Third AP Edition

Java Methods

Object-Oriented Programming and Data Structures

Maria Litvin
Phillips Academy, Andover, Massachusetts

Gary Litvin
Skylight Software, Inc.

Skylight Publishing
Andover, Massachusetts

Skylight Publishing
9 Bartlet Street, Suite 70
Andover, MA 01810

web: http://www.skylit.com
e-mail: sales@skylit.com
support@skylit.com

Copyright © 2015-2020 by Maria Litvin, Gary Litvin, and Skylight Publishing

This material is provided to you as a supplement to the book *Java Methods*, third AP edition You may print out one copy for personal use and for face-to-face teaching for each copy of the *Java Methods* book that you own or receive from your school. You are not authorized to publish or distribute this document in any form without our permission. **You are not permitted to post this document on the Internet**. Feel free to create Internet links to this document's URL on our web site from your web pages, provided this document won't be displayed in a frame surrounded by advertisement or material unrelated to teaching AP* Computer Science or Java. You are not permitted to remove or modify this copyright notice.

Library of Congress Control Number: 2014922396

ISBN 978-0-9824775-6-4

* AP and Advanced Placement are registered trademarks of The College Board, which was not involved in the production of and does not endorse this book.

The names of commercially available software and products mentioned in this book are used for identification purposes only and may be trademarks or registered trademarks owned by corporations and other commercial entities. Skylight Publishing and the authors have no affiliation with and disclaim any sponsorship or endorsement by any of these product manufacturers or trademark owners.

Oracle, Java, and Java logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates in the U.S. and other countries.

ch 001

Hardware, Software, and the Internet

1.1 Prologue 1-2
1.2 Hardware Overview 1-5
  1.2.1 The CPU 1-5
  1.2.2 Memory 1-6
  1.2.3 Secondary Storage Devices 1-7
  1.2.4 Input and Output Devices 1-8
1.3 Software Overview 1-9
1.4 What Do Software Engineers Do? 1-11
1.5 Representation of Information in Computer Memory 1-15
  1.5.1 Numbers 1-16
  1.5.2 Characters 1-20
1.6 The Internet 1-22
1.7 Summary 1-24
Exercises 📖

Copyright © 2015-2020 by Skylight Publishing

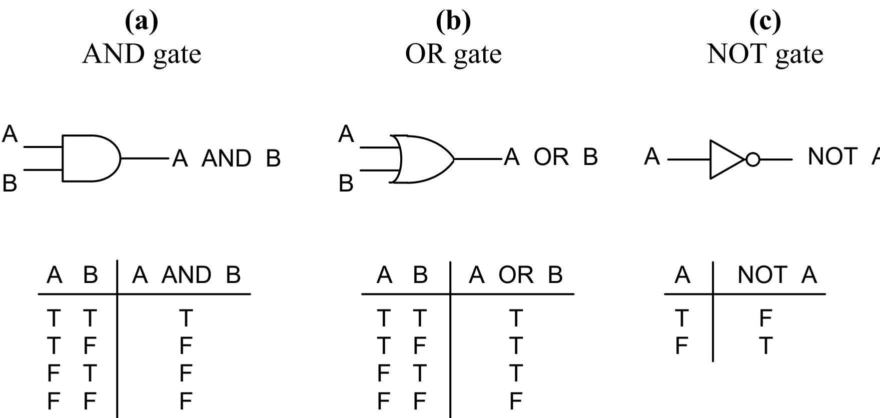
1.1 Prologue

The most important piece of a typical computer is the *Central Processing Unit* or *CPU* [1]. In a personal computer, the CPU is a microprocessor made from a tiny chip of silicon, sometimes as small as half an inch square. Immensely precise manufacturing processes etch a huge number of semiconductor devices, called *transistors*, into the silicon wafer. Each transistor is a microscopic digital switch and together they control, with perfect precision, billions of signals — little spikes of electricity — that arise and disappear every second. The size of the spikes doesn't matter, only their presence or absence. The transistors in the CPU recognize only two states of a signal, "on" or "off," "1" or "0," "true" or "false." This is called *digital electronics* (as opposed to *analog electronics* where the actual amplitudes of signals carry information).

The transistors on a chip combine to form logical devices called *gates*. Gates implement *Boolean* operations (named after the British mathematician George Boole, 1815-1864 [1] who studied the properties of logical relations). For example, an *AND* gate takes two inputs and combines them into one output signal. The output is set to "true" if both the first and the second input are "true," and to "false" otherwise (Figure 1-1-a). In an *OR* gate, the output is set to "true" if either the first or the second (or both) inputs are true (Figure 1-1-b). A *NOT* gate takes one input and sets the output to its opposite (Figure 1-1-c). Note the special shapes used to denote each type of gate.

These three basic types of gates can be combined to make other Boolean operations and logical circuits. Figure 1-2, for example, shows how you can combine AND, OR, and NOT gates to make an *XOR* ("*eXclusive OR*") operation. This operation sets the output to "true" if exactly one of its two inputs is "true." In the late 1940s, John von Neumann, [1] a great mathematician and one of the founding fathers of computer technology, showed that all arithmetic operations can be reduced to AND, OR, and NOT logical operations.

Copyright © 2015-2020 by Skylight Publishing

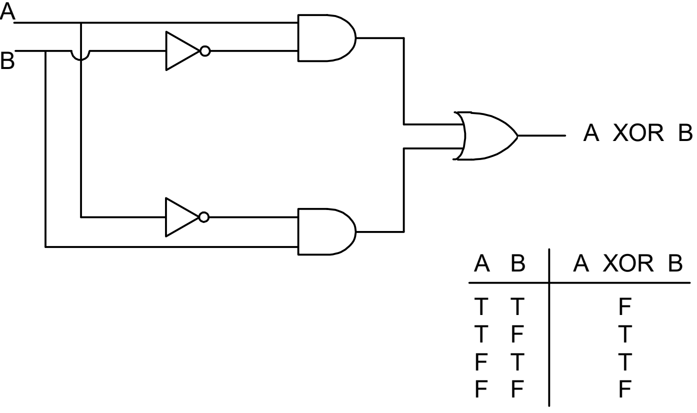


A	B	A AND B
T	T	T
T	F	F
F	T	F
F	F	F

A	B	A OR B
T	T	T
T	F	T
F	T	T
F	F	F

A	NOT A
T	F
F	T

Figure 1-1. AND, OR, and NOT gates



A	B	A XOR B
T	T	F
T	F	T
F	T	T
F	F	F

Figure 1-2. XOR circuit made of AND, OR, and NOT gates

The microprocessor is protected by a small ceramic case mounted on a *PC board* (*Printed Circuit board*) called the *motherboard* [1]. Also on the motherboard are memory chips. The computer memory is a uniform pool of storage units called *bits*. A bit is the smallest possible unit of information, with its value set to 0 or 1. For practical reasons, bits are grouped into groups of eight, called *bytes*. So 8 bits make 1 byte.

Copyright © 2015-2020 by Skylight Publishing

One byte is eight bits.

There is no other structure to memory: the same memory is used to store numbers and letters and sounds and images and programs. All these things must be encoded, one way or another, in sequences of 0s and 1s. A typical personal computer made in the year 2015 had 4 to 6 "gigs" (gigabytes; 1 *gigabyte* is $2^{30} \approx 10^9$ bytes) of *RAM* (Random-Access Memory) packed in a few *SIMMs* (Single In-Line Memory Modules).

The CPU interprets and executes ("runs") computer programs, or sequences of instructions stored in the memory. The CPU fetches the next instruction, interprets its operation code, and performs the appropriate operation. There are instructions for arithmetic and logical operations, for copying bytes from one location to another, and for changing the order of execution of instructions. The instructions are executed in sequence unless a particular instruction tells the CPU to "jump" to another place in the program. Conditional branching instructions tell the CPU to continue with the next instruction or jump to another place depending on the result of the previous operation.

Besides the CPU, a general-purpose computer system also includes *peripheral devices*, which provide input and output and secondary mass storage. In a laptop or tablet computer, the "peripheral" devices are no longer quite so peripheral: a keyboard, a display, a hard drive, a DVD drive, a wireless network adapter, a web cam (camera), a touch pad, a microphone, and speakers are all built into one portable unit.

CPU, memory, peripherals — all of this is called *hardware*. It is a lot of power concentrated in a small device. But to make it useful, to bring life into it, you need programs, *software*. Computer programs are also miracles of engineering, but of a different kind: *software engineering*. They are not cast in iron, nor even silicon, but in intangible texts that can be analyzed, modified, translated from one computer language into another, copied into various media, transmitted over networks, or lost forever. Software is to a computer as tunes are to a band: the best musicians will be silent if they don't have music to play.

Copyright © 2015-2020 by Skylight Publishing

Take this amazing device with its software and connect it to the *Internet*, a network of billions of computers of all kinds connected to each other via communication lines of all kinds and running programs of all kinds, and you end up with a whole new world. Welcome to cyberspace!

In the rest of this chapter we will briefly discuss:

- The main hardware components: CPU, memory, peripheral devices
- What software is
- How numbers and characters are represented in computer memory
- What the Internet is

1.2 Hardware Overview

1.2.1 The CPU

What a CPU can do is defined by its instruction set and internal *registers*. The registers are built-in memory cells that hold operands, memory addresses, and intermediate results. Some of the registers are accessible to the programmer. The instruction set includes instructions for loading CPU registers from memory and storing their values into memory, for logical and arithmetic operations, and for altering the sequence of operations.

Every computer has an internal "clock" that generates electrical pulses at a fixed frequency. All CPU operations are synchronized with the clock's pulses; the duration of CPU operations is measured in *clock cycles*. The CPU's speed depends on the frequency of the clock. The Intel 8088 microprocessor in the original IBM Personal Computer (1981), for example, ran at 4.77 MHz (megahertz, or million pulses per second). Forty years and dozens of computer generations later, CPU chips are pushing into the 4 GHz range (1 GHz, gigahertz, is equal to 1,000 MHz) [1].

A microprocessor CPU connects to memory and to other devices through a set of lines (wires printed on the motherboard) controlled by digital electronics, called a *bus* [1]. A CPU uses a separate *address bus* for specifying memory addresses and a *data bus* for reading and writing memory values. Besides the internal clock speed, the computer's overall performance depends on the speed of the bus transfers and the width of the bus. The Mac 512k computer [1], introduced in 1984, had the Motorola MC68000 8-MHz CPU, 512 kilobytes of RAM, and a 16-bit bus running at 8 MHz, so it could carry 16 bits of data concurrently from memory to the CPU roughly at the

Copyright © 2015-2020 by Skylight Publishing

same speed as the CPU could handle the data. The MacBook Pro offered by Apple Computer Inc. in 2020 had a 3.9 or 4.5 GHz (with Turbo Boost) CPU, 64-bit bus, and up to 16 gigabytes of RAM.

1.2.2 Memory

Each byte of memory has a unique address that can be used to fetch the value stored in that byte or write a new value into it. A CPU does not have to read or write memory bytes sequentially: bytes can be accessed in any arbitrary order. This is why computer memory is called *random-access memory* or *RAM*. This is similar to a CD where you can choose any track to play, as opposed to a tape that has to be played in sequence.

Table 1-1 shows units used to describe memory size. (Powers of 2 have a special significance in computer technology for a reason that will become clear shortly.)

Name	Abbrev.	Bytes	Equals
Kilobyte	KB	$2^{10} = 1{,}024$	
Megabyte	MB	$2^{20} = 1{,}048{,}576$	1024 KB
Gigabyte	GB	$2^{30} = 1{,}073{,}741{,}824 \approx 10^{9}$	1024 MB
Terabyte	TB	$2^{40} \approx 10^{12}$	1024 GB
Petabyte	PB	$2^{50} \approx 10^{15}$	1024 TB
Exabyte	EB	$2^{60} \approx 10^{18}$	1024 PB
Zettabyte	ZB	$2^{70} \approx 10^{21}$	1024 EB
Yottabyte	YB	$2^{80} \approx 10^{24}$	1024 ZB

Table 1-1. Memory units [1, 2]

In the early days, designers of personal computers thought 64KB of RAM would suffice for the foreseeable future. An additional hardware mechanism, the *segment registers*, had to be added to the later versions of Intel's microprocessors to access a larger memory space, up to one megabyte, while maintaining compatibility with the old programs. But the one megabyte limit very quickly proved inadequate too. A 32-bit memory address bus allows programs to directly address four *gigabytes* (GB) of memory. This limit has now been exceeded, too. In 64-bit systems, the CPU can

Copyright © 2015-2020 by Skylight Publishing

address 2^{64} bytes, which is over 4 billion gigabytes, and this should suffice for a while.

A small part of the computer memory is permanent non-erasable memory, known as *read-only memory* or *ROM*. ROM contains, among other things, the initialization code that *boots up* the *operating system* (that is, loads into memory the *boot record* or initialization code from the disk and passes control to it). In modern computers ROM is not exactly "read-only": its content can be changed slowly, which also provides a measure of security against inadvertent or malicious changes.

An executable program has to be loaded from a hard disk or another peripheral device (or from the Internet) into RAM before it can run.

ROM solves the "first-program" dilemma — some program must already be running to load any other program into memory. The *operating system* is a program that has the job of loading and executing other programs. In a personal computer, ROM also contains the computer configuration program and hardware diagnostic programs that check various computer components. The ROM BIOS (Basic Input Output System) contains programs for controlling the keyboard, display, disk drives, and other devices. A special small memory — EPROM (Erasable Programmable ROM) — preserves system configuration data when the power is off.

1.2.3 Secondary Storage Devices

A computer's RAM has only limited space, and its contents are wiped out when the power is turned off. All the programs and data in a computer system have to be stored in secondary mass storage. The auxiliary storage devices include hard disks, CD-ROM drives, USB flash drives, and other devices. A hard disk can hold hundreds of gigabytes and even terabytes. A CD-ROM can hold more than 700 MB (or up to 2 or 3 GB with data compression). Access to data on these devices is much slower than access to RAM. Flash drives, (also known as "memory sticks") available in 2015 could hold up to 512 GB of data.

The operating system software organizes the data in secondary storage into *files*. A file may contain a set of related data, a program, a document, an image, and so on; it has a unique name. The operating system maintains a *directory* of file names, locations, sizes, dates and times of the last updates, and other attributes.

Thus a "file" is not a hardware but rather a software concept.

Copyright © 2015-2020 by Skylight Publishing

When a program is running, it can read and write data directly to and from files stored on secondary storage devices.

1.2.4 Input and Output Devices

A personal computer receives user input through the keyboard and displays the output on the computer display (also called the *monitor*). In many programs the input is echoed on the screen as you type, creating the illusion that the keyboard is directly connected to the monitor. In fact these are two entirely different devices connected only through the CPU and the currently running program. The keyboard sends to the program digital codes that represent the pressed keys. The program captures these codes and takes appropriate actions, which may include displaying corresponding characters on the screen.

The screen is controlled by a *video adapter* and displays the contents of special video memory in the adapter, called *VRAM*. VRAM is addressable by the CPU and may contain codes, colors and attributes of characters when running in text modes, or colors or intensities of individual *pixels* ("picture elements") in graphics modes. The original IBM PC ran mostly in the text mode. You don't see the text mode much any more — everything is in graphics now, except diagnostic and setup programs.
A *mainframe* computer [1, 2] (a very large multi-user computer) may have hundreds of terminals attached to it. The terminals send keystrokes and receive commands and character codes from the computer through digital transmission lines.

Printers, plotters, touch screens, digitizing tablets, scanners, and other devices receive commands and data from the computer in digital form and may send data or control codes back to the computer according to a specific communications protocol.

Network adapters and modems are essential for getting your computer connected to other computers. Network adapters and cables are used to connect several computers into a LAN (Local Area Network), which in turn may be connected to the Internet.

Special data acquisition devices equipped with *A/D* (*analog-to-digital*) converters [1] allow computers to convert an electrical signal into digital form by frequently sampling the amplitude of the signal and storing the digitized values in memory. *D/A* (*digital-to-analog*) converters perform the reverse transformation: they generate electrical currents from the digitized amplitudes stored in the computer. These devices allow the computer to receive data from all kinds of instruments and to serve as a universal control device in industrial applications and scientific experiments. (In 2010, the Stuxnet computer virus was introduced into programmable controllers in Iran's centrifuges used for uranium enrichment, which led to their destruction.)

Copyright © 2015-2020 by Skylight Publishing

Input and output devices are connected to the computer via hardware *interface* modules that implement specific data transfer protocols. In a personal computer, the interfaces may be built into the motherboard or take the form of special adapter cards that plug into special sockets on the motherboard, called *extension slots*. Devices connected to the computer are usually controlled by special programs called *drivers* that handle all the details and peculiarities of the device and the data transfer protocol. Today's personal computers are equipped with several USB (Universal Serial Bus) ports, used for connecting virtually all peripheral devices: mice, printers, digital cameras, external disk drives, and so on. A high-speed USB interface can handle a data transmission rate of up to 480 megabits per second. The SuperSpeed USB 3.0 of today can transmit data ten times faster, at 5 gigabits per second (which is about 15 minutes of high-definition video [1]).

1.3 Software Overview

The term *software* refers to computer programs (now frequently called *apps*); it is also used as an adjective to refer to tasks or functions implemented through programs, as in "software interface," "software fonts," and so on. The line between hardware and software is not always clear. In the modern world, microprocessors are embedded in many objects, from microwave ovens and DVD players to cars and satellites. Their programs are developed using simulation tools on normal computers; when a program is finalized, it is permanently "burned" into ROMs. Such programs are referred to as *firmware*.

A modern computer not only runs individual programs but also maintains a "software environment." This environment involves several functional layers (Figure 1-3). The bottom layer in this environment comprises BIOS, device drivers, interrupt handlers, etc. — programs that directly support hardware devices and functions. The next layer is the *operating system*, a software program that provides computer access services to users and standard support functions to other programs. The top layer is software applications (word processors, Internet browsers, business, industrial, or scientific applications, games, etc.).

Copyright © 2015-2020 by Skylight Publishing

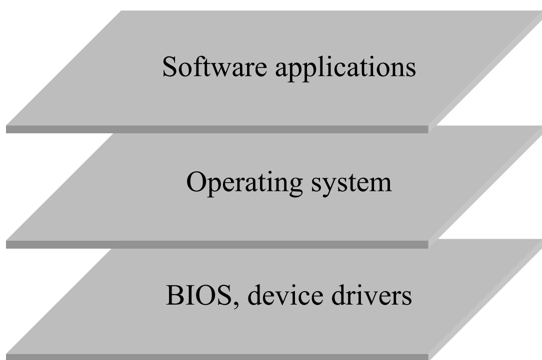


Figure 1-3. Software functional layers

The operating system loads programs into RAM from secondary storage and runs them. On a mainframe, the operating system allows multiple users to work on the computer at once through *time sharing*. In such a multi-user system, one user may be slowly editing a file or entering data on a terminal using only a small fraction of the available CPU time. At the same time another program may be doing "number crunching." A multi-user operating system allocates "time slices" to each program and automatically switches between them. The operating system prioritizes the jobs and swaps segments of programs in and out of memory as needed. A personal computer assumes one user, but contemporary users enjoy *multi-tasking* operating systems that let them keep several programs active concurrently (for example, a word processor, an e-mail program, and a music player).

The operating system also establishes and maintains a file system in secondary storage. Files are organized into a branching structure of directories and subdirectories (also called *folders*). The operating system provides commands and utility programs for navigating through the directory tree. Part of the operating system is a set of *routines* (sets of instructions, callable from other programs) that provide standard service functions to programs. These include functions for creating, reading, and writing files. The operating system *shell* provides a set of user commands, including commands for displaying, copying, deleting, and printing files, executing programs, and so on. Modern operating systems use *GUI* (*Graphical User Interface*, pronounced "gooey"), where commands can be entered by selecting items in menus or by clicking a mouse on an icon that represents a command or an object graphically.

Copyright © 2015-2020 by Skylight Publishing

The top layer of software consists of application programs (apps) that make computers useful to people. In the old days, application programs mostly performed calculations or processed files. If interaction with the user was required, it was accomplished via an unsophisticated text dialog: a computer would display questions (or *prompts*) in plain text and let the user enter responses, then print out the resulting numbers, tables, or reports. Such applications are called *console applications*. In a modern computing environment, users expect apps to have GUI. Menus, buttons, and icons select different functions in the app; dialog boxes and text-edit fields are used to set options and enter data or text. The shift from console applications to GUI has brought changes in the way programs are designed and written and brought about new software development tools and methodologies, such as *OOP* (Object-Oriented Programming).

1.4 What do Software Engineers Do?

Don't they write computer software? In brief, yes, of course. But this simple answer no longer explains what the computer programming (or, to sound more lofty, "software engineering") profession has become. Software professionals create intricate and immensely complex invisible and intangible worlds. These worlds have their laws, their architectures, their aesthetics. These worlds also have "windows" into the "real" world. So many different skills are involved that computer programming has become one of the most versatile and demanding professions. We are not talking about specific technical skills, listed by cryptic acronyms in a typical employment ad for a software engineer. They are important, of course, but they are changing so rapidly that many programmers just learn them on the job. What we are talking about are the core skills that define the profession.

The most important skill is, probably, the ability to absorb new technical information all the time and to put it to work. In no other technical field are things changing so fast. Some of the changes are driven by new hardware and the new capabilities it provides; others by user demands, and some simply by fashion.

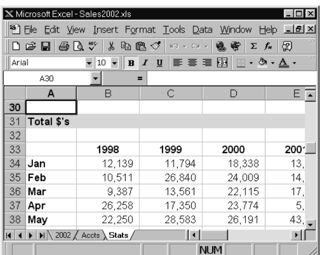
A programmer also needs to be an architect. Computers manipulate data and follow specific instructions written by a programmer. To make a computer perform useful tasks, a programmer must represent both the data and the instructions in coherent structures and procedures. Consider, for example, a word processor application. Before you proceed to write code for it, you need to have an abstract model of a "document." You consider: a document may have text, drawings, images, links, and so on. The user has to be able to edit a document, cut and paste pieces, and so on. Each of these elements and procedures must be reflected in the structure of your program. Sound software design makes it easier to implement and test the program,

Copyright © 2015-2020 by Skylight Publishing

modify it when necessary, share the work among a team of programmers, and reuse pieces of the software in future projects.

Programmers are not building cities, bridges, or cathedrals, of course, and you won't find pieces of computer software exhibited in a museum. Very few people in the world will be able to examine the software structures that a programmer creates. But if you could enter and walk around software systems, you would find all kinds of things, from rather elegant structures to huge monstrosities, often barely standing, with pieces held together by the software equivalent of string and duct tape. A small change in one place upsets the whole structure; more strings and tape and props become necessary to prevent the contraption from collapsing. Perhaps these specimens do belong in a museum after all! For better or worse, software remains mostly invisible, hidden behind thousands of lines of code. It is mysterious, sometimes even full of surprises (Figure 1-4).

Another important skill: a programmer needs to be able to understand *algorithms* and devise new ones. An algorithm is a more or less abstract, formal, and general step-by-step recipe that tells how to perform a certain task or solve a certain problem on a computer. We will discuss algorithms in Chapters 7 and 13.



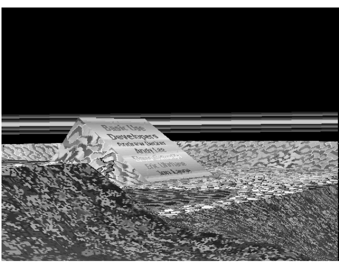


Figure 1-4. In *Excel 97*, if you pressed `Ctrl-G`, typed `L97:X97`, pressed `Enter`, pressed `Tab`, and then clicked on the "Chart Wizard" icon while holding the `Shift` and `Ctrl` keys down, a toy *Flight Simulator* popped up. Expert flying eventually revealed a monolith with the names of the product development team scrolling on it. Such a hidden "feature" is called an "Easter Egg" [1].

The next programmer's skill is using programming languages. People often refer to programmers as "a C++ programmer," "a Java programmer," and so on. They think that knowing a particular programming language is what the programming profession is about. This is not so. First, a programmer is likely to encounter several programming languages over his or her career. Second, as we have seen, a programming job has many different aspects, and mastering a particular language is just one of them.

Copyright © 2015-2020 by Skylight Publishing

Still, ultimately a programmer needs to be able to express his or her brilliant structural designs and clever algorithms in a written program. A program written in a high-level language must obey the very precise *syntax rules* of that language and must also follow less precise but as important stylistic conventions established among professionals. We discuss Java syntax and style in Chapter 3.

But even when a program *compiles* with no errors, there is no guarantee that it works as desired. It may show unexpected behavior, produce incorrect output, or just crash. The programmer then has to figure out what is wrong and fix the problem. This takes yet another important skill: solving mysteries. A programmer must analyze the behavior of the program, hypothesize what may be wrong, either in the algorithm or in the code, then verify his hypothesis and make corrections. Of course, there are software tools, called *debuggers*, that help programmers locate *bugs* (errors) in their programs. But it would be too slow and tedious to use a debugger for every little error. Like a good doctor, a good programmer must develop an intuition for diagnosing and locating bugs quickly.

Believe it or not, there was time when programmers wrote their programs on paper and sent them to keypunch operators who would put them on *punch cards* [1]. Now programmers use sophisticated *software development tools*. Being able to use these tools proficiently is another skill in the programmer's skill set.

In addition to general tools, a development environment for a particular programming language includes libraries of pre-programmed and pre-tested reusable software modules. There is no way to remember their technical details, of course. For example, in the Java 8 release (March 2014), the standard Java library documentation consisted of 15,333 files in 886 folders and took 373 MB of disk space (which is roughly equivalent to a PDF file with 100,000 pages). Luckily the documentation is well organized and available online, so programmers can quickly find the information they need. But like a librarian, a programmer must know where and how to look.

Sometimes programmers forget that there are other people in the world who are not programmers, and that programs are written mostly for those other people, "users." A programmer must make sure that his or her software is "user-friendly," that is, easy to use and appears logical and intuitive to a non-technical user. Designing a friendly and effective *user interface* is another very important programming skill. A good user interface designer knows how to lay out screens, use colors, sound, animation, menus, keyboard commands, printouts and reports, and so on. The user interface is the only part of your software that is visible to the outside world, and this is often how your program is judged by non-technical users.

Copyright © 2015-2020 by Skylight Publishing

Last but not least, programmers must be familiar with the ethical codes for their profession and strive to uphold the highest professional ethical standards (see Chapter 28).

To summarize, these are some of the things that a programmer must be able to do:

- Absorb and use emerging technical information
- Create sound software system architectures
- Understand and devise effective algorithms
- Be proficient with the syntax and style of programming languages
- Diagnose and correct programming errors
- Use software development tools and documentation
- Find and utilize reusable software components
- Design and implement friendly user interfaces
- Uphold the highest standards of professional ethics

Of course, when we said "A programmer must be able to do this; a programmer should know that," we meant an abstract "programmer" who does not exist in the real world. In reality, a number of people are involved in every software development project, and different people may possess different skills. One person may specialize in designing system architectures, another person may be good at devising complex algorithms, a third person can quickly write and debug code, while someone else may not know much about programming but be an excellent designer of user interfaces. In the software development enterprise, there is a need for different types of work and room for people with different interests and skills.

The discipline that focuses on all aspects of software development is called *computer science*. A computer scientist takes a more theoretical view of one or another aspect of software development, looking for general ways to improve the efficiency of algorithms or for better software development methodologies and new programming languages. Unlike the computer scientist, a software engineer (or programmer) works on specific software development projects.

Copyright © 2015-2020 by Skylight Publishing

1.5 Representation of Information in Computer Memory

Computer memory (RAM) is a uniform array of bytes that does not privilege any particular type of information. The memory can contain CPU instructions, numbers and text characters, and any other information that can be represented in digital form. Since a suitable analog/digital (A/D) converter can fairly accurately convert any electrical signal to digital form, any information that can be carried over a wire can be represented in computer memory. This includes sounds, images, motion, and so on (but, so far, excludes taste and smell).

CPU instructions are represented in computer memory in a manner specific to each particular brand of CPU. The first byte or two represent the operation code that identifies the instruction and the total number of bytes in that instruction; the following bytes may represent the values or memory addresses of the operands. How memory addresses are represented depends on the CPU architecture, but they are basically numbers that indicate the absolute sequential number of the byte in memory. A CPU may have special *index registers* that help calculate the actual address in memory for a specified instruction or operand.

The format for numbers is mostly dictated by the CPU, too, because the CPU has instructions for arithmetic operations that expect numbers to be represented in a certain way. Characters (letters, digits, etc.) are represented using one of the several character code systems that have become standard not only for representing text inside computers but also in computer terminals, printers, and other devices. The code assigns each character a number.

Fortunately, high-level programming languages, such as Java, shield computer programmers from the intricacies of how to represent CPU instructions, memory addresses, numbers, and characters.

Representing other types of information is often a matter of a specific application's design. A black-and-white image, for example, may be represented as a sequence of bytes where each bit represents a pixel of the image: 0 for white and 1 for black. The sequence of pixels typically goes from left to right along each horizontal line of the image and then from top to bottom by row. Other memory locations may hold numbers that represent the image dimensions.

Copyright © 2015-2020 by Skylight Publishing

1.5.1 Numbers

Integers from 0 to 255 can be represented in one byte using the binary (base-2) system as follows:

```
 Decimal        Binary

     0         00000000
     1         00000001
     2         00000010
     3         00000011
     4         00000100
     5         00000101
     6         00000110
     7         00000111
     8         00001000
     9         00001001
    10         00001010

   ...            ...

   252         11111100
   253         11111101
   254         11111110
   255         11111111
```

If we use 2 bytes (16 bits), we can represent integers from 0 to $2^{16}–1$ = 65535:

```
 Decimal            Binary

       0        00000000 00000000
       1        00000000 00000001
       2        00000000 00000010
     ...               ...
   65534        11111111 11111110
   65535        11111111 11111111
```

In general, k bits can produce 2^k different combinations of 0s and 1s. Therefore, k bits used as binary digits can represent non-negative integers in the range from 0 to $2^k–1$. A 32-bit memory address can identify 2^{32} = 4,294,967,296 different memory locations. So if we want to be able to address each individual byte, 32-bit addresses cover 4 GB of memory space.

Copyright © 2015-2020 by Skylight Publishing

CPUs perform all their arithmetic operations on binary numbers. A CPU may have instructions that perform 16-bit, 32-bit, or 64-bit arithmetic, for instance. Since it is difficult for a human brain to grasp long sequences of 0s and 1s, programmers who have to deal with binary data often use the *hexadecimal* (or simply "*hex*") representation in their documentation and programs. The hexadecimal system is the base-16 system, which uses 16 "digits." The first ten digits are the usual 0 through 9, with the eleventh through sixteenth digits represented by the letters A through F. A byte can be split into two four-bit *quads*; each quad represents one hex digit, as follows:

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Experienced programmers remember the bit patterns for the sixteen hex digits and can easily convert a binary number into hex and back. It is often convenient to use hex representation as an intermediate step for converting numbers from binary to decimal and back. For example:

$$700_{10} = 2 \cdot 16^2 + 11 \cdot 16 + 12 = 2BC_{16} = \underbrace{0010}_{2}\underbrace{1011}_{B}\underbrace{1100}_{C}{}_2$$

$$\underbrace{110}_{6}\underbrace{1101}_{D}{}_2 = 6D_{16} = 6 \cdot 16 + 13 = 109_{10}$$

Copyright © 2015-2020 by Skylight Publishing

The following examples show a few numbers represented in the decimal, hex, and 16-bit binary systems:

Decimal	Hex	Binary
0	0000	00000000 00000000
1	0001	00000000 00000001
12	000C	00000000 00001100
32	0020	00000000 00100000
128	0080	00000000 10000000
255	00FF	00000000 11111111
256	0100	00000001 00000000
32767	7FFF	01111111 11111111
32768	8000	10000000 00000000
65535	FFFF	11111111 11111111

For a software developer, knowing the hex system is a matter of cultural literacy. In practice, programmers who use a high-level programming language, like Java, don't have to use it very often and there are calculators and programs that can do conversions.

What about negative numbers? The same bit pattern may represent an unsigned (positive) integer and a negative integer, depending on how a particular instruction interprets it. Suppose we use 32-bit binary numbers, but now we decide that they represent signed integers. Positive integers from 0 to $2^{31}-1$ can be represented as before. These use only the 31 least significant bits. As to negative integers, their representation may be *machine-dependent*, varying from CPU to CPU. Many CPUs, including the Intel family, use a method called *two's-complement arithmetic*. In this method, a negative integer x in the range from -1 to -2^{31} is represented the same way as the unsigned binary number $2^{32}-|x|$ where $|x|$ is the absolute value of x. For example, 37 and -37 will be represented as:

```
00000000 00000000 00000000 00100101 =  37₁₀
11111111 11111111 11111111 11011011 = -37₁₀
```

There are dozens of applets on the Internet that illustrate binary representation of numbers and conversions from decimal to binary.

Copyright © 2015-2020 by Skylight Publishing

Real numbers are represented using one of the standard formats expected by the CPU (or a separate floating-point arithmetic unit). Like scientific notation, this representation consists of a fractional part (mantissa) and an exponent part, but here both parts are represented as binary numbers. The IEEE (Institute of Electrical and Electronics Engineers) standard for a 4-byte (32-bit) representation uses 1 bit for the sign, 8 bits for the exponent and 23 bits for the mantissa. 127 is added to the exponent to ensure that negative exponents are still represented by non-negative numbers. This format lets programmers represent numbers in the range from approximately -3.4×10^{38} to 3.4×10^{38} with at least seven digits of precision. Figure 1-5 gives a few examples.

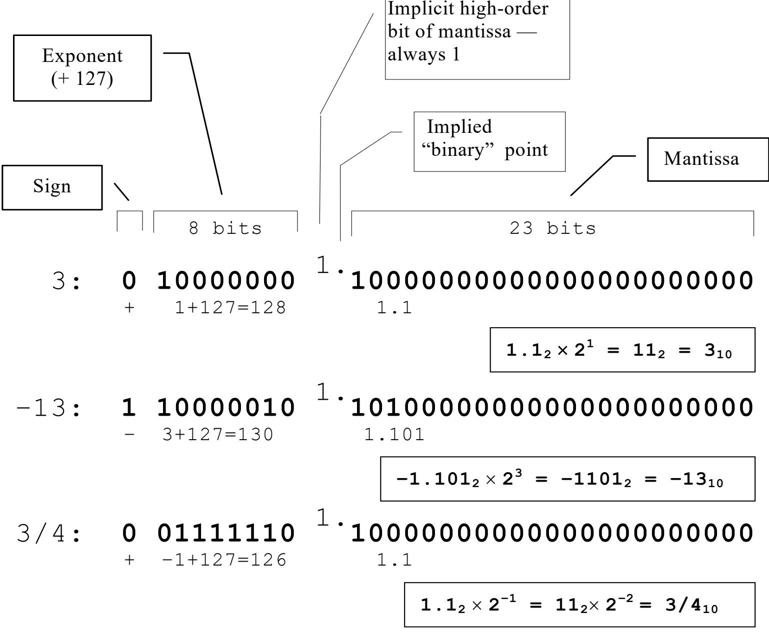


Figure 1-5. IEEE standard representation of 32-bit floating-point numbers

Copyright © 2015-2020 by Skylight Publishing

1.5.2 Characters

Characters are represented by numeric codes. These days most applications use platform-independent Unicode standard for encoding characters [1]. For example, the reason you can see the sentence written in Russian — Многие программы используют Юникод [1] — is that the software that displays this document understands the Unicode representation of Cyrillic characters.

> **Java programs use Unicode internally for representing characters and text strings.**

Unicode uses two bytes per character. It can encode up to 65,000 characters, enough to encode the alphabets of most world languages and many special characters. Unicode has a provision to extend the character set even further, into millions of different codes.

In the old days, the two most common character codes were EBCDIC (Extended Binary Coded Decimal Interchange Code) [1], used in IBM mainframes, and ASCII (American Standard Code for Information Interchange, pronounced as'-kee), used in personal computers, printers and other devices. Both of these use one byte per character. In the PC world, the term *ASCII file* refers to a text file (in which characters are represented in ASCII code), as opposed to a *binary file* that may contain numbers, images, or any other digitized information. Normally you won't find EBCDIC-encoded data on a PC unless the file originated on a mainframe.

Unicode includes ASCII codes as a subset (called "C0 Controls and Basic Latin"). ASCII code defines 128 characters with codes from 0 to 127 and uses only the seven least-significant bits of a byte. Codes from 33 to 127 represent "printable" characters: digits, upper- and lowercase letters, punctuation marks, and so on. 32 (hex 20) is a space.

The first 32 ASCII codes (0-31) are reserved for special control codes. For example, code 13 (hex 0D) is "carriage return" (CR), 10 (hex 0A) is "line feed" (LF), 12 (hex 0C) is "form feed" (FF) and 9 (hex 09) is "horizontal tab" (HT). How control codes are used may depend to some extent on the program or device that processes them. A standard ASCII table, including the more obscure control codes, is presented in Figure 1-6.

Copyright © 2015-2020 by Skylight Publishing

<table>
<tr><th>Hex</th><th>0_</th><th>1_</th><th>2_</th><th>3_</th><th>4_</th><th>5_</th><th>6_</th><th>7_</th></tr>
<tr><td>_0</td><td>0 NUL</td><td>16 DEL</td><td>32 (SPACE)</td><td>48 0</td><td>64 @</td><td>80 P</td><td>96 `</td><td>112 p</td></tr>
<tr><td>_1</td><td>1 SOH</td><td>17 DC1</td><td>33 !</td><td>49 1</td><td>65 A</td><td>81 Q</td><td>97 a</td><td>113 q</td></tr>
<tr><td>_2</td><td>2 STX</td><td>18 DC2</td><td>34 "</td><td>50 2</td><td>66 B</td><td>82 R</td><td>98 b</td><td>114 r</td></tr>
<tr><td>_3</td><td>3 ETX</td><td>19 DC3</td><td>35 #</td><td>51 3</td><td>67 C</td><td>83 S</td><td>99 c</td><td>115 s</td></tr>
<tr><td>_4</td><td>4 EOT</td><td>20 DC4</td><td>36 $</td><td>52 4</td><td>68 D</td><td>84 T</td><td>100 d</td><td>116 t</td></tr>
<tr><td>_5</td><td>5 ENQ</td><td>21 NAK</td><td>37 %</td><td>53 5</td><td>69 E</td><td>85 U</td><td>101 e</td><td>117 u</td></tr>
<tr><td>_6</td><td>6 ACK</td><td>22 SYN</td><td>38 &</td><td>54 6</td><td>70 F</td><td>86 V</td><td>102 f</td><td>118 v</td></tr>
<tr><td>_7</td><td>7 BEL</td><td>23 ETB</td><td>39 '</td><td>55 7</td><td>71 G</td><td>87 W</td><td>103 g</td><td>119 w</td></tr>
<tr><td>_8</td><td>8 BS</td><td>24 CAN</td><td>40 (</td><td>56 8</td><td>72 H</td><td>88 X</td><td>104 h</td><td>120 x</td></tr>
<tr><td>_9</td><td>9 HT</td><td>25 EM</td><td>41)</td><td>57 9</td><td>73 I</td><td>89 Y</td><td>105 i</td><td>121 y</td></tr>
<tr><td>_A</td><td>10 LF</td><td>26 SUB</td><td>42 *</td><td>58 :</td><td>74 J</td><td>90 Z</td><td>106 j</td><td>122 z</td></tr>
<tr><td>_B</td><td>11 VT</td><td>27 ESC</td><td>43 +</td><td>59 ;</td><td>75 K</td><td>91 [</td><td>107 k</td><td>123 {</td></tr>
<tr><td>_C</td><td>12 FF</td><td>28 FS</td><td>44 ,</td><td>60 <</td><td>76 L</td><td>92 \</td><td>108 l</td><td>124 |</td></tr>
<tr><td>_D</td><td>13 CR</td><td>29 GS</td><td>45 -</td><td>61 =</td><td>77 M</td><td>93]</td><td>109 m</td><td>125 }</td></tr>
<tr><td>_E</td><td>14 SO</td><td>30 RS</td><td>46 .</td><td>62 ></td><td>78 N</td><td>94 ^</td><td>110 n</td><td>126 ~</td></tr>
<tr><td>_F</td><td>15 SI</td><td>31 US</td><td>47 /</td><td>63 ?</td><td>79 O</td><td>95 _</td><td>111 o</td><td>127 (NUL)</td></tr>
</table>

Figure 1-6. ASCII code used in personal computers and printers

Copyright © 2015-2020 by Skylight Publishing

1.6 The Internet

You basically need three things to make computers talk to each other:

1. A wire, or a radio (Bluetooth) link between them;
2. Hardware adapters at each *host* (the term used for a computer connected to a network), switches at the network junctions, and other hardware that controls the network;
3. A common language, called a *protocol*, so that the hosts can understand each other.

Of the three, the protocol is probably the most important. Once a reliable and flexible protocol is designed, the hardware and the connections will somehow follow and fall into place. Actually a protocol standard defines hundreds, even thousands of distinct protocols that function at many different levels, or, as network designers say, layers. (Figure 1-7).

The protocols in the bottom layer deal directly with the hardware — the network technology itself and various devices that are connected to it: switches, high-speed adapters, routers, and so on. These hardware protocols are constantly evolving due to changing and competing new technologies and standards set by manufacturers and professional organizations.

Copyright © 2015-2020 by Skylight Publishing

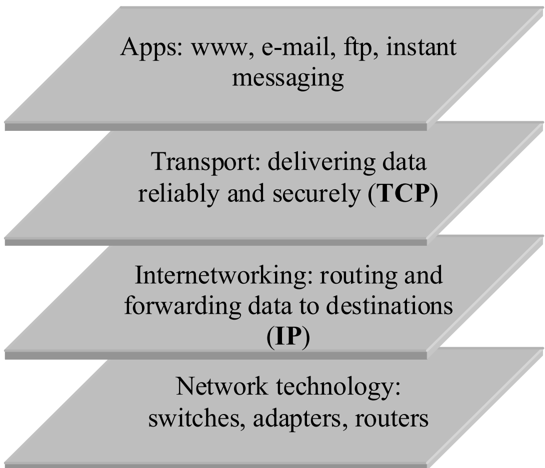


Figure 1-7. TCP/IP protocols in layered network architecture

The next layer deals with routing and forwarding: how to make sure that the information from one host eventually reaches another host. These internetworking protocols must know about the general layout of the network (who is connected to whom), and be efficient and robust for reliable connections. The Internet's internetworking layer is called *IP* (the *Internet Protocol*).

The layer above internetworking, the transport protocol, is responsible for properly handling information from specific applications used on the network and for meeting the requirements of these applications: reliability, security, data compression, and so on. *TCP* (the *Transmission Control Protocol*) is the Internet's transport protocol. The *TCP/IP* combination is what really defines the Internet.

Finally, at the very top layer, there are protocols for network apps such as e-mail (using SMTP — Simple Mail Transfer Protocol), the World Wide Web (using HTTP — HyperText Transfer Protocol), file transfer (using FTP — File Transfer Protocol), instant messaging, remote terminal emulation (*telnet*), and other applications.

If we had to set a birth date for the Internet, it would probably be early September 1969 when the first computer network was tested. The network had only four nodes: University of California in Los Angeles (UCLA), Stanford Research Institute (SRI), University of California in Santa Barbara (UCSB), and the University of Utah in Salt Lake City. Here is how Dr. Leonard Kleinrock [1], a computer science professor at UCLA and one of the Internet pioneers, describes the event. Kleinrock and his group of graduate students hoped to log onto the Stanford computer and try to send it some data. They would start by typing "login" and seeing if the letters appeared on the remote terminal.

> "We set up a telephone connection between us and the guys at SRI...We typed the L and we asked on the phone, "Do you see the L?" "Yes, we see the L," came the response. "We typed the O, and we asked, "Do you see the O." "Yes, we see the O." Then we typed the G, and the system crashed! Yet a revolution had begun."

More precisely, the world's first instance of host-to-host, packet-switched data communications between networked computers had taken place.

The Internet has certainly made a lot of headway since. In 2020, over 90 percent of people in the United States and Canada and over 4.8 billion people worldwide (62%) were online [1]. The best way to explore the Internet and its history is certainly not through a book, but by getting online and browsing [1, 2]. A *browser* is an app that helps its user navigate through the Internet and presents the information that comes from the Internet back to the user. Google *Chrome*, Microsoft *Edge*, *Firefox*, and *Safari* are the four most popular browsers [1].

Copyright © 2015-2020 by Skylight Publishing

Information comes from the Internet in many different formats. The most common one is HTML (HyperText Markup Language) documents — text files with embedded formatting tags in them. You can find many HTML tutorials on the web. A common format for representing documents is PDF (Portable Document Format). Other files use standard formats for representing images, sounds, and so on. A browser and its helper modules, called *plugins*, know how to handle different formats of data and show (or play) the data to the user.

In a matter of just a few years the Internet has become a vast repository of knowledge and information (and misinformation) of all kinds and from all sources. It would be very difficult to find anything in this ocean without some guidance. *Portals* are popular web sites that arrange a large number of Internet links by category and help users navigate to the subjects they need. *Search engines* are programs based in large Internet data centers that analyze and index the contents of web pages and find web sites relevant to user queries.

As more and more individuals and businesses use computing services and resources on remote servers over the Internet, a new technology has emerged: *cloud computing*. Cloud computing may refer to different things. In its simplest form it can mean using a calendar or a calculator that runs on a remote web site or using a word processor or spreadsheet software remotely through the Internet and keeping your documents on a remote server, too, so that you can access them anywhere from any computer. Cloud computing can also mean structuring a whole enterprise's information technology needs around remote services provided by a third party vendor.

The idea is that with the Internet, computing companies become services, which, like water and electricity, are delivered to your home or business in quantity that you need. That way a business does not have to invest in expensive hardware and software; it just buys a service that meets its needs and can be easily scaled up or down according to the demand. Not everyone knows, for example, that amazon.com, besides selling books and consumer goods, sells cloud computing services through its Amazon Web Services (AWS) division [1]. Netflix, a popular movie delivery company, is an AWS's customer.

1.7 Summary

Digital electronics represents information using two states: "on" or "off," "1" or "0." Digital devices, called gates, implement simple logical operations on signals: AND, OR, NOT. All other logical and arithmetic operations can be implemented using these three simple operations.

Copyright © 2015-2020 by Skylight Publishing

The heart of a computer is a CPU (central processing unit) that can perform logical and arithmetic operations. An executable program is a sequence of CPU instructions in machine code. It must be loaded into RAM (random-access memory) before it can run. All instructions and data addresses are encoded in binary sequences of 0s and 1s. The CPU fetches instructions and data from RAM, interprets operation codes, and executes the instructions.

Computer memory (RAM) is arranged into bytes; each byte is 8 bits; each bit can hold either 1 or 0. The size of RAM is measured in kilobytes (1 KB = 2^{10} = 1024 bytes), megabytes (1 MB = 2^{20} = 1,048,576 bytes), or gigabytes (1 GB is over one billion bytes). The contents of RAM are erased when the power is turned off.

Mass storage devices have a larger memory capacity — hundreds of gigabytes or terabytes (1 TB is over one trillion bytes) — and they can hold the information permanently, but access to them is slower. Data in mass storage is arranged into files by the operating system software. The files are stored in a branching system of directories (represented as folders). The operating system also loads and runs applications, provides a GUI (graphical user interface) to users, and provides system services to programs (such as reading and writing files, supporting input devices, etc.).

All kinds of information are represented in the computer memory as sequences of 0s and 1s. Integers are represented as binary numbers. Real numbers use standard floating-point binary representation. Characters are represented as their codes in one of the standard coding systems. The most common codes are ASCII and Unicode. The latter includes thousands of characters from virtually all world alphabets.

The Internet is a network of millions of computers connected in many different ways. The Internet is based on the TCP/IP (Transmission Control Protocol / Internet Protocol), which controls information routing on the network and supports various network applications. Higher-level protocols are used in Internet applications such as e-mail, the World Wide Web, file transfer, and remote terminal emulation. A browser is a program on your computer that processes your requests for Internet connections and delivers and displays the received Internet information, web pages in particular. Portals are popular web sites that list many Internet links, arranged by category. Search engines are Internet indexing services that collect and index information from the Internet and help you find web sites relevant to your requests.

Copyright © 2015-2020 by Skylight Publishing

🕮 Exercises 🕮

The exercises for this chapter are in the book (*Java Methods: Object-Oriented Programming and Data Structures*, 3rd AP Edition, ISBN 978-0-9824775-6-4, Skylight Publishing, 2015 [1]).

Copyright © 2015-2020 by Skylight Publishing