicus, New York, 1995.
- [43] Cliff Stoll. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. Simon & Schuster/Pocket Books, New York, 2000. (first published 1990).
- [44] Norbert Wiener. *The Human Use of Human Beings: Cybernetics and Society*. Perseus Books/Da Capo Press, 1988. (first published in 1950).