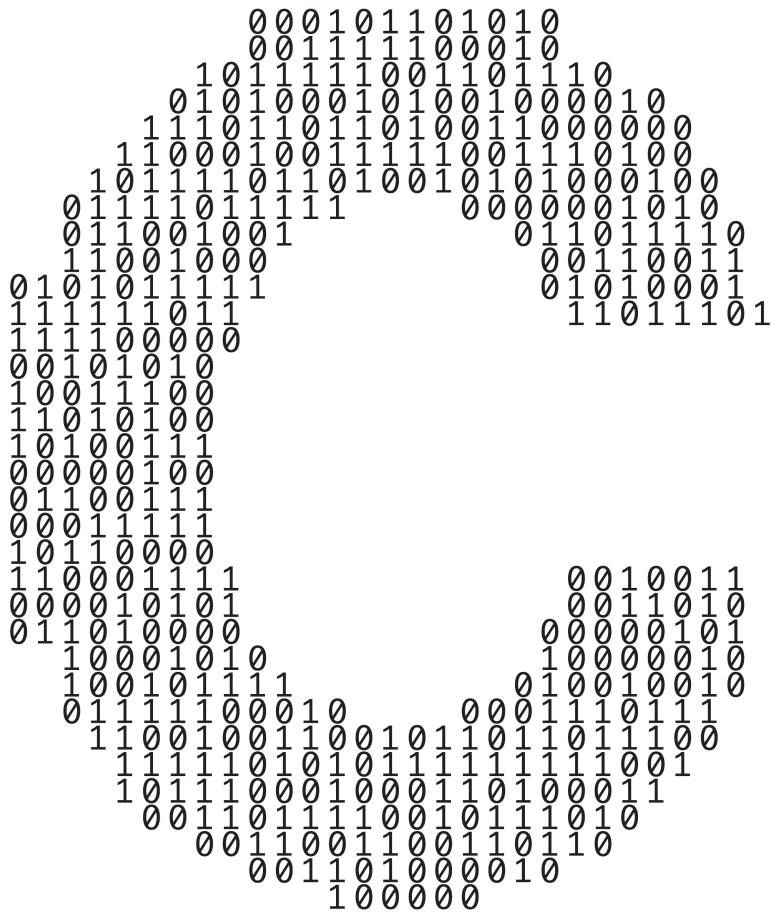
1

Introduction to Computers and C



Objectives

In this chapter, you'll:

- Learn about exciting recent developments in computing.
- Learn computer hardware, software and Internet basics.
- Understand the data hierarchy from bits to databases.
- Understand the different types of programming languages.
- Understand the strengths of C and other leading programming languages.
- Be introduced to the C standard library of reusable functions that help you avoid "reinventing the wheel."
- Test-drive a C program that you compile with one or more of the popular C compilers we used to develop the book's hundreds of C code examples, exercises and projects (EEPs).
- Be introduced to big data and data science.
- Be introduced to artificial intelligence—a key intersection of computer science and data science.

Outline

1.1 Introduction

Welcome to C—one of the world's most senior computer programming languages and, according to the Tiobe Index, the world's most popular.[1] You're probably familiar with many of the powerful tasks computers perform. In this textbook, you'll get intensive, hands-on experience writing C instructions that command computers to perform those and other tasks. **Software** (that is, the C instructions you write, which are also called **code**) controls **hardware** (that is, computers and related devices).

1. "TIOBE Index." Accessed November 4, 2020. `https://www.tiobe.com/tiobe-index/`.

C is widely used in industry for a wide range of tasks.[2] Today's popular desktop operating systems—Windows[3], macOS[4] and Linux[5]—are partially written in C. Many popular applications are partially written in C, including popular web browsers (e.g., Google Chrome[6] and Mozilla Firefox[7]), database management systems (e.g., Microsoft SQL Server[8], Oracle[9] and MySQL[10]) and more.

In this chapter, we introduce terminology and concepts that lay the groundwork for the C programming you'll learn, beginning in Chapter 2. We'll introduce hardware and software concepts. We'll also overview the data hierarchy—from individual bits (ones and zeros) to databases, which store the massive amounts of data that organizations need to implement contemporary applications such as Google Search, Netflix, Twitter, Waze, Uber, Airbnb and a myriad of others.

We'll discuss the types of programming languages. We'll introduce the C standard library and various C-based "open-source" libraries that help you avoid "reinventing the wheel." You'll use these libraries to perform powerful tasks with modest numbers of instructions. We'll introduce additional software technologies that you're likely to use as you develop software in your career.

Many development environments are available in which you can compile, build and run C applications. You'll work through one or more of the four test-drives showing how to compile and execute C code using:

- Microsoft Visual Studio 2019 Community edition for Windows.
- Clang in Xcode on macOS.
- GNU `gcc` in a shell on Linux.
- GNU `gcc` in a shell running inside the GNU Compiler Collection (GCC) Docker container.

You can read only the test-drive(s) required for your course or projects in industry.

In the past, most computer applications ran on "standalone" computers (that is, not networked together). Today's applications can communicate among the world's computers via the Internet. We'll introduce the Internet, the World Wide Web, the Cloud and the Internet of Things (IoT), each of which could play a significant part in the applications you'll build in the 2020s (and probably long afterward).

2. "After All These Years, the World is Still Powered by C Programming." Accessed Nov. 4, 2020. `https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming`.
3. "What Programming Language is Windows written in?" Accessed Nov. 4, 2020. `https://social.microsoft.com/Forums/en-US/65a1fe05-9c1d-48bf-bd40-148e6b3da9f1/what-programming-language-is-windows-written-in`.
4. "macOS." Accessed Nov. 4, 2020. `https://en.wikipedia.org/wiki/MacOS`.
5. "Linux kernel." Accessed Nov. 4, 2020. `https://en.wikipedia.org/wiki/Linux_kernel`.
6. "Google Chrome." Accessed Nov. 4, 2020. `https://en.wikipedia.org/wiki/Google_Chrome`.
7. "Firefox." Accessed Nov. 4, 2020. `https://en.wikipedia.org/wiki/Firefox`.
8. "Microsoft SQL Server." Accessed Nov. 4, 2020. `https://en.wikipedia.org/wiki/Microsoft_SQL_Server`.
9. "Oracle Database." Accessed Nov. 4, 2020. `https://en.wikipedia.org/wiki/Oracle_Database`.
10. "MySQL." Accessed Nov. 4, 2020. `https://en.wikipedia.org/wiki/MySQL`.

1.2 Hardware and Software

Computers can perform calculations and make logical decisions phenomenally faster than human beings can. Today's personal computers and smartphones can perform billions of calculations in one second—more than a human can perform in a lifetime. **Supercomputers** already perform *thousands of trillions (quadrillions)* of instructions per second! As of December 2020, Fujitsu's Fugaku[11] is the world's fastest supercomputer—it can perform 442 quadrillion calculations per second (442 *petaflops*)![12] To put that in perspective, this supercomputer *can perform in one second almost 58 million calculations for every person on the planet!*[13] And supercomputing upper limits are growing quickly.

Computers process data under the control of sequences of instructions called **computer programs** (or simply **programs**). These programs guide the computer through ordered actions specified by people called computer **programmers**.

A computer consists of various physical devices referred to as hardware—such as the keyboard, screen, mouse, solid-state disks, hard disks, memory, DVD drives and processing units. Computing costs are dropping dramatically due to rapid developments in hardware and software technologies. Computers that might have filled large rooms and cost millions of dollars decades ago are now inscribed on silicon computer chips smaller than a fingernail, costing perhaps a few dollars each. Ironically, silicon is one of the most abundant materials on Earth—it's an ingredient in common sand. Silicon-chip technology has made computing so economical that computers and computerized devices have become commodities.

1.2.1 Moore's Law

Every year, you probably expect to pay at least a little more for most products and services. The opposite has been the case in the computer and communications fields, especially with regard to the hardware supporting these technologies. Over the years, hardware costs have fallen rapidly.

For decades, every couple of years, computer processing power approximately doubled inexpensively. This remarkable trend often is called **Moore's Law**, named for Gordon Moore, co-founder of Intel and the person who identified this trend in the 1960s. Intel is a leading manufacturer of the processors in today's computers and **embedded systems**, such as smart home appliances, home security systems, robots, intelligent traffic intersections and more.

---

11. "Top 500." Accessed December 24, 2020. `https://en.wikipedia.org/wiki/TOP500#TOP_500`.
12. "Flops." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/FLOPS`.
13. For perspective on how far computing performance has come, consider this: In his early computing days in the 1960s, Harvey Deitel used the Digital Equipment Corporation PDP-1 (`https://en.wikipedia.org/wiki/PDP-1`), which was capable of performing only 93,458 operations per second, and the IBM 1401 (`http://www.ibm-1401.info/1401GuidePosterV9.html`), which performed only 86,957 operations per second.

Key executives at computer-processor companies NVIDIA and Arm have indicated that Moore's Law no longer applies.[14,15] Computer processing power continues to increase but relies on new processor designs, such as multicore processors (Section 1.2.2).

Moore's Law *and related observations* apply especially to

- the amount of memory that computers have for programs,
- the amount of secondary storage (such as hard disks and solid-state drive storage) they have to hold programs and data, and
- their processor speeds—that is, the speeds at which computers **execute** programs to do their work.

Similar growth has occurred in the communications field. Costs have plummeted as enormous demand for communications **bandwidth** (that is, information-carrying capacity) has attracted intense competition. We know of no other fields in which technology improves so quickly, and costs fall so rapidly. Such phenomenal improvement is truly fostering the **Information Revolution**.

1.2.2 Computer Organization

Regardless of physical differences, computers can be envisioned as divided into various **logical units** or sections.

Input Unit

This "receiving" section obtains information (data and computer programs) from **input devices** and places it at the other units' disposal for processing. Computers receive most user input through keyboards, touch screens, mice and touchpads. Other forms of input include:

- receiving voice commands,
- scanning images and barcodes,
- reading data from secondary storage devices (such as solid-state drives, hard drives, Blu-ray Disc™ drives and USB flash drives—also called "thumb drives" or "memory sticks"),
- receiving video from a webcam,
- receiving information from the Internet (such as when you stream videos from YouTube® or download e-books from Amazon),
- receiving position data from a GPS device,
- receiving motion and orientation information from an **accelerometer** (a device that responds to up/down, left/right and forward/backward accelera-

14. "Moore's Law turns 55: Is it still relevant?" Accessed November 2, 2020. `https://www.techrepublic.com/article/moores-law-turns-55-is-it-still-relevant`.
15. "Moore's Law is dead: Three predictions about the computers of tomorrow." Accessed November 2, 2020. `https://www.techrepublic.com/article/moores-law-is-dead-three-predictions-about-the-computers-of-tomorrow/`.

tion) in a smartphone or wireless game controllers, such as those for Microsoft® Xbox®, Nintendo Switch™ and Sony® PlayStation®, and

- receiving voice input from intelligent assistants like Apple Siri®, Amazon Alexa® and Google Home®.

Output Unit

This "shipping" section takes information the computer has processed and places it on various **output devices** to make it available outside the computer. Most information that's output from computers today is

- displayed on screens,
- printed on paper ("going green" discourages this),
- played as audio or video on smartphones, tablets, PCs and giant screens in sports stadiums,
- transmitted over the Internet, or
- used to control other devices, such as self-driving cars (and **autonomous vehicles** in general), robots and "intelligent" appliances.

Information is also commonly output to secondary storage devices, such as solid-state drives (SSDs), hard drives, USB flash drives and DVD drives. Popular recent forms of output are smartphone and game-controller vibration, virtual reality devices like Oculus Rift®, Oculus Quest®, Sony® PlayStation® VR and Samsung Gear VR®, and mixed reality devices like Magic Leap® One and Microsoft HoloLens™.

Memory Unit

This rapid-access, relatively low-capacity "warehouse" section retains information entered through the input unit, making it immediately available for processing when needed. The memory unit also retains processed information until it can be placed on output devices by the output unit. Information in the memory unit is **volatile**—it's typically lost when the computer's power is turned off. The memory unit is often called either **memory**, **primary memory** or **RAM** (Random Access Memory). Main memories on desktop and notebook computers contain as much as 128 GB of RAM, though 8 to 16 GB is most common. GB stands for gigabytes; a **gigabyte** is approximately one billion bytes. A **byte** is eight bits. A **bit** (short for "*bi*nary digi*t*") is either a `0` or a `1`.

Arithmetic and Logic Unit (ALU)

This "manufacturing" section performs calculations (e.g., addition, subtraction, multiplication and division) and makes decisions (e.g., comparing two items from the memory unit to determine whether they're equal). In today's systems, the ALU is part of the next logical unit, the CPU.

Central Processing Unit (CPU)

This "administrative" section coordinates and supervises the operation of the other sections. The CPU tells

- the input unit when to read information into the memory unit,
- the ALU when to use information from the memory unit in calculations, and
- the output unit when to send information from the memory unit to specific output devices.

Most computers today have **multicore processors** that economically implement multiple processors on a single integrated circuit chip. Such processors can perform many operations simultaneously. A **dual-core processor** has two CPUs, a **quad-core processor** has four and an **octa-core processor** has eight. Intel has some processors with up to 72 cores.

Secondary Storage Unit

This is the long-term, high-capacity "warehousing" section. Programs and data not actively being used by the other units are placed on secondary storage devices until they're again needed, possibly hours, days, months or even years later. Information on secondary storage devices is **persistent**—it's preserved even when the computer's power is turned off. Secondary storage information takes much longer to access than information in primary memory, but its cost per byte is much less. Examples of secondary storage devices include solid-state drives (SSDs), USB flash drives, hard drives and read/write Blu-ray drives. Many current drives hold terabytes (TB) of data. A **terabyte** is approximately one trillion bytes. Typical desktop and notebook-computer hard drives hold up to 4 TB, and some recent desktop-computer hard drives hold up to 20 TB.[16] The largest commercial SSD holds up to 100 TB (and costs $40,000).[17]

Self Check

1 ***(Fill-In)*** For many decades, every year or two, computers' capacities have approximately doubled inexpensively. This remarkable trend often is called _____.
Answer: Moore's Law.

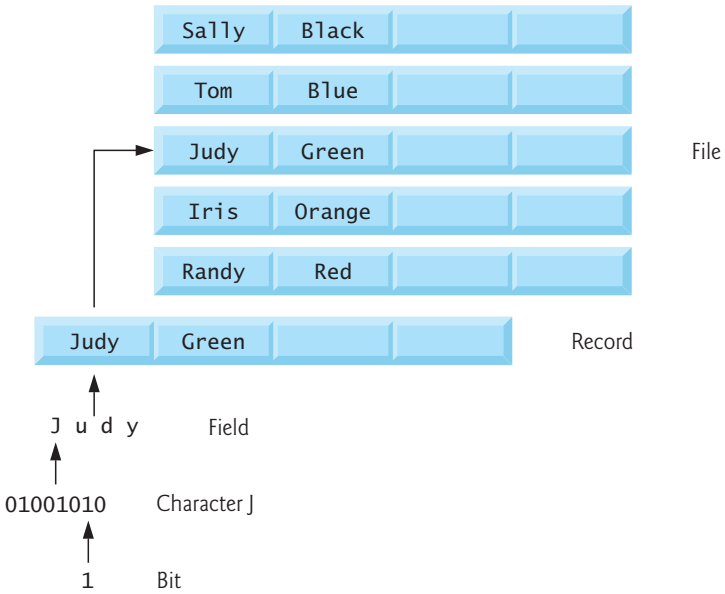
2 ***(True/False)*** Information in the memory unit is persistent—it's preserved even when the computer's power is turned off
Answer: False. Information in the memory unit is volatile—it's typically lost when the computer's power is turned off.

3 ***(Fill-In)*** Most computers today have _____ processors that implement multiple processors on a single integrated-circuit chip. Such processors can perform many operations simultaneously.
Answer: multicore.

16. "History of hard disk drives." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/History_of_hard_disk_drives`.
17. "At 100TB, the world's biggest SSD gets an (eye-watering) price tag." Accessed November 1, 2020. `https://www.techradar.com/news/at-100tb-the-worlds-biggest-ssd-gets-an-eye-watering-price-tag`.

1.3 Data Hierarchy

Data items processed by computers form a **data hierarchy** that becomes larger and more complex in structure as we progress from the simplest data items (called "bits") to richer ones, such as characters and fields. The following diagram illustrates a portion of the data hierarchy:



Bits

A bit is short for "*b*inary dig*it*"—a digit that can assume one of *two* values—and is a computer's smallest data item. It can have the value `0` or `1`. Remarkably, computers' impressive functions involve only the simplest manipulations of `0`s and `1`s—examining a bit's value, setting a bit's value and reversing a bit's value (from `1` to `0` or from `0` to `1`). Bits form the basis of the binary number system, which we discuss in our "Number Systems" appendix.

Characters

Work with data in the low-level form of bits is tedious. Instead, people prefer to work with **decimal digits** (`0`–`9`), **letters** (`A`–`Z` and `a`–`z`) and **special symbols** such as

```
$  @  %  &  *  (  )  -  +  "  :  ;  ,  ?  /
```

Digits, letters and special symbols are known as **characters**. The computer's **character set** contains the characters used to write programs and represent data items. Computers process only `1`s and `0`s, so a computer's character set represents each character as a pattern of `1`s and `0`s. C uses the **ASCII (American Standard Code for Information Interchange)** character set by default. C also supports **Unicode®** characters composed of one, two, three or four bytes (8, 16, 24 or 32 bits, respectively).[18]

18. "Programming with Unicode." Accessed November 1, 2020. `https://unicodebook.readthedocs.io/programming_languages.html`.

Unicode contains characters for many of the world's languages. ASCII is a (tiny) subset of Unicode representing letters (a–z and A–Z), digits and some common special characters. You can view the ASCII subset of Unicode at

```
https://www.unicode.org/charts/PDF/U0000.pdf
```

For the lengthy Unicode charts for all languages, symbols, emojis and more, visit

```
http://www.unicode.org/charts/
```

Fields

Just as characters are composed of bits, **fields** are composed of characters or bytes. A field is a group of characters or bytes that conveys meaning. For example, a field consisting of uppercase and lowercase letters could represent a person's name, and a field consisting of decimal digits could represent a person's age in years.

Records

Several related fields can be used to compose a **record**. In a payroll system, for example, the record for an employee might consist of the following fields (possible types for these fields are shown in parentheses):

- Employee identification number (a whole number).
- Name (a group of characters).
- Address (a group of characters).
- Hourly pay rate (a number with a decimal point).
- Year-to-date earnings (a number with a decimal point).
- Amount of taxes withheld (a number with a decimal point).

Thus, a record is a group of related fields. All the fields listed above belong to the same employee. A company might have many employees and a payroll record for each.

Files

A **file** is a group of related records. More generally, a file contains arbitrary data in arbitrary formats. Some operating systems view a file simply as a *sequence of bytes*—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer. You'll see how to do that in Chapter 11, File Processing. It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information. As we'll see below, with big data, far larger file sizes are becoming increasingly common.

Databases

A **database** is a collection of data organized for easy access and manipulation. The most popular model is the **relational database**, in which data is stored in simple tables. A table includes records and fields. For example, a table of students might include first name, last name, major, year, student ID number and grade-point-average fields. The data for each student is a record, and the individual pieces of informa-

tion in each record are the fields. You can search, sort and otherwise manipulate the data based on its relationship to multiple tables or databases. For example, a university might use data from the student database combined with data from databases of courses, on-campus housing, meal plans, etc.

Big Data

The table below shows some common byte measures:

Unit	Bytes	Which is approximately
1 kilobyte (KB)	1024 bytes	10^3 bytes (1024 bytes exactly)
1 megabyte (MB)	1024 kilobytes	10^6 (1,000,000) bytes
1 gigabyte (GB)	1024 megabytes	10^9 (1,000,000,000) bytes
1 terabyte (TB)	1024 gigabytes	10^{12} (1,000,000,000,000) bytes
1 petabyte (PB)	1024 terabytes	10^{15} (1,000,000,000,000,000) bytes
1 exabyte (EB)	1024 petabytes	10^{18} (1,000,000,000,000,000,000) bytes
1 zettabyte (ZB)	1024 exabytes	10^{21} (1,000,000,000,000,000,000,000) bytes

The amount of data being produced worldwide is enormous, and its growth is accelerating. **Big data** applications deal with massive amounts of data. This field is growing quickly, creating lots of opportunities for software developers. Millions of information technology (IT) jobs globally already support big-data applications.

Twitter®—A Favorite Big-Data Source

One big-data source favored by developers is Twitter. There are approximately 800,000,000 tweets per day.[19] Though tweets appear to be limited to 280 characters, Twitter actually provides almost 10,000 bytes of data per tweet to programmers who want to analyze tweets. So 800,000,000 times 10,000 is about 8,000,000,000,000 bytes or 8 terabytes (TB) of data per day. That's big data.

Prediction is a challenging and often costly process, but the potential rewards for accurate predictions are great. **Data mining** is the process of searching through extensive collections of data, often big data, to find insights that can be valuable to individuals and organizations. The sentiment that you data-mine from tweets could help predict the election results, the revenues a new movie is likely to generate and the success of a company's marketing campaign. It could also help companies spot weaknesses in competitors' product offerings.

Self Check

1 ***(Fill-In)*** A(n) _____ is short for "binary digit"—a digit that can assume one of two values and is a computer's smallest data item.

Answer: bit.

19. "Twitter Usage Statistics." Accessed November 1, 2020. `https://www.internetlivestats.com/twitter-statistics/`.

2 *(True/False)* In some operating systems, a file is viewed simply as a sequence of bytes—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer.
Answer: True.

3 *(Fill-In)* A database is a collection of data organized for easy access and manipulation. The most popular model is the _____ database, in which data is stored in simple tables.
Answer: relational.

1.4 Machine Languages, Assembly Languages and High-Level Languages

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps. Hundreds of such languages are in use today. These may be divided into three general types:

- Machine languages.
- Assembly languages.
- High-level languages.

Machine Languages

Any computer can directly understand only its own **machine language**, defined by its hardware design. Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time. Machine languages are machine-dependent—a particular machine language can be used on only one type of computer. Such languages are cumbersome for humans. For example, here's a section of an early machine-language payroll program that adds overtime pay to base pay and stores the result in gross pay:

```
+1300042774
+1400593419
+1200274027
```

In our Building Your Own Computer case study (Exercises 7.28–7.30), you'll "peel open" a computer and look at its internal structure. We'll introduce machine-language programming, and you'll write several machine-language programs. To make this an especially valuable experience, you'll then build a software simulation of a computer on which you can execute your machine-language programs.

Assembly Languages and Assemblers

Programming in machine language was simply too slow and tedious for most programmers. Instead of using the strings of numbers that computers could directly understand, programmers began using English-like abbreviations to represent elementary operations. These abbreviations formed the basis of **assembly languages**.

Translator programs called **assemblers** were developed to convert assembly-language programs to machine language at computer speeds. The following section of an assembly-language payroll program also adds overtime pay to base pay and stores the result in gross pay:

```
load    basepay
add     overpay
store   grosspay
```

Although such code is clearer to humans, it's incomprehensible to computers until it's translated to machine language.

High-Level Languages and Compilers

With the advent of assembly languages, the use of computers increased rapidly. However, programmers still had to use numerous instructions to accomplish even simple tasks. To speed the programming process, **high-level languages** were developed in which single statements could accomplish substantial tasks. A typical high-level-language program contains many statements, known as the program's **source code**.

Translator programs called **compilers** convert high-level-language source code into machine language. High-level languages allow you to write instructions that look almost like everyday English and contain common mathematical notations. A payroll program written in a high-level language might contain a single statement such as

```
grossPay = basePay + overTimePay
```

From the programmer's standpoint, high-level languages are preferable to machine and assembly languages. C is among the world's most widely used high-level programming languages.

In our Building Your Own Compiler case study (Exercises 12.24–12.27), you'll build a compiler that takes programs written in a high-level programming language and converts them to the Simpletron Machine Language that you learn in Exercise 7.28. Exercises 12.24–12.27 "tie" together the entire programming process. You'll write programs in a simple high-level language, compile the programs on the compiler you build, then run the programs on the Simpletron simulator you build in Exercise 7.29.

Interpreters

Compiling a large high-level language program into machine language can take considerable computer time. **Interpreters** execute high-level language programs directly. Interpreters avoid compilation delays, but your code runs slower than compiled programs. Some programming languages, such as Java[20] and Python[21], use a clever mixture of compilation and interpretation to run programs.

---

20. "Java virtual machine." Accessed November 2, 2020. https://en.wikipedia.org/wiki/Java_virtual_machine#Bytecode_interpreter_and_just-in-time_compiler.
21. "An introduction to Python bytecode." Accessed November 1, 2020. https://opensource.com/article/18/4/introduction-python-bytecode.

Self Check

1 *(Fill-In)* Translator programs called _____ convert assembly-language programs to machine language at computer speeds.
Answer: assemblers.

2 *(Fill-In)* _____ programs, developed to execute high-level-language programs directly, avoid compilation delays, although they run slower than compiled programs
Answer: Interpreter.

3 *(True/False)* High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical notations.
Answer: True.

1.5 Operating Systems

Operating systems are software that make using computers more convenient for users, software developers and system administrators. They provide services that allow applications to execute safely, efficiently and concurrently with one another. The software that contains the core operating-system components is called the kernel. Linux, Windows and macOS are popular desktop computer operating systems—you can use any of these with this book. Each is partially written in C. The most popular mobile operating systems used in smartphones and tablets are Google's Android and Apple's iOS.

Windows—A Proprietary Operating System

In the mid-1980s, Microsoft developed the Windows operating system, consisting of a graphical user interface built on top of DOS (Disk Operating System)—an enormously popular personal-computer operating system that users interacted with by typing commands. Windows 10 is Microsoft's latest operating system—it includes the Cortana personal assistant for voice interactions. Windows is a proprietary operating system—it's controlled by Microsoft exclusively. It is by far the world's most widely used desktop operating system.

Linux—An Open-Source Operating System

The Linux operating system is among the greatest successes of the open-source movement. Proprietary software for sale or lease dominated software's early years. With open source, individuals and companies contribute to developing, maintaining and evolving the software. Anyone can then use that software for their own purposes—normally at no charge, but subject to a variety of (typically generous) licensing requirements. Open-source code is often scrutinized by a much larger audience than proprietary software, so errors can get removed faster, making the software more robust. Open source increases productivity and has contributed to an explosion of innovation. You'll use various popular open-source libraries and tools throughout this book.

There are many organizations in the open-source community. Some key ones are:

- GitHub (provides tools for managing open-source projects—it has millions of them under development).

- The **Apache Software Foundation** (originally the creators of the Apache web server) now oversees 350+ open-source projects, including several big-data infrastructure technologies.
- The **Eclipse Foundation** (the Eclipse Integrated Development Environment helps programmers conveniently develop software).
- The **Mozilla Foundation** (creators of the Firefox web browser).
- **OpenML** (which focuses on open-source tools and data for machine learning.
- **OpenAI** (which does research on artificial intelligence and publishes open-source tools used in AI reinforcement-learning research).
- **OpenCV** (which focuses on open-source computer-vision tools that can be used across various operating systems and programming languages.
- **Python Software Foundation** (responsible for the Python programming language).

Rapid improvements to computing and communications, decreasing costs and open-source software have made it much easier and more economical to create software-based businesses now than just a decade ago. A great example is Facebook, which was launched from a college dorm room and built with open-source software.

The **Linux kernel** is the core of the most popular open-source, freely distributed, full-featured operating system. It's developed by a loosely organized team of volunteers and is popular in servers, personal computers and embedded systems (such as the computer systems at the heart of smartphones, smart TVs and automobile systems). Unlike Microsoft's Windows and Apple's macOS source code, the Linux source code is available to the public for examination and modification and is free to download and install. As a result, Linux users benefit from a huge community of developers actively debugging and improving the kernel, and from the ability to customize the operating system to meet specific needs.

Apple's macOS and Apple's iOS for iPhone® and iPad® Devices

Apple, founded in 1976 by Steve Jobs and Steve Wozniak, quickly became a leader in personal computing. In 1979, Jobs and several Apple employees visited Xerox PARC (Palo Alto Research Center) to learn about Xerox's desktop computer that featured a graphical user interface (GUI). That GUI served as the inspiration for the Apple Macintosh, launched in 1984.

The Objective-C programming language, created by Stepstone in the early 1980s, added object-oriented programming (OOP) capabilities to the C programming language. Steve Jobs left Apple in 1985 and founded NeXT Inc. In 1988, NeXT licensed Objective-C from Stepstone. NeXT developed an Objective-C compiler and libraries, which were used as the platform for the NeXTSTEP operating system's user interface and Interface Builder (for constructing graphical user interfaces).

Jobs returned to Apple in 1996 when they bought NeXT. Apple's **macOS operating system** is a descendant of NeXTSTEP. Apple has several other proprietary operating systems derived from macOS:

- **iOS** is used in iPhones.
- **iPadOS** is used in iPads.
- **watchOS** is used in Apple Watches.
- **tvOS** is used in Apple TV devices.

In 2014, Apple introduced its Swift programming language, which it open-sourced in 2015. The Apple app-development community has largely shifted from Objective-C to Swift. Swift-based apps can import Objective-C and C software components.[22]

Google's Android

Android—the most widely used mobile and smartphone operating system—is based on the Linux kernel, the Java programming language and, now, the open-source Kotlin programming language. Android is open source and free. Though you can't develop Android apps purely in C, you can incorporate C code into Android apps.[23]

According to `idc.com`, 84.8% of smartphones shipped in 2020 use Android, compared to 15.2% for Apple.[24] The Android operating system is used in numerous smartphones, e-reader devices, tablets, TVs, in-store touch-screen kiosks, cars, robots, multimedia players and more.

Billions of Computerized Devices

Billions of personal computers and an even larger number of mobile devices are now in use. The explosive growth of smartphones, tablets and other devices creates significant opportunities for mobile-app developers. The following table lists many computerized devices, each of which can be part of the Internet of Things (see Section 1.11).

Some computerized devices		
Automobiles	Blu-ray Disc™ players	Building controls
Cable boxes	Desktop computers	Credit cards
CT scanners	GPS navigation systems	e-Readers
Game consoles	Lottery systems	Home appliances
Home security systems	MRIs	Medical devices
Mobile phones	Parking meters	Personal computers
Optical sensors	Printers	Robots
Point-of-sale terminals	Servers	Smartcards
Smart meters	Televisions	Smartphones
Tablets	TV set-top boxes	Thermostats
Transportation passes	ATMs	Vehicle diagnostic systems

22. "Imported C and Objective-C APIs." Accessed November 3, 2020. `https://developer.apple.com/documentation/swift/imported_c_and_objective-c_apis`.
23. "Add C and C++ code to your project." Accessed November 3, 2020. `https://developer.android.com/studio/projects/add-native-code`.
24. "Smartphone Market Share." Accessed December 24, 2020. `https://www.idc.com/promo/smartphone-market-share/os`.

Self Check

1 *(Fill-In)* Windows is a(n) _____ operating system—it's controlled by Microsoft exclusively.
Answer: proprietary.

2 *(True/False)* Proprietary code is often scrutinized by a much larger audience than open-source software, so errors often get removed faster.
Answer: False. Open-source code is often scrutinized by a much larger audience than proprietary software, so errors often get removed faster.

3 *(True/False)* iOS dominates the global smartphone market over Android.
Answer: False. Android currently controls 84.8% of the smartphone market, but iOS apps earn almost twice as much revenue as Android apps.[25]

1.6 The C Programming Language

C evolved from two earlier languages, BCPL[26] and B[27]. BCPL was developed in 1967 by Martin Richards as a language for writing operating systems and compilers. Ken Thompson modeled many features in his B language after their counterparts in BCPL, and in 1970 he used B to create early versions of the UNIX operating system at Bell Laboratories.

The C language was evolved from B by Dennis Ritchie at Bell Laboratories and was originally implemented in 1972.[28] C initially became widely known as the development language of the UNIX operating system. Many of today's leading operating systems are written in C and/or C++. C is mostly hardware-independent—with careful design, it's possible to write C programs that are **portable** to most computers.

Built for Performance

C is widely used to develop systems that demand performance, such as operating systems, embedded systems, real-time systems and communications systems:

Application	Description
Operating systems	C's portability and performance make it desirable for implementing operating systems, such as Linux and portions of Microsoft's Windows and Google's Android. Apple's macOS is built in Objective-C, which was derived from C. We discussed some key popular desktop/notebook operating systems and mobile operating systems in Section 1.5.

25. "Global App Revenue Reached $50 Billion in the First Half of 2020, Up 23% Year-Over-Year." Accessed November 1, 2020. `https://sensortower.com/blog/app-revenue-and-downloads-1h-2020`.
26. "BCLP." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/BCPL`.
27. "B (programming language)." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/B_(programming_language)`.
28. "C (programming language)." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/C_(programming_language)`.

Application	Description
Embedded systems	The vast majority of the microprocessors produced each year are embedded in devices other than general-purpose computers. These **embedded systems** include navigation systems, smart home appliances, home security systems, smartphones, tablets, robots, intelligent traffic intersections and more. C is one of the most popular programming languages for developing embedded systems, which typically need to run as fast as possible and conserve memory. For example, a car's antilock brakes must respond immediately to slow or stop the car without skidding; video-game controllers should respond instantaneously to prevent lag between the controller and the game action.
Real-time systems	Real-time systems are often used for "mission-critical" applications that require nearly instantaneous and predictable response times. Real-time systems need to work continuously. For example, an air-traffic-control system must continuously monitor planes' positions and velocities and report that information to air-traffic controllers without delay so they can alert the planes to change course if there's a possibility of a collision.
Communications systems	Communications systems need to route massive amounts of data to their destinations quickly to ensure that things such as audio and video are delivered smoothly and without delay.

By the late 1970s, C had evolved into what's now referred to as "traditional C." The publication in 1978 of Kernighan and Ritchie's book, *The C Programming Language,* drew wide attention to the language. This became one of the most successful computer-science books of all time.

Standardization

C's rapid expansion to various **hardware platforms** (that is, types of computer hardware) led to many similar but often incompatible C versions. This was a serious problem for programmers who needed to develop code for several platforms. It became clear that a standard C version was needed. In 1983, the American National Standards Committee on Computers and Information Processing (X3) created the X3J11 technical committee to "provide an unambiguous and machine-independent definition of the language." In 1989, the standard was approved in the United States through the **American National Standards Institute (ANSI)**, then worldwide through the **International Standards Organization (ISO)**. This version was simply called Standard C.

The C11 and C18 Standards

We discuss the latest C standard (referred to as C11), which was approved in 2011 and updated with bug fixes in 2018 (referred to as C18). C11 refined and expanded C's capabilities. We've integrated into the text and Appendix C (in easy-to-include-or-omit sections) many of the new features implemented in leading C compilers. The current C standard document is referred to as *ISO/IEC 9899:2018*. Copies may be ordered from

```
https://www.iso.org/standard/74528.html
```

According to the C standard committee, the next C standard is likely to be released in 2022.[29]

Because C is a hardware-independent, widely available language, C applications often can run with little or no modification on a wide range of computer systems.

Self Check

1 ***(Fill-In)*** C evolved from two previous languages, _____ and _____.
Answer: BCPL, B.

2 ***(True/False)*** It's possible to write C programs that are portable to most computers.
Answer: *True*.

1.7 The C Standard Library and Open-Source Libraries

C programs consist of pieces called **functions**. You can program all the functions you need to form a C program. However, most C programmers take advantage of the rich collection of existing functions in the **C standard library**. Thus, there are really two parts to learning C programming:

- learning the C language itself, and
- learning how to use the functions in the C standard library.

Throughout the book, we discuss many of these functions. P. J. Plauger's book *The Standard C Library* is must reading for programmers who need a deep understanding of the library functions, how to implement them and how to use them to write portable code. We use and explain many C library functions throughout this text.

C How to Program, 9/e encourages a **building-block approach** to creating programs. When programming in C, you'll typically use the following building blocks:

- C standard library functions,
- open-source C library functions,
- functions you create yourself, and
- functions other people (whom you trust) have created and made available to you.

The advantage of creating your own functions is that you'll know exactly how they work. The disadvantage is the time-consuming effort that goes into designing, developing, debugging and performance-tuning new functions. Throughout the book, we focus on using the existing C standard library to leverage your program-development efforts and avoid "reinventing the wheel." This is called **software reuse**.

PERF Using C standard library functions instead of writing your own versions can improve program performance, because these functions are carefully written to perform efficiently. Using C standard library functions instead of writing your own comparable versions also can improve program portability.

29. "Programming Language C — C2x Charter." Accessed November 4, 2020. `http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2086.htm`.

Open-Source Libraries

There are enormous numbers of third-party and open-source C libraries that can help you perform significant tasks with modest amounts of code. GitHub lists over 32,000 repositories in their C category:

```
https://github.com/topics/c
```

In addition, pages such as *Awesome C*

```
https://github.com/kozross/awesome-c
```

provide curated lists of popular C libraries for a wide range of application areas.

Self Check

1 ***(Fill-In)*** Most C programmers take advantage of the rich collection of existing functions called the _____.

Answer: C standard library.

2 ***(Fill-In)*** Avoid "reinventing the wheel" Instead, use existing pieces. This is called _____.

Answer: software reuse.

1.8 Other Popular Programming Languages

The following is a brief intro to several other popular programming languages:

- *BASIC* was developed in the 1960s at Dartmouth College to familiarize novices with programming techniques. Many of its latest versions are object-oriented.
- *C++*, which is based on C, was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories. C++ provides features that enhance the C language and adds object-oriented programming capabilities. We introduce object-oriented programming concepts in Appendix D.
- *Python* is an object-oriented language that was released publicly in 1991. It was developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam. Python has rapidly become one of the world's most popular programming languages, especially for educational and scientific computing, and in 2017 it surpassed the programming language R as the most popular data-science programming language.[30,31,32] Some rea-

30. "5 things to watch in Python in 2017." Accessed November 1, 2020. https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017.
31. "Python overtakes R, becomes the leader in Data Science, Machine Learning platforms." Accessed November 1, 2020. https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html.
32. "Data Science Job Report 2017: R Passes SAS, But Python Leaves Them Both Behind." Accessed November 1, 2020. https://www.r-bloggers.com/data-science-job-report-2017-r-passes-sas-but-python-leaves-them-both-behind/.

sons why Python is popular:[33,34,35] It's open-source, free and widely available. It's supported by a massive open-source community. It's relatively easy to learn. Its code is easier to read than many other popular programming languages. It enhances developer productivity with extensive standard libraries and thousands of third-party open-source libraries. It's popular in web development and in artificial intelligence, which is enjoying explosive growth, in part because of its special relationship with data science. It's widely used in the financial community.[36]

- *Java*—Sun Microsystems in 1991 funded an internal corporate research project led by James Gosling, which resulted in the C++-based object-oriented programming language called Java. A key Java goal is "write once, run anywhere," enabling developers to write programs that run on a wide variety of computer systems. Java is used in enterprise applications, in web servers (the computers that provide the content to our web browsers), in applications for consumer devices (e.g., smartphones, tablets, television set-top boxes, appliances, automobiles and more) and for many other purposes. Java was originally the preferred Android app-development language, though several other languages are now supported.
- *C#* (based on C++ and Java) is one of Microsoft's three primary object-oriented programming languages—the other two are Visual C++ and Visual Basic. C# was developed to integrate the web into computer applications and is now widely used to develop many kinds of applications. As part of Microsoft's many open-source initiatives, they now offer open-source versions of C# and Visual Basic.
- *JavaScript* is a widely used scripting language that's primarily used to add programmability to web pages (e.g., animations, user interactivity and more). All major web browsers support it. Many Python visualization libraries output JavaScript to create interactive visualizations you can view in your web browser. Tools such as NodeJS also enable JavaScript to run outside of web browsers.
- *Swift*, which was introduced in 2014, is Apple's programming language for developing iOS and macOS apps. Swift is a contemporary language that includes popular features from Objective-C, Java, C#, Ruby, Python and other languages. Swift is open-source, so it can be used on non-Apple platforms as well.

33. "Why Learn Python? Here Are 8 Data-Driven Reasons." Accessed November 1, 2020. `https://dbader.org/blog/why-learn-python`.
34. "Why Learn Python." Accessed November 1, 2020. `https://simpleprogrammer.com/7-reasons-why-you-should-learn-python`.
35. "5 things to watch in Python in 2017." Accessed November 1, 2020. `https://www.oreilly.com/ideas/5-things-to-watch-in-python-in-2017`.
36. Kolanovic, M. and R. Krishnamachari, *Big Data and AI Strategies: Machine Learning and Alternative Data Approach to Investing* (J.P. Morgan, 2017).

- *R* is a popular open-source programming language for statistical applications and visualization. Python and R are the two most widely used data-science languages.

Self Check

1 ***(Fill-In)*** Today, most code for general-purpose operating systems and other performance-critical systems is written in _____.
Answer: C or C++.

2 ***(Fill-In)*** A key goal of _____ is "write once, run anywhere," enabling developers to write programs that will run on a great variety of computer systems and computer-controlled devices.
Answer: Java.

3 ***(True/False)*** R is the most popular data-science programming language.
Answer: False. In 2017, Python surpassed R as the most popular data-science programming language.

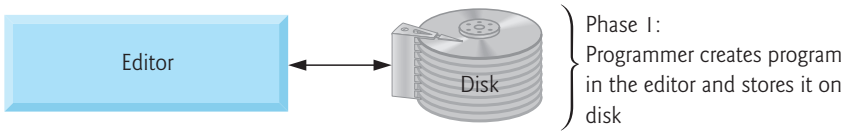
1.9 Typical C Program-Development Environment

C systems generally consist of several parts: a program-development environment, the language and the C standard library. The following discussion explains the typical C development environment.

C programs typically go through six phases to be executed—**edit**, **preprocess**, **compile**, **link**, **load** and **execute**. Although *C How to Program, 9/e*, is a generic C textbook (written independently of any particular operating system), we concentrate in this section on a typical Linux-based C system. In Section 1.10, you'll test-drive creating and running C programs on Windows, macOS and/or Linux.

1.9.1 Phase 1: Creating a Program

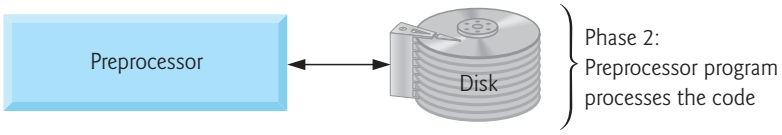
Phase 1 (in the following diagram) consists of editing a file in an **editor program**:



Two editors widely used on Linux systems are `vi` and `emacs`. C and C++ integrated development environments (IDEs) such as Microsoft Visual Studio and Apple Xcode have integrated editors. You type a C program in the editor, make corrections if necessary, then store the program on a secondary storage device such as a hard disk. C program filenames should end with the `.c` extension.

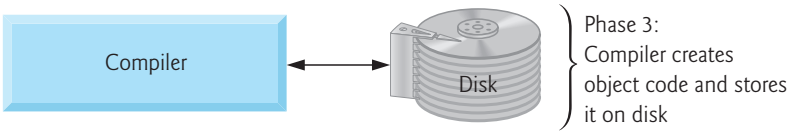
1.9.2 Phases 2 and 3: Preprocessing and Compiling a C Program

In Phase 2 (shown in the following diagram), you give the command to **compile** the program:



The compiler translates the C program into machine-language code (also referred to as **object code**). In a C system, the compilation command invokes a **preprocessor** program before the compiler's translation phase begins. The **C preprocessor** obeys special commands called **preprocessor directives**, which perform text manipulations on a program's source-code files. These manipulations consist of inserting the contents of other files and various text replacements. The early chapters discuss the most common preprocessor directives. Chapter 14 discusses other preprocessor features.

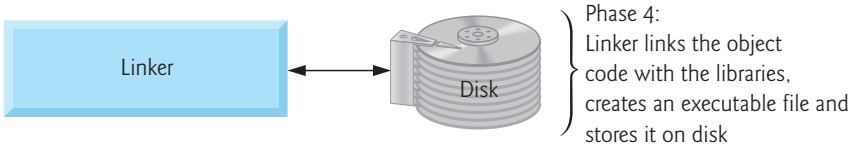
In Phase 3 (shown in the following diagram), the compiler translates the C program into machine-language code:



A **syntax error** occurs when the compiler cannot recognize a statement because it violates the language rules. The compiler issues an error message to help you locate and fix the incorrect statement. The C standard does not specify the wording for error messages issued by the compiler, so the messages you see on your system may differ from those on other systems. Syntax errors are also called **compile errors** or **compile-time errors.**

1.9.3 Phase 4: Linking

The next phase (shown in the following diagram) is called **linking**:



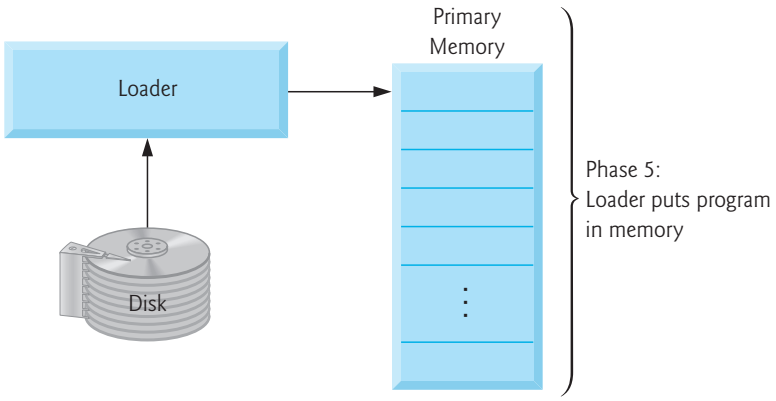
C programs typically use functions defined elsewhere, such as in the standard libraries, open-source libraries or private libraries of a particular project. The object code produced by the C compiler typically contains "holes" due to these missing parts. A **linker** links a program's object code with the code for the missing functions to produce an **executable image** (with no missing pieces). On a typical Linux system, the command to compile and link a program is **gcc** (the GNU C compiler). To compile and link a program named `welcome.c` using the latest C standard (C18), type

```
gcc -std=c18 welcome.c
```

at the Linux prompt and press the *Enter* key (or *Return* key). Linux commands are case sensitive. If the program compiles and links correctly, the compiler produces a file named `a.out` (by default), which is `welcome.c`'s executable image.

1.9.4 Phase 5: Loading

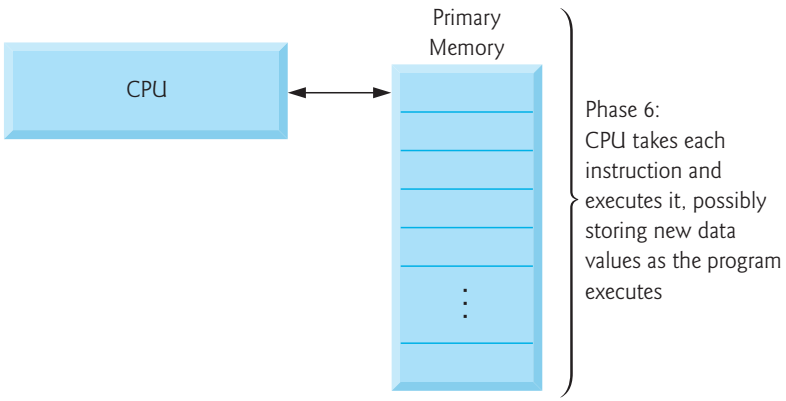
The next phase (shown in the following diagram) is called **loading**:



Before a program can execute, the operating system must load it into memory. The **loader** takes the executable image from disk and transfers it to memory. Additional components from shared libraries that support the program also are loaded.

1.9.5 Phase 6: Execution

Finally, in the last phase (shown in the following diagram), the computer, under the control of its CPU, **executes** the program one instruction at a time:



To load and execute the program on a Linux system, type `./a.out` at the Linux prompt and press *Enter*.

1.9.6 Problems That May Occur at Execution Time

Programs do not always work on the first try. Each of the preceding phases can fail because of various errors that we'll discuss. For example, an executing program might attempt to divide by zero (an illegal operation on computers just as in arithmetic). This would cause the computer to display an error message. You would then return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections work properly.

Errors such as division-by-zero that occur as programs run are called runtime errors or execution-time errors. Divide-by-zero is generally a fatal error that causes the program to terminate immediately without successfully performing its job. Nonfatal errors allow programs to run to completion, often producing incorrect results.

1.9.7 Standard Input, Standard Output and Standard Error Streams

Most C programs input and/or output data. Certain C functions take their input from **`stdin`** (the **standard input stream**), which is normally the keyboard. Data is often output to **`stdout`** (the **standard output stream**), which is normally the computer screen. When we say that a program prints a result, we normally mean that the result is displayed on a screen. Data also may be output to devices such as disks and printers. There's also a **standard error stream** referred to as **`stderr`**, which is normally connected to the screen and used to display error messages. It's common to route regular output data, i.e., `stdout`, to a device other than the screen while keeping `stderr` assigned to the screen so that the user can be immediately informed of errors.

Self Check

1 *(Fill-In)* C programs typically go through six phases to be executed: _____, _____, _____, _____, _____ and _____.
Answer: edit, preprocess, compile, link, load, execute.

2 *(Fill-In)* A(n) _____ occurs when the compiler cannot recognize a statement because it violates the rules of the language.
Answer: syntax error.

3 *(Fill-In)* Errors that occur as a program runs are called _____ or execution-time errors
Answer: runtime errors.

1.10 Test-Driving a C Application in Windows, Linux and macOS

In this section, you'll compile, run and interact with your first C application—a guess-the-number game, which picks a random number from 1 to 1000 and prompts you to guess it. If you guess correctly, the game ends. If you guess incorrectly, the application indicates whether your guess is higher or lower than the correct number. There's no limit to your number of guesses, but you should be able to guess a number from 1 to 1000 correctly in 10 or fewer tries. There's some nice computer science behind this game—in a later chapter, you'll explore the binary search technique

You'll create this application in Chapter 5's exercises. Usually, this application randomly selects the correct answers. We disabled random selection for the test-drives. The application uses the same correct answer every time you run it. That way, you can use the same guesses we use and see the same results. This answer may vary by compiler.

Summary of the Test-Drives

We'll demonstrate creating a C application using:

- Microsoft Visual Studio 2019 Community edition for Windows (Section 1.10.1).
- Clang in Xcode on macOS (Section 1.10.2).
- GNU gcc in a shell on Linux (Section 1.10.3).
- GNU gcc in a shell running inside the GNU Compiler Collection (GCC) Docker container (Section 1.10.4).

You need to read only the section that corresponds to your setup.

Many development environments are available in which you can compile, build and run C applications. If your course uses a different tool from those we demonstrate here, consult your instructor for information on that tool.

1.10.1 Compiling and Running a C Application with Visual Studio 2019 Community Edition on Windows 10

In this section, you'll run a C program on Windows using Microsoft Visual Studio 2019 Community edition. Several versions of Visual Studio are available. In some versions, the options, menus and instructions we present might differ slightly. From this point forward, we'll simply say "Visual Studio" or "the IDE."

Step 1: Checking Your Setup

If you have not already done so, read the Before You Begin section of this book for instructions on installing the IDE and downloading the book's code examples.

Step 2: Launching Visual Studio

Launch Visual Studio from the **Start** menu. Dismiss the initial Visual Studio window by pressing the *Esc* key. **Do not** click the **X** in the upper-right corner, as that will terminate Visual Studio. You can access this window at any time by selecting **File > Start Window**. We use **>** to indicate selecting a menu item from a menu, so **File > Open** means "select the **Open** menu item from the **File** menu."

Step 3: Creating a Project

A **project** is a group of related files, such as the C source-code files that compose an application. Visual Studio organizes applications into projects and **solutions**, which contain one or more projects. Programmers use multiple-project solutions to create large-scale applications. Our examples require only single-project solutions. For our code examples, you'll begin with an **Empty Project** and add files to it. To create a project:

1. Select **File > New > Project...** to display the **Create a new project** dialog.
2. Select the **Empty Project** template with the tags **C++**, **Windows** and **Console**. Visual Studio does not have a C compiler, but its Visual C++ compiler can compile most C programs. The template we use here is for programs that execute at the command line in a Command Prompt window. Depending on

your Visual Studio version and the installed options, there may be many other project templates. You can filter your choices using the **Search for templates** textbox and the drop-down lists below it. Click **Next** to display the **Configure your new project** dialog.

3. Provide a **Project name** and **Location**. For the **Project name**, we specified `c_test`. For the **Location**, we selected this book's `examples` folder, which we assume is in your user account's `Documents` folder. Click **Create** to open your new project in Visual Studio.

At this point, Visual Studio creates your project, places its folder in

`C:\Users\`*YourUserAccount*`\Documents\examples`

(or the folder you specified) and opens Visual Studio's main window.

When you edit C code, Visual Studio displays each file as a separate tab within the window. The **Solution Explorer**—docked to Visual Studio's left or right side—is for viewing and managing your application's files. In this book's examples, you'll typically place each program's code files in the **Source Files** folder. If the **Solution Explorer** is not displayed, you can display it by selecting **View > Solution Explorer**.

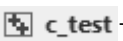
Step 4: Adding the `GuessNumber.c` File into the Project

Next, let's add the file `GuessNumber.c` to the project. In the **Solution Explorer**:

1. Right-click the **Source Files** folder and select **Add > Existing Item...**.
2. In the dialog that appears, navigate to the `ch01` subfolder of the book's `examples` folder, select `GuessNumber.c` and click **Add**.[37]

Step 5: Configuring Your Project's Compiler Version and Disabling a Microsoft Error Message

The Before You Begin section mentioned that Visual C++ can compile most C programs. The Visual C++ compiler supports several C++ standard versions. We'll use Microsoft's C++17 compiler, which we must configure in our project's settings:

1. Right-click the project's node—  —in the **Solution Explorer** and select **Properties** to display the project's **c_test Property Pages** dialog.
2. In the **Configuration** drop-down list, change **Active(Debug)** to **All Configurations**. In the **Platform** drop-down list, change **Active(Win32)** to **All Platforms**.
3. In the left column, expand the **C/C++** node, then select **Language**.
4. In the right column, click in the field to the right of **C++ Language Standard**, click the down arrow, then select **ISO C++17 Standard (/std:c++17)**.[38]

---

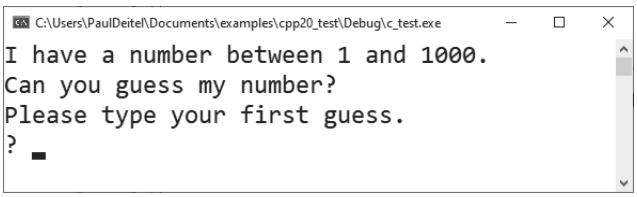
37. For the multiple-source-code-file programs that you'll see in later chapters, select all the files for a given program. When you begin creating programs yourself, you can right-click the **Source Files** folder and select **Add > New Item...** to display a dialog for adding a new file. You'll need to change the filename extension from `.cpp` to `.c` for your C program files.
38. At the time of this writing, Microsoft was still completing its support for the C++20 standard. Once available, you should choose **ISO C++20 Standard (/std:c++20)**.

5. In the left column, in the **C/C++** node, select **Preprocessor**.
6. In the right column, at the end of the value for **Preprocessor Definitions**, insert
   ```
   ;_CRT_SECURE_NO_WARNINGS
   ```
7. In the left column, in the **C/C++** node, select **General**.
8. In the right column, click in the field to the right of **SDL checks**, click the down arrow, then select **No (/sdl-)**.
9. Click **OK** to save the changes.

Items 6 and 8 above eliminate Microsoft Visual C++ warning and error messages for several C library functions we use throughout this book. We'll say more about this issue in Section 3.13.

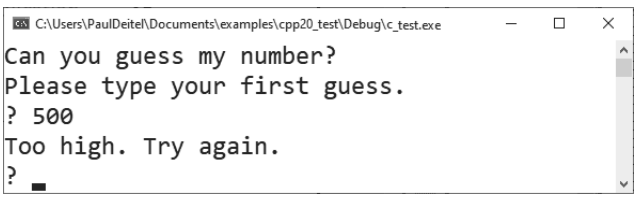
Step 6: Compiling and Running the Project

Next, let's compile and run the project so you can test-drive the application. Select **Debug > Start without debugging** or type *Ctrl* + *F5*. If the program compiles correctly, Visual Studio opens a Command Prompt window and executes the program. We changed the Command Prompt's color scheme[39] and font size for readability:



Step 7: Entering Your First Guess

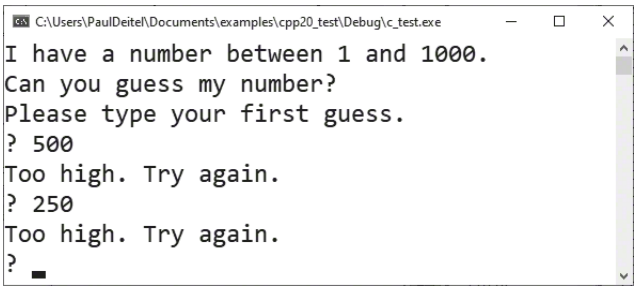
At the `?` prompt, type `500` and press *Enter*. The application displays, `"Too high. Try again."` to indicate that the value you entered is greater than the number the application chose as the correct guess:



Step 8: Entering Another Guess

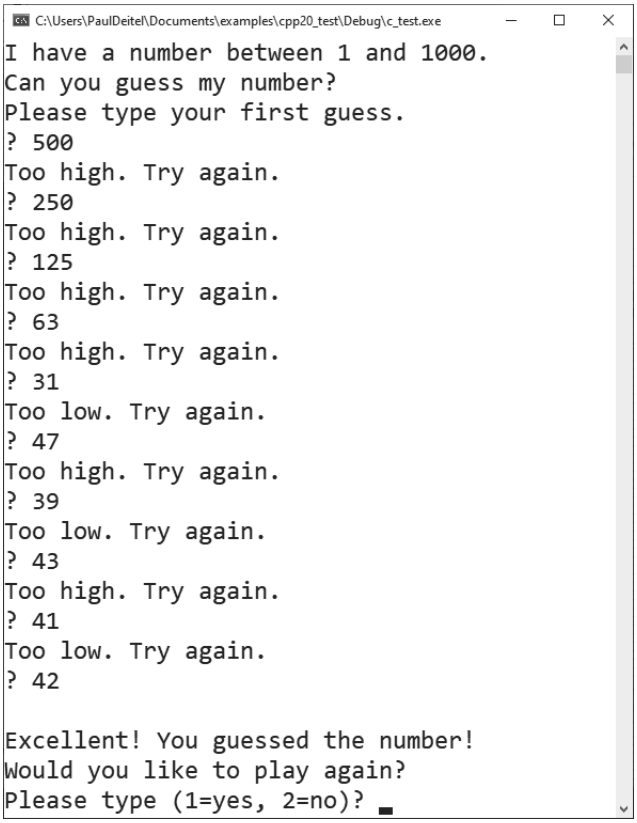
At the next prompt, type `250` and press *Enter*. The application displays, `"Too high. Try again."` to indicate that the value you entered is greater than the correct guess:

39. If you'd like to modify the **Command Prompt** colors on your system, right click the title bar and select **Properties**. In the **"Command Prompt" Properties** dialog, click the **Colors** tab, and select your preferred text and background colors.



Step 9: Entering Additional Guesses

Continue to play the game by entering values until you guess the correct number. When you guess correctly, the application displays, `"Excellent! You guessed the number!"`:



Step 10: Playing the Game Again or Exiting the Application

After you guess the correct number, the application asks if you'd like to play another game. At the `"Please type (1=yes, 2=no)?"` prompt, enter `1` to play again, which chooses a new number to guess. Enter `2` if you wish to terminate the application. Each time you execute this application (*Step 6*), it will choose the same numbers for you to guess. To play a randomized version of the game, use the version of `GuessNumber.c` in the `ch01` folder's `randomized_version` subfolder.

Reusing This Project for Subsequent Examples

You can follow the steps in this section to create a separate project for every application in the book. However, for our examples, you may find it more convenient to reuse this project by removing the `GuessNumber.c` program from the project, then adding another C program. To remove a file from your project (but not your system), select it in the **Solution Explorer**, then press *Del* (or *Delete*). Repeat *Step 4* to add a different program to the project.

Using Ubuntu Linux in the Windows Subsystem for Linux

Some Windows users may want to use the GNU `gcc` compiler on Windows, especially for the few programs in this book that Visual C++ cannot compile. You can do this using the GNU Compiler Collection Docker container (Section 1.10.4), or you can use `gcc` in Ubuntu Linux running in the Windows Subsystem for Linux. To install the Windows Subsystem for Linux, follow the instructions at

```
https://docs.microsoft.com/en-us/windows/wsl/install-win10
```

Once you install and launch Ubuntu on your Windows System, you can use the following command to change to the folder containing the test-drive code example on your Windows system:

```
cd /mnt/c/Users/YourUseName/Documents/examples/ch01
```

Then you can continue with *Step 2* in Section 1.10.3.

1.10.2 Compiling and Running a C Application with Xcode on macOS

In this section, you'll run a C program on a macOS using the Clang compiler in Apple's Xcode IDE.

Step 1: Checking Your Setup

If you have not already done so, read the Before You Begin section of this book for instructions on installing the IDE and downloading the book's code examples.

Step 2: Launching Xcode

Open a Finder window, select **Applications** and double-click the Xcode icon (). If this is your first time running Xcode, the **Welcome to Xcode** window appears. Close this window by clicking the **X** in the upper-left corner—you can access it at any time by selecting **Window > Welcome to Xcode**. We use the **>** character to indicate selecting a menu item from a menu. For example, the notation **File > Open...** means "select the **Open...** menu item from the **File** menu."

Step 3: Creating a Project

A **project** is a group of related files, such as the C source-code files that compose an application. The Xcode projects we created for this book's examples are **Command Line Tool** projects that you'll execute in the IDE. To create a project:

1. Select **File > New > Project...**.

2. At the top of the **Choose a template for your new project** dialog, click **macOS**.
3. Under **Application**, click **Command Line Tool** and click **Next**.
4. For **Product Name**, enter a name for your project—we specified `C_test_Xcode`.
5. In the **Language** drop-down list, select **C**, then click **Next**.
6. Specify where you want to save your project. We selected the `examples` folder containing this book's code examples.
7. Click **Create**.

Xcode creates your project and displays the **workspace window**, initially showing three areas—the **Navigator area** (left), **Editor area** (middle) and **Utilities area** (right).

The left-side **Navigator** area has icons at its top for the *navigators* that can be displayed there. For this book, you'll primarily work with

- **Project** ( )—Shows all the files and folders in your project.
- **Issue** ( )—Shows you warnings and errors generated by the compiler.

Clicking a navigator button displays the corresponding navigator panel.

The middle **Editor** area is for managing project settings and editing source code. This area is always displayed in your workspace window. Selecting a file in the **Project** navigator displays the file's contents in the **Editor** area. You will not use the right-side **Utilities** area in this book. You'll run and interact with the guess-the-number program in the **Debug area**, which will appear below the **Editor** area.

The workspace window's toolbar contains options for executing a program, displaying the progress of tasks executing in Xcode and hiding or showing the left (**Navigator**), right (**Utilities**) and bottom (**Debug**) areas.

Step 4: Deleting the `main.c` File from the Project

By default, Xcode creates a `main.c` source-code file containing a simple program that displays, `"Hello, World!"`. You won't use `main.c` in this test-drive. In the **Project** navigator, right-click the `main.c` file and select **Delete**. In the dialog that appears, select **Move to Trash**. The file will be removed from your system if you empty your trash.

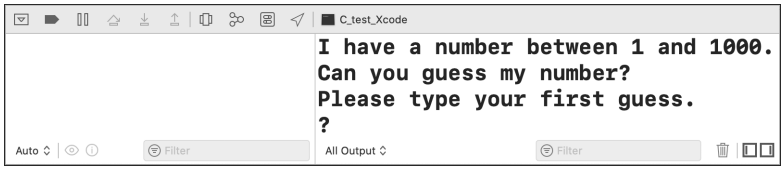
Step 5: Adding the `GuessNumber.c` File into the Project

In a Finder window, open the `ch01` folder in the book's `examples` folder, then drag `GuessNumber.c` onto the **Project** navigator's **C_Test_Xcode** folder. In the dialog that appears, ensure that **Copy items if needed** is checked, then click **Finish**.[40]

Step 6: Compiling and Running the Project

To compile and run the project so you can test-drive the application, simply click the run ( ) button on Xcode's toolbar. If the program compiles correctly, Xcode opens the **Debug** area and executes the program in the right half of the **Debug** area:

40. For the multiple-source-code-file programs that you'll see later in the book, drag all the files for a given program to the project's folder. When you begin creating your own programs, you can right-click the project's folder and select **New File...** to display a dialog for adding a new file.



The application displays, `"Please type your first guess."`, then displays a question mark (?) as a prompt on the next line.

Step 7: Entering Your First Guess

Click in the **Debug** area, then type **500** and press *Return*:

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too low. Try again.
?
```

The application displays, `"Too low. Try again."`, meaning that the value you entered is less than the number the application chose as the correct guess.

Step 8: Entering Another Guess

At the next prompt, enter **750**:

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too low. Try again.
? 750
Too low. Try again.
?
```

The application displays, `"Too low. Try again."`, meaning the value you entered once again is less than the correct guess.

Step 9: Entering Additional Guesses

Continue to play the game until you guess the correct number. When you guess correctly, the application displays, `"Excellent! You guessed the number!"`:

```
? 875
Too high. Try again.
? 812
Too high. Try again.
? 781
Too low. Try again.
? 797
Too low. Try again.
? 805
Too low. Try again.
? 808

Excellent! You guessed the number!
Would you like to play again?
Please type (1=yes, 2=no)?
```

Playing the Game Again or Exiting the Application

After you guess the correct number, the application asks if you'd like to play another game. At the `"Please type (1=yes, 2=no)?"` prompt, enter `1` to play again, which chooses a new number to guess. Enter `2` if you wish to terminate the application. Each time you execute this application (*Step 6*), it will choose the same numbers for you to guess. To play a randomized version of the game, use the version of `GuessNumber.c` in the `ch01` folder's `randomized_version` subfolder.

Reusing This Project for Subsequent Examples

You can follow the steps in this section to create a separate project for every application in the book. For our examples, you may find it more convenient to reuse this project by removing the project's current program, then adding a new one. To remove a file from your project (but not your system), right-click the file in the **Project** navigator and select **Delete**. In the dialog that appears, select **Remove Reference**. You can then repeat *Step 6* to add a different program to the project.

1.10.3 Compiling and Running a C Application with GNU gcc on Linux

For this test-drive, we assume that you read the Before You Begin section and that you placed the downloaded examples in your user account's `Documents` folder.

Step 1: Changing to the ch01 Folder

From a Linux shell, use the `cd` command to change to the `ch01` subfolder of the book's `examples` folder:

```
~$ cd ~/Documents/examples/ch01
~/Documents/examples/ch01$
```

In this section's figures, we use **`bold`** to highlight the text you should type. The prompt in our Ubuntu Linux shell uses a tilde (`~`) to represent your home directory. Each prompt ends with a dollar sign (`$`). The prompt may differ on other Linux distributions.

Step 2: Compiling the Application

Before running the application, you must first compile it:

```
~/Documents/examples/ch01$ gcc -std=c18 GuessNumber.c -o GuessNumber
~/Documents/examples/ch01$
```

The `gcc` command compiles the application:

- The `-std=c18` option indicates that we're using C18—the latest version of the C programming language standard.
- The `-o` option names the executable file (`GuessNumber`) you'll use to run the program.

Step 3: Running the Application

Type `./GuessNumber` at the prompt and press *Enter* to run the program:

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
?
```

The `./` tells Linux to run a file from the current directory. It's needed here to indicate that `GuessNumber` is an executable file.

Step 4: Entering Your First Guess

The application displays, `"Please type your first guess."`, then displays a question mark (`?`) as a prompt on the next line. At the prompt, enter **`500`**—note that the outputs may vary based on the compiler you're using:

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
?
```

The application displays, `"Too high. Try again."`, meaning the value you entered is greater than the number the application chose as the correct guess.

Step 5: Entering Another Guess

At the next prompt, enter **`250`**:

```
~/Documents/examples/ch01$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too low. Try again.
?
```

This time the application displays, `"Too low. Try again."`, meaning the value you entered is less than the correct guess.

Step 6: Entering Additional Guesses

Continue to play the game by entering values until you guess the correct number. When you guess correctly, the application displays, `"Excellent! You guessed the number!"`:

```
Too low. Try again.
? 375
Too low. Try again.
? 437
Too high. Try again.
? 406
Too high. Try again.
? 391
Too high. Try again.
? 383
Too low. Try again.
? 387
Too high. Try again.
? 385
Too high. Try again.
? 384

Excellent! You guessed the number!
Would you like to play again?
Please type (1=yes, 2=no)?
```

Step 7: Playing the Game Again or Exiting the Application

After you guess the correct number, the application asks if you'd like to play another game. At the `"Please type (1=yes, 2=no)?"` prompt, enter `1` to play again, which chooses a new number to guess. Enter `2` if you wish to terminate the application. Each time you execute this application (*Step 3*), it will choose the same numbers for you to guess. To play a randomized version of the game, use the version of `GuessNumber.c` in the `ch01` folder's `randomized_version` subfolder.

1.10.4 Compiling and Running a C Application in a GCC Docker Container Running Natively over Windows 10, macOS or Linux

One of the most convenient cross-platform ways to run GNU's `gcc` compiler is via the GNU Compiler Collection (GCC) Docker container. This section assumes you've installed Docker Desktop (Windows or macOS) or Docker Engine (Linux)—as discussed in the Before You Begin section—and that Docker is running on your computer.

Executing the GNU Compiler Collection (GCC) Docker Container

Open a Command Prompt (Windows), Terminal (macOS/Linux) or shell (Linux), then perform the following steps to launch the GCC Docker container:

1. Use the `cd` command to navigate to the `examples` folder containing this book's examples. Executing the Docker container from here will enable the container to access our code examples.
2. **Windows users:** Launch the GCC Docker container with the command[41]

```
docker run --rm -it -v "%CD%":/usr/src gcc:latest
```

41. A notification will appear asking you to allow Docker to access the files in the current folder. You must allow this; otherwise, you will not be able to access our source-code files in Docker.

3. **macOS/Linux users:** Launch the GCC Docker container with the command

```
docker run --rm -it -v "$(pwd)":/usr/src gcc:latest
```

In the preceding commands:

- `--rm` cleans up the GCC container's resources when you eventually shut it down.
- `-it` runs the container in interactive mode, so you can enter commands to change folders, compile programs using the GNU `gcc` compiler and run programs.
- `-v "%CD%":/usr/src` (Windows) or `-v "$(pwd)":/usr/src` (macOS/Linux) allows the Docker container to access your current folder's files and subfolders via the Docker container's `/usr/src` folder. You can navigate with the `cd` command to subfolders of `/usr/src` to compile and run our programs.
- `gcc:latest` is the container name that you installed in the Before You Begin.[42]

Once the container is running, you'll see a prompt similar to

```
root@67773f59d9ea:/#
```

though `"@67773f59d9ea"` will differ on your computer. The container uses a Linux operating system in which folder separators are forward slashes (/). The prompt displays the current folder location between the : and #.

Changing to the ch01 Folder in the Docker Container

Use the `cd` command to change to the `/usr/src/ch01` folder:

```
root@01b4d47cadc6:/# cd /usr/src/ch01
root@01b4d47cadc6:/usr/src/ch01#
```

You can now compile, run and interact with the `GuessNumber` application in the Docker container, using the commands in Section 1.10.3, *Steps 2–7*.

Terminating the Docker Container

You can terminate the Docker container by typing *Ctrl* + *d* at the container's prompt.

1.11 Internet, World Wide Web, the Cloud and IoT

In the late 1960s, ARPA—the Advanced Research Projects Agency of the United States Department of Defense—rolled out plans for networking the main computer systems of approximately a dozen ARPA-funded universities and research institu-

42. `gcc:latest` is the name of the `gcc` Docker container's latest version at the time you downloaded it onto your machine. Once downloaded, the container does not auto-update. You can keep your GCC container up-to-date with the latest available release by executing `docker pull gcc:latest`. If there's a new version, Docker will download it.

tions. The computers were to be connected with communications lines operating at speeds on the order of 50,000 bits per second, a stunning rate at a time when most people (of the few who even had networking access) were connecting over telephone lines to computers at a rate of 110 bits per second. Academic research was about to take a giant leap forward. ARPA proceeded to implement what quickly became known as the ARPANET, the precursor to today's **Internet**. Today's fastest Internet speeds are on the order of billions of bits per second, with trillion-bits-per-second (terabit) speeds already being tested![43] In 2020, Australian researchers successfully tested a 44.2 terrabits per second Internet connection.[44]

Things worked out differently from the original plan. Although the ARPANET enabled researchers to network their computers, its main benefit proved to be the capability for quick and easy communication via what came to be known as electronic mail (e-mail). This is true even on today's Internet, with e-mail, instant messaging, file transfer and social media, such as Snapchat, Instagram, Facebook and Twitter, enabling billions of people worldwide to communicate quickly and easily.

The protocol (set of rules) for communicating over the ARPANET became known as the **Transmission Control Protocol (TCP)**. TCP ensured that messages, consisting of sequentially numbered pieces called **packets**, were properly delivered from sender to receiver, arrived intact and were assembled in the correct order.

1.11.1 The Internet: A Network of Networks

In parallel with the early evolution of the Internet, organizations worldwide were implementing their own networks for intra-organization (that is, within an organization) and inter-organization (that is, between organizations) communication. A huge variety of networking hardware and software appeared. One challenge was to enable these different networks to communicate with each other. ARPA accomplished this by developing the **Internet Protocol (IP)**, which created a true "network of networks," the Internet's current architecture. The combined set of protocols is now called **TCP/IP**. Each Internet-connected device has an **IP address**—a unique numerical identifier used by devices communicating via TCP/IP to locate one another on the Internet.

Businesses rapidly realized that, by using the Internet, they could improve their operations and offer new and better services to their clients. Companies started spending large amounts of money to develop and enhance their Internet presence. This generated fierce competition among communications carriers and hardware and software suppliers to meet the increased infrastructure demand. As a result, Internet bandwidth—the information-carrying capacity of communications lines—has increased tremendously, while hardware costs have plummeted.

---

43. "BT Testing 1.4 Terabit Internet Connections." Accessed November 1, 2020. `https://testinternetspeed.org/blog/bt-testing-1-4-terabit-internet-connections/`.
44. "Monash, Swinburne, and RMIT universities use optical chip to achieve 44Tbps data speed." Accessed January 9, 2021. `https://www.zdnet.com/article/monash-swinburne-and-rmit-universities-achieve-44tbps-data-speed-using-single-optical-chip/`.

1.11.2 The World Wide Web: Making the Internet User-Friendly

The **World Wide Web** (simply called "the web") is a collection of hardware and software associated with the Internet that allows computer users to locate and view documents (with various combinations of text, graphics, animations, audios and videos) on almost any subject. In 1989, Tim Berners-Lee of CERN (the European Organization for Nuclear Research) began developing **HyperText Markup Language (HTML)**—the technology for sharing information via "hyperlinked" text documents. He also wrote communication protocols such as **HyperText Transfer Protocol (HTTP)** to form the backbone of his new hypertext information system, which he referred to as the World Wide Web.

In 1994, Berners-Lee founded the **World Wide Web Consortium** (**W3C**, `https://www.w3.org`), devoted to developing web technologies. One of the W3C's primary goals is to make the web universally accessible to everyone regardless of disabilities, language or culture.

1.11.3 The Cloud

More and more computing today is done "in the cloud"—that is, using software and data distributed across the Internet worldwide, rather than locally on your desktop, notebook computer or mobile device. **Cloud computing** allows you to increase or decrease computing resources to meet your needs at any given time, which is more cost-effective than purchasing hardware to provide enough storage and processing power to meet occasional peak demands. Cloud computing also saves money by shifting to the service provider the burden of managing these apps (such as installing and upgrading the software, security, backups and disaster recovery).

The apps you use daily are heavily dependent on various **cloud-based services**. These services use massive clusters of computing resources (computers, processors, memory, disk drives, etc.) and databases that communicate over the Internet with each other and the apps you use. A service that provides access to itself over the Internet is known as a **web service**.

Software as a Service

Cloud vendors focus on **service-oriented architecture (SOA)** technology. They provide "as-a-Service" capabilities that applications connect to and use in the cloud. Common services provided by cloud vendors include:[45]

"As-a-Service" acronyms (note that several are the same)	
Big data as a Service (BDaaS)	Platform as a Service (PaaS)
Hadoop as a Service (HaaS)	Software as a Service (SaaS)
Infrastructure as a Service (IaaS)	Storage as a Service (SaaS)

45. For more "as-a-Service" acronyms, see `https://en.wikipedia.org/wiki/Cloud_computing` and `https://en.wikipedia.org/wiki/As_a_service`.

Mashups

The applications-development methodology of **mashups** enables you to rapidly develop powerful software applications by combining (often free) complementary web services and other forms of information feeds. One of the first mashups, `www.housingmaps.com`, combined the real-estate listings from `www.craigslist.org` with Google Maps to show the locations of homes for sale or rent in a given area. Check out `www.housingmaps.com` for some interesting facts, history, articles and how it influenced real-estate industry listings.

ProgrammableWeb (`https://programmableweb.com/`) provides a directory of nearly 24,000 web services and almost 8,000 mashups. They also provide how-to guides and sample code for working with web services and creating your own mashups. According to their website, some of the most widely used web services are Google Maps and others provided by Facebook, Twitter and YouTube.

1.11.4 The Internet of Things

The Internet is no longer just a network of *computers*—it's an **Internet of Things (IoT)**. A *thing* is any object with an IP address and the ability to send, and in some cases receive, data automatically over the Internet. Such *things* include:

- a car with a transponder for paying tolls,
- monitors for parking-space availability in a garage,
- a heart monitor implanted in a human,
- water-quality monitors,
- a smart meter that reports energy usage,
- radiation detectors,
- item trackers in a warehouse,
- mobile apps that can track your movement and location,
- smart thermostats that adjust room temperatures based on weather forecasts and activity in the home, and
- intelligent home appliances.

According to `statista.com`, there are already over 23 billion IoT devices in use today, and there could be over 75 billion IoT devices in 2025.[46]

Self Check

1 ***(Fill-In)*** The _____ was the precursor to today's Internet.

Answer: ARPANET.

46. "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025." Accessed November 1, 2020. `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`.

2 ***(Fill-In)*** The _____ (simply called "the web") is a collection of hardware and software associated with the Internet that allows computer users to locate and view documents (with various combinations of text, graphics, animations, audios and videos).
Answer: World Wide Web.

3 ***(Fill-In)*** In the Internet of Things (IoT), a *thing* is any object with a(n) _____ and the ability to send, and in some cases receive, data over the Internet.
Answer: IP address.

1.12 Software Technologies

As you learn about and work in software development, you'll frequently encounter the following buzzwords:

- Refactoring: Reworking programs to make them clearer and easier to maintain while preserving their correctness and functionality. Many IDEs contain built-in *refactoring tools* to do major portions of the reworking automatically.
- Design patterns: Proven architectures for constructing flexible and maintainable object-oriented software. The field of design patterns tries to enumerate those recurring patterns, encouraging software designers to *reuse* them to develop better-quality software using less time, money and effort.
- Software Development Kits (SDKs)—The tools and documentation that developers use to program applications.

Self Check

1 ***(Fill-In)*** _____ is the process of reworking programs to make them clearer and easier to maintain while preserving their correctness and functionality.
Answer: refactoring.

1.13 How Big Is Big Data?

For computer scientists and data scientists, data is now as crucial as writing programs. According to IBM, approximately 2.5 quintillion bytes (2.5 *exabytes*) of data are created daily,[47] and 90% of the world's data was created in the last two years.[48] The Internet, which will play an important part in your career, is responsible for much of this trend. According to IDC, the global data supply will reach 175 *zettabytes* (equal to 175 trillion gigabytes or 175 billion terabytes) annually by 2025.[49] Consider the following examples of various popular data measures.

47. "Welcome to the world of A.I.." Accessed November 1, 2020. https://www.ibm.com/blogs/watson/2016/06/welcome-to-the-world-of-a-i/.
48. "Accelerate Research and Discovery." Accessed November 1, 2020. https://www.ibm.com/watson/advantages/accelerate.
49. "IDC: Expect 175 zettabytes of data worldwide by 2025." Accessed November 1, 2020. https://www.networkworld.com/article/3325397/storage/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html.

Megabytes (MB)

One megabyte is about one million (actually 2^{20}) bytes. Many of the files we use daily require one or more MBs of storage. Some examples include:

- MP3 audio files—High-quality MP3s range from 1 to 2.4 MB per minute.[50]
- Photos—JPEG format photos taken on a digital camera can require about 8 to 10 MB per photo.
- Video—Smartphone cameras can record video at various resolutions. Each minute of video can require many megabytes of storage. For example, on one of our iPhones, the **Camera** settings app reports that 1080p video at 30 frames-per-second (FPS) requires 130 MB/minute and 4K video at 30 FPS requires 350 MB/minute.

Gigabytes (GB)

One gigabyte is about 1000 megabytes (actually 2^{30} bytes). A dual-layer DVD can store up to 8.5 GB[51], which translates to:

- as much as 141 hours of MP3 audio,
- approximately 1000 photos from a 16-megapixel camera,
- approximately 7.7 minutes of 1080p video at 30 FPS, or
- approximately 2.85 minutes of 4K video at 30 FPS.

The current highest-capacity Ultra HD Blu-ray discs can store up to 100 GB of video.[52] Streaming a 4K movie can use between 7 and 10 GB per hour (highly compressed).

Terabytes (TB)

One terabyte is about 1000 gigabytes (actually 2^{40} bytes). Recent disk drives for desktop computers come in sizes up to 20 TB,[53] which is equivalent to:

- approximately 28 years of MP3 audio,
- approximately 1.68 million photos from a 16-megapixel camera,
- approximately 226 hours of 1080p video at 30 FPS, or
- approximately 84 hours of 4K video at 30 FPS.

Nimbus Data now has the largest solid-state drive (SSD) at 100 TB, which can store five times the 20-TB examples of audio, photos and video listed above.[54]

---

50. "Audio File Size Calculations." Accessed November 1, 2020. `https://www.audiomountain.com/tech/audio-file-size.html`.
51. "DVD." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/DVD`.
52. "Ultra HD Blu-ray." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/Ultra_HD_Blu-ray`.
53. "History of hard disk drives." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/History_of_hard_disk_drives`.
54. "Nimbus Data 100TB SSD – World's Largest SSD." Accessed November 1, 2020. `https://www.cinema5d.com/nimbus-data-100tb-ssd-worlds-largest-ssd/`.

Petabytes, Exabytes and Zettabytes

There are over four billion people online, creating about 2.5 quintillion bytes of data each day[55]—that's 2500 petabytes (each petabyte is about 1000 terabytes) or 2.5 exabytes (each exabyte is about 1000 petabytes). A March 2016 *AnalyticsWeek* article stated that by 2021 there would be over 50 billion devices connected to the Internet (most of them through the Internet of Things; Section 1.11.4) and, by 2020, there would be 1.7 megabytes of new data produced per second for *every person on the planet*.[56] At today's numbers (approximately 7.7 billion people[57]), that's about

- 13 petabytes of new data per second,
- 780 petabytes per minute,
- 46,800 petabytes (46.8 exabytes) per hour, or
- 1,123 exabytes per day—that's 1.123 zettabytes (ZB) per day (each zettabyte is about 1000 exabytes).

That's the equivalent of over 5.5 million hours (over 600 years) of 4K video every day or approximately 116 billion photos every day!

Additional Big-Data Stats

For a real-time sense of big data, check out `https://www.internetlivestats.com`, with various statistics, including the numbers so far today of

- Google searches.
- Tweets.
- Videos viewed on YouTube.
- Photos uploaded on Instagram.

You can click each statistic to drill down for more information.

Some other interesting big-data facts:

- Every hour, YouTube users upload 30,000 hours of video, and almost 1 billion hours of video are watched on YouTube every day.[58]
- Every second, there are 103,777 GBs (or 103.777 TBs) of Internet traffic, 9204 tweets sent, 87,015 Google searches and 86,617 YouTube videos viewed.[59]

---

55. "How Much Data Is Created Every Day in 2020?" Accessed November 1, 2020. `https://techjury.net/blog/how-much-data-is-created-every-day/#gref`.
56. "Big Data Facts." Accessed November 1, 2020. `https://analyticsweek.com/content/big-data-facts/`.
57. "World Population." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/World_population`.
58. "57 Fascinating and Incredible YouTube Statistics." Accessed November 1, 2020. `https://www.brandwatch.com/blog/youtube-stats/`.
59. "Tweets Sent in 1 Second." Accessed November 1, 2020. `http://www.internetlivestats.com/one-second`.

- On Facebook each day, there are 3.2 billion "likes" and comments,[60] and 5 billion emojis sent via Facebook Messenger.[61]

Domo, Inc.'s infographic called "Data Never Sleeps 8.0" shows interesting statistics regarding how much data is generated *every minute*, including:[62]

- 347,222 Instagram posts.
- 500 hours of video uploaded to YouTube.
- 147,000 photos uploaded to Facebook.
- 41,666,667 WhatsApp messages shared.
- 404,444 hours of Netflix video viewed.
- 479,452 users interact with Reddit content.
- 208,333 users participate in Zoom meetings.
- 1,388,889 people make video calls.

Computing Power Over the Years

Data is getting more massive, and so is the computing power for processing it. Today's processor performance is often measured in terms of **FLOPS (floating-point operations per second)**. In the early to mid-1990s, the fastest supercomputer speeds were measured in gigaflops (10^9 FLOPS). By the late 1990s, Intel produced the first teraflop (10^{12} FLOPS) supercomputers. In the early-to-mid 2000s, speeds reached hundreds of teraflops, then in 2008, IBM released the first petaflop (10^{15} FLOPS) supercomputer. Currently, the fastest supercomputer—Fujitsu's Fugaku[63]—is capable of 442 petaflops.[64]

Distributed computing can link thousands of personal computers via the Internet to produce even more FLOPS. In late 2016, the Folding@home network—a distributed network in which people volunteer their personal computers' resources for use in disease research and drug design[65]—was capable of over 100 petaflops. Companies like IBM are now working toward supercomputers capable of exaflops (10^{18} FLOPS).[66]

---

60. "Facebook: 3.2 Billion Likes & Comments Every Day." Accessed November 1, 2020. `https://marketingland.com/facebook-3-2-billion-likes-comments-every-day-19978`.
61. "Facebook celebrates World Emoji Day by releasing some pretty impressive facts." Accessed November 1, 2020. `https://mashable.com/2017/07/17/facebook-world-emoji-day/`.
62. "Data Never Sleeps 8.0." Accessed November 1, 2020. `https://www.domo.com/learn/data-never-sleeps-8`.
63. "Top 500." Accessed December 24, 2020. `https://en.wikipedia.org/wiki/TOP500#TOP_500`.
64. "FLOPS." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/FLOPS`.
65. "Folding@home." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/Folding@home`.
66. "A new supercomputing-powered weather model may ready us for Exascale." Accessed November 1, 2020. `https://www.ibm.com/blogs/research/2017/06/supercomputing-weather-model-exascale/`.

The **quantum computers** now under development theoretically could operate at 18,000,000,000,000,000,000 times the speed of today's "conventional computers"![67] This number is so extraordinary that in one second, a quantum computer theoretically could do staggeringly more calculations than the total that have been done by all computers since the world's first computer appeared. This almost unimaginable computing power could wreak havoc with blockchain-based cryptocurrencies like Bitcoin. Engineers are already rethinking blockchain[68] to prepare for such massive increases in computing power.[69]

The history of supercomputing power is that it eventually works its way down from research labs—where extraordinary amounts of money have been spent to achieve those performance numbers—into "reasonably priced" commercial computer systems and even desktop computers, laptops, tablets and smartphones.

Computing power's cost continues to decline, especially with cloud computing. People used to ask the question, "How much computing power do I need on my system to deal with my *peak* processing needs?" Today, that thinking has shifted to "Can I quickly carve out on the cloud what I need *temporarily* for my most demanding computing chores?" You pay for only what you use to accomplish a given task.

Processing the World's Data Requires Lots of Electricity

Data from the world's Internet-connected devices is exploding, and processing that data requires tremendous amounts of energy. According to a recent article, energy use for processing data in 2015 was growing at 20% per year and consuming approximately three to five percent of the world's power. The article says that total data-processing power consumption could reach 20% by 2025.[70]

Another enormous electricity consumer is the blockchain-based cryptocurrency Bitcoin. Processing just one Bitcoin transaction uses approximately the same amount of energy as powering the average American home for a week. The energy use comes from the process Bitcoin "miners" use to prove that transaction data is valid.[71]

According to some estimates, a year of Bitcoin transactions consumes more energy than many countries.[72] Together, Bitcoin and Ethereum (another popular

67. "Only God can count that fast — the world of quantum computing." Accessed November 1, 2020. https://medium.com/@n.biedrzycki/only-god-can-count-that-fast-the-world-of-quantum-computing-406a0a91fcf4.
68. "Blockchain." Accessed December 24, 2020. https://en.wikipedia.org/wiki/Blockchain.
69. "Is Quantum Computing an Existential Threat to Blockchain Technology?" Accessed November 1, 2020. https://singularityhub.com/2017/11/05/is-quantum-computing-an-existential-threat-to-blockchain-technology/.
70. "'Tsunami of data' could consume one fifth of global electricity by 2025." Accessed November 1, 2020. https://www.theguardian.com/environment/2017/dec/11/tsunami-of-data-could-consume-fifth-global-electricity-by-2025.
71. "One Bitcoin Transaction Consumes As Much Energy As Your House Uses in a Week." Accessed November 1, 2020. https://motherboard.vice.com/en_us/article/ywbbpm/bitcoin-mining-electricity-consumption-ethereum-energy-climate-change.
72. "Bitcoin Energy Consumption Index." Accessed November 1, 2020. https://digiconomist.net/bitcoin-energy-consumption.

blockchain-based platform and cryptocurrency) consume more energy per year than Finland, Belgium or Pakistan.[73]

Morgan Stanley predicted in 2018 that "the electricity consumption required to create cryptocurrencies this year could actually outpace the firm's projected global electric vehicle demand—in 2025."[74] This situation is unsustainable, especially given the huge interest in blockchain-based applications, even beyond the cryptocurrency explosion. The blockchain community is working on fixes.[75,76]

Big-Data Opportunities

The big-data explosion is likely to continue exponentially for years to come. With 50 billion computing devices on the horizon, we can only imagine how many more there will be over the next few decades. It's crucial for businesses, governments, the military, and even individuals to get a handle on all this data.

It's interesting that some of the best writings about big data, data science, artificial intelligence and more are coming out of prominent business organizations, such as J.P. Morgan, McKinsey, Bloomberg and the like. Big data's appeal to big business is undeniable, given the rapidly accelerating accomplishments. Many companies are making significant investments and getting valuable results through technologies like big data, machine learning and natural-language processing. This is forcing competitors to invest as well, rapidly increasing the need for computing professionals with computer-science and data-science experience. This growth is likely to continue for many years.

Self Check

1 ***(Fill-In)*** Today's processor performance is often measured in terms of _____.
Answer: FLOPS (floating-point operations per second).

2 ***(Fill-In)*** The technology that could wreak havoc with blockchain-based cryptocurrencies, like Bitcoin, and other blockchain-based technologies is _____.
Answer: quantum computers.

3 ***(True/False)*** With cloud computing you pay a fixed price for cloud services regardless of how much you use those services.
Answer: False. A key cloud-computing benefit is that you pay for only what you use to accomplish a given task.

---

73. "Ethereum Energy Consumption Index." Accessed November 1, 2020. `https://digiconomist.net/ethereum-energy-consumption`.
74. "Power Play: What Impact Will Cryptocurrencies Have on Global Utilities?" Accessed November 1, 2020. `https://www.morganstanley.com/ideas/cryptocurrencies-global-utilities`.
75. "Blockchains Use Massive Amounts of Energy—But There's a Plan to Fix That." Accessed November 1, 2020. `https://www.technologyreview.com/s/609480/bitcoin-uses-massive-amounts-of-energybut-theres-a-plan-to-fix-it/`.
76. "How to fix Bitcoin's energy-consumption problem." Accessed November 1, 2020. `http://mashable.com/2017/12/01/bitcoin-energy/`.

1.13.1 Big-Data Analytics

Data analytics is a mature and well-developed discipline. The term "data analysis" was coined in 1962,[77] though people have been analyzing data using statistics for thousands of years, going back to the ancient Egyptians.[78] Big-data analytics is a more recent phenomenon—the term "big data" was coined around 1987.[79]

Consider four of the V's of big data[80,81]:

1. Volume—the data the world is producing is growing exponentially.
2. Velocity—the speed at which data is being produced, the speed at which it moves through organizations and the speed at which data changes are growing quickly.[82,83,84]
3. Variety—data used to be alphanumeric (that is, consisting of alphabetic characters, digits, punctuation and some special characters)—today, it also includes images, audios, videos and data from an exploding number of Internet of Things sensors in our homes, businesses, vehicles, cities and more.
4. Veracity—the validity of the data—is it complete and accurate? Can we trust that data when making crucial decisions? Is it real?

Most data is now being created digitally in a *variety* of types, in extraordinary *volumes* and moving at astonishing *velocities*. Moore's Law and related observations have enabled us to store data economically and process and move it faster—and all at rates growing exponentially over time. Digital data storage has become so vast in capacity, and so cheap and small, that we can now conveniently and economically retain *all* the digital data we're creating.[85] That's big data.

---

77. "A Very Short History Of Data Science." Accessed November 1, 2020. https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/.
78. "A Brief History of Data Analysis." Accessed November 1, 2020. https://www.flydata.com/blog/a-brief-history-of-data-analysis/.
79. Diebold, Francis. (2012). On the Origin(s) and Development of the Term "Big Data". SSRN Electronic Journal. 10.2139/ssrn.2152421. https://www.researchgate.net/publication/255967292_On_the_Origins_and_Development_of_the_Term_'Big_Data'.
80. "The Four V's of Big Data." Accessed November 1, 2020. https://www.ibmbigdatahub.com/infographic/four-vs-big-data.
81. There are lots of articles and papers that add many other "V-words" to this list.
82. "Volume, velocity, and variety: Understanding the three V's of big data." Accessed November 1, 2020. https://www.zdnet.com/article/volume-velocity-and-variety-understanding-the-three-vs-of-big-data/.
83. "3Vs (volume, variety and velocity)." Accessed November 1, 2020. https://whatis.techtarget.com/definition/3Vs.
84. "Big Data: Forget Volume and Variety, Focus On Velocity." Accessed November 1, 2020. https://www.forbes.com/sites/brentdykes/2017/06/28/big-data-forget-volume-and-variety-focus-on-velocity.
85. "How Much Information Is There In the World?" Accessed November 1, 2020. http://www.lesk.com/mlesk/ksg97/ksg.html. [The following article pointed us to this Michael Lesk article: https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/.]

The following Richard W. Hamming quote—although from 1962—sets the tone for the rest of this book:

"The purpose of computing is insight, not numbers."[86]

Data science is producing new, deeper, subtler and more valuable insights at a remarkable pace. It's truly making a difference. Big-data analytics is an integral part of the answer.

To get a sense of big data's scope in industry, government and academia, check out the high-resolution graphic[87]—you can click to zoom for easier readability:

```
http://mattturck.com/wp-content/uploads/2018/07/
  Matt_Turck_FirstMark_Big_Data_Landscape_2018_Final.png
```

1.13.2 Data Science and Big Data Are Making a Difference: Use Cases

The data-science field is growing rapidly because it's producing significant results that are making a difference. We enumerate data-science and big-data use cases in the following table. We expect that the use cases and our examples, exercises and projects will inspire interesting term projects, directed-study projects, capstone-course projects and thesis research. Big-data analytics has resulted in improved profits, better customer relations, and even sports teams winning more games and championships while spending less on players.[88,89,90]

Data-science use cases		
anomaly detection	credit scoring	data mining
assisting people with disabilities	crime prevention	data visualization
automated closed captioning	CRISPR gene editing	diagnostic medicine
brain mapping	crop-yield improvement	dynamic driving routes
cancer diagnosis/treatment	customer churn and retention	dynamic pricing
classifying handwriting	customer service agents	electronic health records
computer vision	cybersecurity	emotion detection

86. Hamming, R. W., *Numerical Methods for Scientists and Engineers* (New York: McGraw Hill, 1962). [The following article pointed us to Hamming's book and his quote that we cited: `https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/`.]
87. Turck, M., and J. Hao, "Great Power, Great Responsibility: The 2018 Big Data & AI Landscape," `http://mattturck.com/bigdata2018/`.
88. Sawchik, T., *Big Data Baseball: Math, Miracles, and the End of a 20-Year Losing Streak* (New York: Flat Iron Books, 2015).
89. Ayres, I., *Super Crunchers* (Bantam Books, 2007), pp. 7–10.
90. Lewis, M., *Moneyball: The Art of Winning an Unfair Game* (W. W. Norton & Company, 2004).

Data-science use cases		
facial recognition	personalized medicine	sharing economy
fraud detection	phishing elimination	similarity detection
game playing	pollution reduction	smart cities
health outcome improvement	precision medicine	smart homes
human genome sequencing	predicting disease outbreaks	smart meters
identity-theft prevention	predicting health outcomes	smart thermostats
immunotherapy	predicting weather-sensitive product sales	smart traffic control
intelligent assistants	preventative medicine	social graph analysis
Internet of Things (IoT) and medical device monitoring	preventing disease outbreaks	spam detection
inventory control	real-estate valuation	stock market forecasting
language translation	recommendation systems	summarizing text
location-based services	ride-sharing	telemedicine
malware detection	risk minimization	terrorist attack prevention
marketing analytics	robo financial advisors	theft prevention
natural-language translation	saving energy	trend spotting
new pharmaceuticals	self-driving cars	visual product search
personal assistants	sentiment analysis	voice recognition
		weather forecasting

1.14 Case Study—A Big-Data Mobile Application

In your career, you'll work with many programming languages and software technologies. With its 130 million monthly active users,[91] Google's Waze GPS navigation app is one of the most widely used big-data apps. Early GPS navigation devices and apps relied on static maps and GPS coordinates to determine the best route to your destination. They could not adjust dynamically to changing traffic situations.

Waze processes massive amounts of **crowdsourced data**—that is, the data that's continuously supplied by their users and their users' devices worldwide. They analyze this data as it arrives to determine the best route to get you safely to your destination in the least amount of time. To accomplish this, Waze relies on your smartphone's Internet connection. The app automatically sends location updates to their servers (assuming you allow it to). They use that data to dynamically re-route you based on current traffic conditions and to tune their maps. Users report other information, such as roadblocks, construction, obstacles, vehicles in breakdown lanes, police locations, gas prices and more. Waze then alerts other drivers in those locations.

Waze uses many technologies to provide its services. We're not privy to how Waze is implemented, but we infer below a list of technologies they probably use. For example,

- Most apps created today use at least some open-source software. You'll take advantage of open-source libraries and tools in the case studies.

91. "Waze Communities." Accessed November 1, 2020. `https://www.waze.com/communities`.

- Waze communicates information over the Internet between their servers and their users' mobile devices. Today, such data typically is transmitted in JSON (JavaScript Object Notation) format. Often the JSON data will be hidden from you by the libraries you use.
- Waze uses speech synthesis to speak driving directions and alerts to you, and uses speech recognition to understand your spoken commands. Many cloud vendors provide speech-synthesis and speech-recognition capabilities.
- Once Waze converts a spoken natural-language command to text, it determines the action to perform, which requires natural language processing (NLP).
- Waze displays dynamically updated visualizations, such as alerts and interactive maps.
- Waze uses your phone as a streaming Internet of Things (IoT) device. Each phone is a GPS sensor that continuously streams data over the Internet to Waze.
- Waze receives IoT streams from millions of phones at once. It must process, store and analyze that data immediately to update your device's maps, display and speak relevant alerts and possibly update your driving directions. This requires massively parallel processing capabilities implemented with clusters of computers in the cloud. You can use various big-data infrastructure technologies to receive streaming data, store that big data in appropriate databases and process the data with software and hardware that provide massively parallel processing capabilities.
- Waze uses artificial-intelligence capabilities to perform the data-analysis tasks that enable it to predict the best routes based on the information it receives. You can use machine learning and deep learning, respectively, to analyze massive amounts of data and make predictions based on that data.
- Waze probably stores its routing information in a graph database. Such databases can efficiently calculate shortest routes. You can use graph databases, such as Neo4J.
- Many cars are equipped with devices that help them"see" cars and obstacles around them. These are used to help implement automated braking systems and are a key part of self-driving car technology. Rather than relying on users to report obstacles and stopped cars on the side of the road, navigation apps could take advantage of cameras and other sensors by using deep-learning computer-vision techniques to analyze images "on the fly" and automatically report those items. You can use deep learning for computer vision.

1.15 AI—at the Intersection of Computer Science and Data Science

When a baby first opens its eyes, does it "see" its parent's faces? Does it understand any notion of what a face is—or even what a simple shape is? Babies must "learn" the

world around them. That's what artificial intelligence (AI) is doing today. It's looking at massive amounts of data and learning from it. AI is being used to play games, implement a wide range of computer-vision applications, enable self-driving cars, enable robots to learn to perform new tasks, diagnose medical conditions, translate speech to other languages in near real-time, create chatbots that can respond to arbitrary questions using massive databases of knowledge, and much more. Who'd have guessed just a few years ago that artificially intelligent self-driving cars would be allowed on our roads—or even become common? Yet, this is now a highly competitive area. The ultimate goal of all this learning is **artificial general intelligence**—an AI that can perform intelligence tasks as well as humans can.

Artificial-Intelligence Milestones

Several artificial-intelligence milestones, in particular, captured people's attention and imagination, made the general public start thinking that AI is real and made businesses think about commercializing AI:

- In a 1997 match between **IBM's DeepBlue** computer system and chess Grandmaster Gary Kasparov, DeepBlue became the first computer to beat a reigning world chess champion under tournament conditions.[92] IBM loaded DeepBlue with hundreds of thousands of grandmaster chess games. DeepBlue was capable of using *brute force* to evaluate up to 200 million moves per second![93] This is big data at work. IBM received the Carnegie Mellon University Fredkin Prize, which in 1980 offered $100,000 to the creators of the first computer to beat a world chess champion.[94]
- In 2011, **IBM's Watson** beat the two best human Jeopardy! players in a $1 million match. Watson simultaneously used hundreds of language-analysis techniques to locate correct answers in 200 million pages of content (including all of Wikipedia) requiring four terabytes of storage.[95,96] Watson was trained with machine-learning and reinforcement-learning techniques.[97] Powerful libraries enable you to perform machine-learning and reinforcement-learning in various programming languages.

92. "Deep Blue versus Garry Kasparov." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov`.
93. "Deep Blue (chess computer)." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)`.
94. "IBM Deep Blue Team Gets $100,000 Prize." Accessed November 1, 2020. `https://articles.latimes.com/1997/jul/30/news/mn-17696`.
95. "IBM Watson: The inside story of how the Jeopardy-winning supercomputer was born, and what it wants to do next." Accessed November 1, 2020. `https://www.techrepublic.com/article/ibm-watson-the-inside-story-of-how-the-jeopardy-winning-supercomputer-was-born-and-what-it-wants-to-do-next/`.
96. "Watson (computer)." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/Watson_(computer)`.
97. "Building Watson: An Overview of the DeepQA Project." Accessed November 1, 2020. `https://www.aaai.org/Magazine/Watson/watson.php`, *AI Magazine*, Fall 2010.

- Go—a board game created in China thousands of years ago[98]—is widely considered one of the most complex games ever invented with 10^{170} possible board configurations.[99] To give you a sense of how large a number that is, it's believed that there are (only) between 10^{78} and 10^{82} atoms in the known universe![100,101] In 2015, **AlphaGo**—created by Google's DeepMind group—used *deep learning with two neural networks to beat the European Go champion Fan Hui*. Go is considered to be a far more complex game than chess. Powerful libraries enable you to use neural networks for deep learning.
- More recently, Google generalized its AlphaGo AI to create **AlphaZero**—a game-playing AI that *teaches itself to play other games*. In December 2017, AlphaZero learned the rules of and taught itself to play chess in less than four hours using reinforcement learning. It then beat the world champion chess program, Stockfish 8, in a 100-game match—winning or drawing every game. After *training itself* in Go for just eight hours, AlphaZero was able to play Go vs. its AlphaGo predecessor, winning 60 of 100 games.[102]

AI: A Field with Problems But No Solutions

For many decades, AI has been a field with problems and *no* solutions. That's because once a particular problem is solved, people say, "Well, that's not intelligence; it's just a computer program that tells the computer exactly what to do." However, with machine learning, deep learning and reinforcement learning, we're not pre-programming solutions to *specific* problems. Instead, we're letting our computers solve problems by learning from data—and, typically, lots of it. Many of the most interesting and challenging problems are being pursued with deep learning. Google alone has thousands of deep-learning projects underway.[103,104]

Self Check

1 ***(Fill-In)*** The ultimate goal of AI is to produce a(n) _____.
Answer: artificial general intelligence.

98. "A Brief History of Go." Accessed November 1, 2020. `http://www.usgo.org/brief-history-go`.
99. "Google artificial intelligence beats champion at world's most complicated board game." Accessed November 1, 2020. `https://www.pbs.org/newshour/science/google-artificial-intelligence-beats-champion-at-worlds-most-complicated-board-game`.
100. "How Many Atoms Are There in the Universe?" Accessed November 1, 2020. `https://www.universetoday.com/36302/atoms-in-the-universe/`.
101. "Observable universe." Accessed November 1, 2020. `https://en.wikipedia.org/wiki/Observable_universe#Matter_content`.
102. "AlphaZero AI beats champion chess program after teaching itself in four hours." Accessed November 1, 2020. `https://www.theguardian.com/technology/2017/dec/07/alphazero-google-deepmind-ai-beats-champion-program-teaching-itself-to-play-four-hours`.
103. "Google has more than 1,000 artificial intelligence projects in the works." Accessed November 1, 2020. `http://theweek.com/speedreads/654463/google-more-than-1000-artificial-intelligence-projects-works`.
104. "Google says 'exponential' growth of AI is changing nature of compute." Accessed November 1, 2020. `https://www.zdnet.com/article/google-says-exponential-growth-of-ai-is-changing-nature-of-compute/`.

2 *(Fill-In)* IBM's Watson beat the two best human Jeopardy! players. Watson was trained using a combination of _____ learning and _____ learning techniques.
Answer: machine, reinforcement.

3 *(Fill-In)* Google's _____ taught itself to play chess in less than four hours using reinforcement learning, then beat the world champion chess program, Stockfish 8, in a 100-game match—winning or drawing every game.
Answer: AlphaZero.

Self-Review Exercises

1.1 Fill in the blanks in each of the following statements:
a) Computers process data under the control of instructions called _____.
b) A computer's key logical units are: _____ unit, _____ unit, unit, _____ unit, _____ unit and _____ unit.
c) The three types of programming languages discussed in the chapter are _____, _____ and _____.
d) Programs that translate high-level-language programs into machine language are called _____.
e) _____ is an operating system for mobile devices based on the Linux kernel.
f) A(n) _____ allows a device to respond to motion.
g) C is widely known as the language of the _____ operating system.

1.2 Fill in the blanks in each of the following sentences about the C environment.
a) C programs are normally typed into a computer using a(n) _____.
b) In a C system, a(n) _____ automatically executes before the translation phase begins.
c) The _____ combines the output of the compiler with various library functions to produce an executable image.
d) The _____ transfers the executable image from disk to memory.

Answers to Self-Review Exercises

1.1 a) programs. b) input, output, memory, central processing, arithmetic and logic, secondary storage. c) machine languages, assembly languages, high-level languages. d) compilers. e) Android. f) acceleromoter. g) UNIX.

1.2 a) editor. b) preprocessor. c) linker. d) loader.

Exercises

1.3 Categorize each of the following items as either hardware or software:
a) A microprocessor
b) RAM
c) Microsoft Visual Studio
d) A preprocessor
e) A scanner
f) An Internet browser

1.4 Fill in the blanks in each of the following statements:

a) Translator programs called _____ convert programs written in high-level languages into machine language.

b) A multicore processor implements multiple _____ on a single integrated-circuit chip.

c) A _____ places a program in memory so that it can be executed.

d) Programs in _____ _____ generally consist of strings of numbers that instruct computers to perform their most elementary operations one at a time.

e) A _____ is the smallest data item in a computer.

f) _____ are composed of characters or bytes.

g) A _____ is a collection of data organized for easy access and manipulation.

h) C programs typically go through six phases to be executed. These are _____, _____, _____, _____, _____, and _____.

i) _____ _____ usually allow including other files and various text replacements.

j) _____ and _____ are essentially reusable software components.

1.5 Discuss the purpose of each of the following:

a) `stdin`

b) `stdout`

c) `stderr`

1.6 ***(Gender Neutrality)*** Write the steps of a manual procedure to process a text paragraph and replace gender-specific words with gender-neutral ones. Assuming you've been given a list of gender-specific words and their gender-neutral replacements (e.g., replace "wife" or "husband" with "spouse," replace "man" or "woman" with "person," replace "daughter" or "son" with "child," and so on), explain the procedure you'd use to read through a paragraph of text and manually perform these replacements. How might your procedure generate a strange term like "woperchild" and how might you modify your procedure to avoid this possibility? In Chapter 3, you'll learn that a more formal computing term for "procedure" is "algorithm," and that an algorithm specifies the *steps* to be performed and the *order* in which to perform them.

1.7 ***(Self-Driving Cars)*** Just a few years ago, the notion of driverless cars on our streets would have seemed impossible (in fact, our spell-checking software doesn't recognize the word "driverless"). Many of the technologies you'll study in this book are making self-driving cars possible. They're already common in some areas.

a) If you hailed a taxi and a driverless taxi stopped for you, would you get into the back seat? Would you feel comfortable telling it where you want to go and trust that it would get you there? What safety measures would you want in place? What would you do if the car headed off in the wrong direction?

b) What if two self-driving cars approached a one-lane bridge from opposite directions? What protocol should they go through to determine which car should proceed?

c) What if you're behind a car stopped at a red light, the light turns green, and the car doesn't move? You honk, and nothing happens. You get out of your car and notice that there's no driver. What would you do?
d) If a police officer pulls over a speeding self-driving car in which you're the only passenger, who—or what entity—should pay the fine?
e) One serious concern with self-driving vehicles is that they could potentially be hacked. Someone could set the speed high (or low), which could be dangerous. What if they redirect you to a destination other than what you want?

1.8 ***(Research: Reproducibility)*** A crucial concept in data-science studies is reproducibility, which helps others (and you) reproduce your results. Research reproducibility and list the concepts used to create reproducible results in data-science studies.

1.9 ***(Research: Artificial General Intelligence)*** One of the most ambitious goals in the field of AI is to achieve *artificial general intelligence*—the point at which machine intelligence would equal human intelligence. Research this intriguing topic. When is this forecast to happen? What are some key ethical issues this raises? Human intelligence seems to be stable over long periods. Powerful computers with artificial general intelligence could conceivably (and quickly) evolve intelligence far beyond that of humans. Research and discuss the issues this raises.

1.10 ***(Research: Intelligent Assistants)*** Many companies now offer computerized intelligent assistants, such as IBM Watson, Amazon Alexa, Apple Siri, Google Assistant and Microsoft Cortana. Research these and others and list uses that can improve people's lives. Research privacy and ethics issues for intelligent assistants. Locate amusing intelligent-assistant anecdotes.

1.11 ***(Research: AI in Health Care)*** Research the rapidly growing field of AI big-data applications in health care. For example, suppose a diagnostic medical application had access to every x-ray that's ever been taken and the associated diagnoses—that's surely big data. "Deep Learning" computer-vision applications can work with this "labeled" data to learn to diagnose medical problems. Research deep learning in diagnostic medicine and describe some of its most significant accomplishments. What are some ethical issues of having machines instead of human doctors performing medical diagnoses? Would you trust a machine-generated diagnosis? Would you ask for a second opinion?

1.12 ***(Research: Privacy and Data Integrity Legislation)*** In the Preface, we mentioned HIPAA (Health Insurance Portability and Accountability Act) and the California Consumer Privacy Act (CCPA) in the United States and GDPR (General Data Protection Regulation) for the European Union. Laws like these are becoming more common and stricter. Investigate each of these laws and their effects on your privacy.

1.13 ***(Research: Personally Identifiable Information)*** Protecting users personally identifiable information (PII) is an important aspect of privacy. Research and comment on this issue.

1.14 ***(Research: Big Data, AI and the Cloud—How Companies Use These Technologies)*** For a major organization of your choice, research how they may be using each of the following technologies: AI, big data, the cloud, mobile, natural-language processing,

speech recognition, speech synthesis, database, machine learning, deep learning, reinforcement learning, Hadoop, Spark, Internet of Things (IoT) and web services.

1.15 ***(Research: Raspberry Pi and the Internet of Things)*** It's now possible to have a computer at the heart of just about any device and to connect those devices to the Internet. This has led to the Internet of Things (IoT), which interconnects billions of devices. The Raspberry Pi is an economical computer often at the heart of IoT devices. Research the Raspberry Pi and some of the many IoT applications in which it's used.

1.16 ***(Research: The Ethics of Deep Fakes)*** Artificial-intelligence technologies make it possible to create *deep fakes*—realistic fake videos of people that capture their appearance, voice, body motions and facial expressions. You can have them say and do whatever you specify. Research the ethics of deep fakes. What would happen if you turned on your TV and saw a deep-fake video of a prominent government official or newscaster reporting that a nuclear attack was about to happen? Research Orson Welles and his "War of the Worlds" radio broadcast of 1938, which created mass panic.

1.17 ***(Research: Blockchain—A World of Opportunity)*** Cryptocurrencies like Bitcoin and Ethereum are based on a technology called blockchain, which has seen explosive growth over the last few years. Research blockchain's origin, applications, and how it came to be used as the basis for cryptocurrencies. Research other major applications of blockchain. Over the next many years, there will be extraordinary opportunities for software developers who thoroughly understand blockchain applications development.

1.18 ***(Research: Secure C and the CERT Division of Carnegie Mellon University's Software Engineering Institute)*** Experience has shown that it's challenging to build industrial-strength systems that stand up to attacks. Such attacks can be instantaneous and global in scope. Many of the world's largest companies, government agencies, and military organizations have had their systems compromised. Such vulnerabilities often come from simple programming issues. Building security into software from the start of its development can significantly reduce vulnerabilities. Carnegie Mellon University's Software Engineering Institute (SEI) created CERT (`https://www.sei.cmu.edu/about/divisions/cert/index.cfm`) to analyze and respond promptly to attacks. CERT publishes and promotes secure coding standards to help C programmers and others implement industrial-strength systems that avoid the programming practices that leave systems vulnerable to attacks. The CERT standards evolve as new security issues arise. The SEI CERT C Coding Standard is concerned with "hardening" computer systems and applications to resist attacks. Research CERT and discuss their accomplishments and current challenges. To help you focus on secure C coding practices, Chapters 2–12 and 14 contain Secure C Coding sections that present some key issues and techniques and provide links and references so that you can continue learning.

1.19 ***(Research: IBM Watson)*** IBM is partnering with tens of thousands of companies—including our publisher, Pearson Education—across a wide range of industries. Research some of IBM Watson's key accomplishments and the kinds of challenges IBM and its partners are addressing.