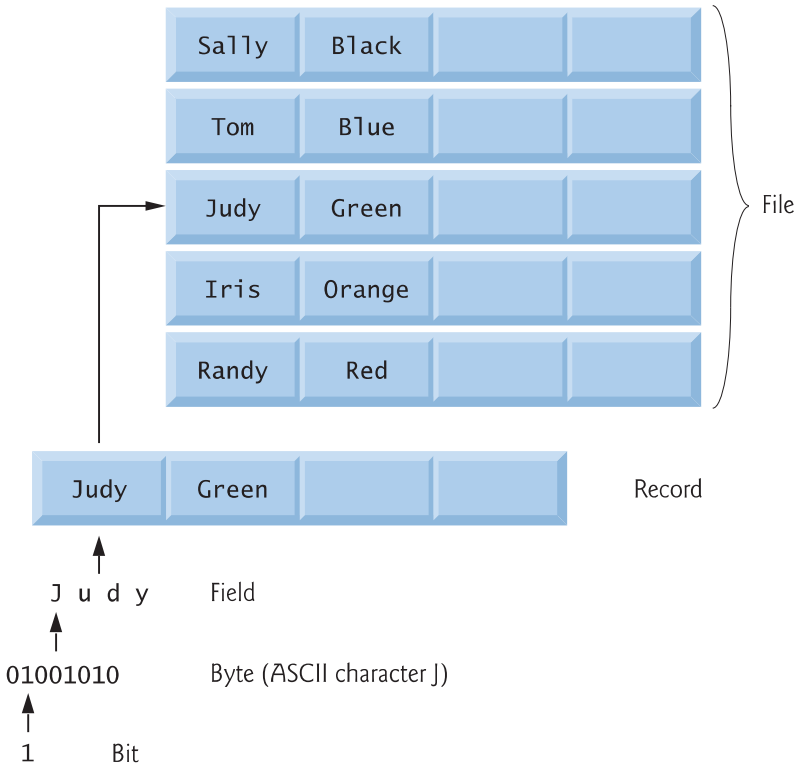

Fig. 1.3 | Data hierarchy.

Level	Description
Bits	The smallest data item in a computer can assume the value 0 or the value 1. Such a data item is called a **bit** (short for "binary digit"—a digit that can assume one of two values). It's remarkable that the impressive functions performed by computers involve only the simplest manipulations of 0s and 1s—*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).
Characters	It's tedious for people to work with data in the low-level form of bits. Instead, they prefer to work with *decimal digits* (0–9), *letters* (A–Z and a–z), and *special symbols* (e.g., $, @, %, &, *, (,), –, +, ", :, ? and /). Digits, letters and special symbols are known as **characters**. The computer's **character set** is the set of all the characters used to write programs and represent data items. Computers process only 1s and 0s, so every character is represented as a pattern of 1s and 0s. The **Unicode®** character set contains characters for many of the world's languages. C++ supports several character sets, including 16-bit Unicode® characters that are composed of two **bytes**, each composed of eight bits. See Appendix B for more information on the **ASCII (American Standard Code for Information Interchange)** character set—the popular subset of Unicode that represents uppercase and lowercase letters, digits and some common special characters.
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 be used to represent a person's name, and a field consisting of decimal digits could represent a person's age.

Fig. 1.4 | Levels of the data hierarchy. (Part 1 of 2.)

Level	Description
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 string of characters) • Address (a string 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. In the preceding example, all the fields belong to the same employee. A company might have many employees and a payroll record for each one.
Files	A **file** is a group of related records. [*Note:* More generally, a file contains arbitrary data in arbitrary formats. 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.] It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information.
Database	A **database** is an electronic collection of data that's organized for easy access and manipulation. The most popular database 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. The data for each student is a record, and the individual pieces of information in each record are the fields. You can search, sort and manipulate the data based on its relationship to multiple tables or databases. For example, a university might use data from the student database in combination with databases of courses, on-campus housing, meal plans, etc.

Fig. 1.4 | Levels of the data hierarchy. (Part 2 of 2.)

1.5 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.

Machine Languages

Any computer can directly understand only its own **machine language** (also called *machine code*), defined by its hardware architecture. Machine languages generally consist of numbers (ultimately reduced to 1s and 0s). Such languages are cumbersome for humans.

Assembly Languages

Programming in machine language was simply too slow and tedious for most programmers. Instead, they began using English-like *abbreviations* to represent elementary opera-