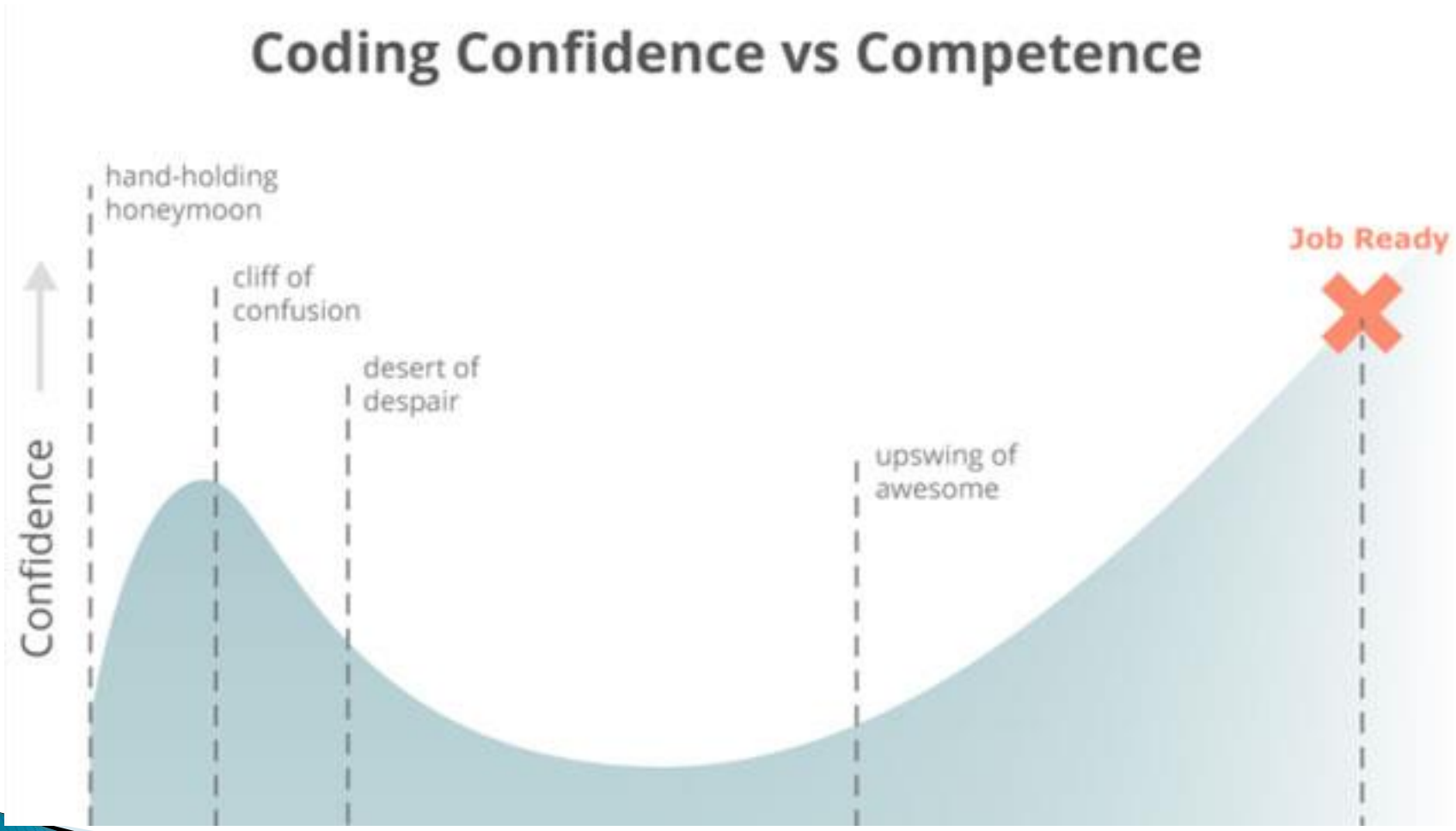
Introduction to Algorithms and Data Structures

Coding



http://lifehacker.com/why-learning-to-code-is-so-hard-and-what-you-can-do-ab-1685229278

1.1 Introduction

- C++—a powerful computer programming language that's appropriate for technically oriented people with little or no programming experience, and for experienced programmers to use in building substantial information systems.
- You'll write instructions commanding computers to perform those kinds of tasks.
- *Software* (i.e., the instructions you write) controls *hardware* (i.e., computers).
- You'll learn *object-oriented programming*—today's key programming methodology.
- You'll create many *software objects* in the real world.

1.1 Introduction (Cont.)

- C++ is one of today's most popular software development languages.
- C++11 and C++14 are the latest versions standardized through the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

1.2 Computers and the Internet in Industry and Research

- Many of the most influential and successful businesses of the last two decades are technology companies, including Apple, IBM, Hewlett Packard, Dell, Intel, Motorola, Cisco, Microsoft, Google, Amazon, Facebook, Twitter, eBay and many more.
- These companies are major employers of people who study computer science, computer engineering, information systems or related disciplines.

Name	Description
Electronic health records	These might include a patient's medical history, prescriptions, immunizations, lab results, allergies, insurance information and more. Making this information available to health care providers across a secure network improves patient care, reduces the probability of error and increases overall efficiency of the health-care system, helping control costs.
Human Genome Project	The Human Genome Project was founded to identify and analyze the 20,000+ genes in human DNA. The project used computer programs to analyze complex genetic data, determine the sequences of the billions of chemical base pairs that make up human DNA and store the information in databases which have been made available over the Internet to researchers in many fields.
AMBER™ Alert	The AMBER (America's Missing: Broadcast Emergency Response) Alert System is used to find abducted children. Law enforcement notifies TV and radio broadcasters and state transportation officials, who then broadcast alerts on TV, radio, computerized highway signs, the Internet and wireless devices. AMBER Alert recently partnered with Facebook, whose users can "Like" AMBER Alert pages by location to receive alerts in their news feeds.

Fig. 1.1 | A few uses for computers. (Part 1 of 5.)

Name	Description
World Community Grid	People worldwide can donate their unused computer processing power by installing a free secure software program that allows the World Community Grid (`http://www.worldcommunitygrid.org`) to harness unused capacity. This computing power, accessed over the Internet, is used in place of expensive supercomputers to conduct scientific research projects that are making a difference—providing clean water to third-world countries, fighting cancer, growing more nutritious rice for regions fighting hunger and more.
Cloud computing	**Cloud computing** allows you to use software, hardware and information stored in the "cloud"—i.e., accessed on remote computers via the Internet and available on demand—rather than having it stored on your personal computer. These services allow you to increase or decrease resources to meet your needs at any given time, so they can be more cost effective than purchasing expensive hardware to ensure that you have enough storage and processing power to meet your needs at their peak levels. Using cloud-computing services shifts the burden of managing these applications from the business to the service provider, saving businesses money.

Fig. 1.1 | A few uses for computers. (Part 2 of 5.)

Name	Description
Medical imaging	X-ray computed tomography (CT) scans, also called CAT (computerized axial tomography) scans, take X-rays of the body from hundreds of different angles. Computers are used to adjust the intensity of the X-rays, optimizing the scan for each type of tissue, then to combine all of the information to create a 3D image. MRI scanners use a technique called magnetic resonance imaging, also to produce internal images noninvasively.
GPS	Global Positioning System (GPS) devices use a network of satellites to retrieve location-based information. Multiple satellites send time-stamped signals to the GPS device, which calculates the distance to each satellite, based on the time the signal left the satellite and the time the signal arrived. This information helps determine the device's exact location. GPS devices can provide step-by-step directions and help you locate nearby businesses (restaurants, gas stations, etc.) and points of interest. GPS is used in numerous location-based Internet services such as check-in apps to help you find your friends (e.g., Foursquare and Facebook), exercise apps such as RunKeeper that track the time, distance and average speed of your outdoor jog, dating apps that help you find a match nearby and apps that dynamically update changing traffic conditions.

Fig. 1.1 | A few uses for computers. (Part 3 of 5.)

Name	Description
Robots	Robots can be used for day-to-day tasks (e.g., iRobot's Roomba vacuuming robot), entertainment (e.g., robotic pets), military combat, deep sea and space exploration (e.g., NASA's Mars rover Curiosity) and more. RoboEarth (`www.roboearth.org`) is "a World Wide Web for robots." It allows robots to learn from each other by sharing information and thus improving their abilities to perform tasks, navigate, recognize objects and more.
E-mail, Instant Messaging, Video Chat and FTP	Internet-based servers support all of your online messaging. E-mail messages go through a mail server that also stores the messages. Instant Messaging (IM) and Video Chat apps, such as Facebook Messenger, AIM, Skype, Yahoo! Messenger, Google Hangouts, Trillian and others allow you to communicate with others in real time by sending your messages and live video through servers. FTP (file transfer protocol) allows you to exchange files between multiple computers (for example, a client computer such as your desktop and a file server) over the Internet.
Internet TV	Internet TV set-top boxes (such as Apple TV, Android TV, Roku and TiVo) allow you to access an enormous amount of content on demand, such as games, news, movies, television shows and more, and they help ensure that the content is streamed to your TV smoothly.

Fig. 1.1 | A few uses for computers. (Part 4 of 5.)

Name	Description
Streaming music services	Streaming music services (such as Apple Music, Pandora, Spotify, Last.fm and more) allow you listen to large catalogues of music over the web, create customized "radio stations" and discover new music based on your feedback.
Game programming	Global video-game revenues are expected to reach $107 billion by 2017 (`http://www.polygon.com/2015/4/22/8471789/worldwide-video-games-market-value-2015`). The most sophisticated games can cost over $100 million to develop, with the most expensive costing half a billion dollars (`http://www.gamespot.com/gallery/20-of-the-most-expensive-games-ever-made/2900-104/`). Bethesda's *Fallout 4* earned $750 million in its first day of sales (`http://fortune.com/2015/11/16/fallout4-is-quiet-best-seller/`)!

Fig. 1.1 | A few uses for computers. (Part 5 of 5.)

1.3 Hardware and Software

- Computers can perform calculations and make logical decisions phenomenally faster than human beings can.
- Today's personal computers can perform billions of calculations in one second—more than a human can perform in a lifetime.
- *Supercomputers* are already performing *thousands of trillions (quadrillions)* of instructions per second!

1.3 Hardware and Software

- Computers process data under the control of sequences of instructions called computer programs.
- These programs guide the computer through ordered actions specified by people called computer programmers.
- The programs that run on a computer are referred to as software.
- You'll learn a key programming methodology that's enhancing programmer productivity, thereby reducing software-development costs—*object-oriented programming.*

1.3 Hardware and Software (Cont.)

- A computer consists of various devices referred to as hardware
  - (e.g., the keyboard, screen, mouse, 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 chips smaller than a fingernail, costing perhaps a few dollars each.
- Silicon-chip technology has made computing so economical that computers have become commodity.

1.3.1 Moore's Law

- For many decades, hardware costs have fallen rapidly.
- Every year or two, the capacities of computers have approximately *doubled* inexpensively.
- This trend often is called Moore's Law, named for the person who identified it in the 1960s, Gordon Moore, co-founder of Intel.

1.3.1 Moore's Law (Cont.)

- 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 disk storage) they have to hold programs and data over longer periods of time, and their processor speeds—the speeds at which they execute their programs (i.e., do their work).
- These increases make computers more capable
  - puts greater demands on programming language designers to innovate

1.3.2 Computer Organization

- Regardless of differences in physical appearance, computers can be envisioned as divided into various **logical units** or sections (Fig. 1.2)

Logical unit	Description
Input unit	This "receiving" section obtains information (data and computer programs) from **input devices** and places it at the disposal of the other units for processing. Most user input is entered into computers through keyboards, touch screens and mouse devices. Other forms of input include receiving voice commands, scanning images and barcodes, reading from secondary storage devices (such as hard drives, DVD drives, Blu-ray Disc™ drives and USB flash drives—also called "thumb drives" or "memory sticks"), receiving video from a webcam and having your computer receive information from the Internet (such as when you stream videos from YouTube® or download e-books from Amazon). Newer forms of input include position data from a GPS device, and motion and orientation information from an *accelerometer* (a device that responds to up/down, left/right and forward/backward acceleration) in a smartphone or game controller (such as Microsoft® Kinect® for Xbox®, Wii™ Remote and Sony® PlayStation® Move).

Fig. 1.2 | Logical units of a computer. (Part 1 of 5.)

Logical unit	Description
Output unit	This "shipping" section takes information the computer has processed and places it on various **output devices** to make it available for use outside the computer. Most information that's output from computers today is displayed on screens (including touch screens), printed on paper ("going green" discourages this), played as audio or video on PCs and media players (such as Apple's iPods) and giant screens in sports stadiums, transmitted over the Internet or used to control other devices, such as robots and "intelligent" appliances. Information is also commonly output to secondary storage devices, such as hard drives, DVD drives and USB flash drives. Popular recent forms of output are smartphone and game controller vibration, and virtual reality devices like Oculus Rift.

Fig. 1.2 | Logical units of a computer. (Part 2 of 5.)

Logical unit	Description
Memory unit	This rapid-access, relatively low-capacity "warehouse" section retains information that has been 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 2 to 16 GB is most common. GB stands for gigabytes; a gigabyte is approximately one billion bytes. A **byte** is eight bits. A bit is either a `0` or a `1`.
Arithmetic and logic unit (ALU)	This "manufacturing" section performs *calculations*, such as addition, subtraction, multiplication and division. It also contains the *decision* mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they're equal. In today's systems, the ALU is implemented as part of the next logical unit, the CPU.

Fig. 1.2 | Logical units of a computer. (Part 3 of 5.)

Logical unit	Description
Central processing unit (CPU)	This "administrative" section coordinates and supervises the operation of the other sections. The CPU tells the input unit when information should be read into the memory unit, tells the ALU when information from the memory unit should be used in calculations and tells the output unit when to send information from the memory unit to certain output devices. Many of today's computers have multiple CPUs and, hence, can perform many operations simultaneously. A **multi-core processor** implements multiple processors on a single integrated-circuit chip—a *dual-core processor* has two CPUs, a *quad-core processor* has four and an *octa-core processor* has eight. Today's desktop computers have processors that can execute billions of instructions per second. To take full advantage of multi-core architecture you need to write multithreaded applications, which we introduce in Section 24.3.

Fig. 1.2 | Logical units of a computer. (Part 4 of 5.)

Logical unit	Description
Secondary storage unit	This is the long-term, high-capacity "warehousing" section. Programs or data not actively being used by the other units normally are placed on secondary storage devices (e.g., your *hard drive*) 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 unit is much less. Examples of secondary storage devices include hard drives, DVD drives and USB flash drives, some of which can hold over 2 TB (TB stands for terabytes; a terabyte is approximately one trillion bytes). Typical hard drives on desktop and notebook computers hold up to 2 TB, and some desktop hard drives can hold up to 6 TB.

Fig. 1.2 | Logical units of a computer. (Part 5 of 5.)

Unit	Bytes	Which is approximately
1 kilobyte (KB)	1024 bytes	10^3 (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

Fig. 1.4 | Byte measurements.

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.*
- These may be divided into three general types:
  - Machine languages
  - Assembly languages
  - High-level languages

1.5 Machine Languages, Assembly Languages and High-Level Languages

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.

1.5 Machine Languages, Assembly Languages and High-Level Languages

Assembly Languages

- English-like *abbreviations* to represent elementary operations. These abbreviations formed the basis of assembly languages.
- *Translator programs* called assemblers were developed to convert early assembly-language programs to machine language.

1.5 Machine Languages, Assembly Languages and High-Level Languages

High-Level Languages

- To speed up the programming process further, high-level languages were developed in which single statements could be written to accomplish substantial tasks.
- Translator programs called compilers convert high-level language programs into machine language.
- Allow you to write instructions that look more like everyday English and contain commonly used mathematical expressions.

1.5 Machine Languages, Assembly Languages and High-Level Languages

- Compiling a high-level language program into machine language can take a considerable amount of computer time.
- Interpreter programs were developed to execute high-level language programs directly (without the need for compilation), although more slowly than compiled programs.
- Scripting languages such as the popular web languages JavaScript and PHP are processed by interpreters.

Performance Tip 1.1

Interpreters have an advantage over compilers in Internet scripting. An interpreted program can begin executing as soon as it's downloaded to the client's machine, without needing to be compiled before it can execute. On the downside, interpreted scripts generally run slower and consume more memory than compiled code.

1.6 C and C++

- C was implemented in 1972 by Dennis Ritchie at Bell Laboratories.
  - Initially became widely known as the UNIX operating system's development language.
  - Today, most of the code for general-purpose operating systems is written in C or C++.
- C++ evolved from C, which is available for most computers and is hardware independent.

1.6 C and C++

- The widespread use of C with various kinds of computers (sometimes called hardware platforms) led to many variations.
- American National Standards Institute (ANSI) cooperated with the International Organization for Standardization (ISO) to standardize C worldwide.
- Joint standard document was published in 1990.

1.6 C++ (Cont.)

- C11
  - Latest ANSI standard for the language.
  - Developed to evolve the C language to keep pace with increasingly powerful hardware and ever more demanding user requirements.
  - Makes C more consistent with C++.
- C++, an extension of C, was developed by Bjarne Stroustrup in 1979 at Bell Laboratories.
- C++ provides a number of features that "spruce up" the C language.
- C++ also provides capabilities for object-oriented programming that were inspired by the Simula simulation programming language.

1.6 C++ (Cont.)

- C++ Standard Library
  - C++ programs consist of pieces called classes and functions.
  - Most C++ programmers take advantage of the rich collections of classes and functions in the C++ Standard Library.
  - Two parts to learning the C++ "world."
    - The C++ language itself (the core language), and
    - How to use the classes and functions in the C++ Standard Library.
  - Many special-purpose class libraries are supplied by independent software vendors.

Software Engineering Observation 1.1

Use a "building-block" approach to create programs. Avoid reinventing the wheel. Use existing pieces wherever possible. Called ***software reuse****, this practice is central to effective object-oriented programming.*

Software Engineering Observation 1.2

When programming in C++, you typically will use the following building blocks: classes and functions from the C++ Standard Library, classes and functions you and your colleagues create, and classes and functions from various popular third-party libraries.

Performance Tip 1.2

Using C++ Standard Library functions and classes instead of writing your own versions can improve program performance, because they're written carefully to perform efficiently. This technique also shortens program development time.

Portability Tip 1.1

Using C++ Standard Library functions and classes instead of writing your own improves program portability, because they're included in every C++ implementation.

1.7 Programming Languages

- In this section, we provide brief comments on several popular programming languages (Fig. 1.5).

Programming language	Description
Fortran	Fortran (FORmula TRANslator) was developed by IBM Corporation in the mid-1950s to be used for scientific and engineering applications that require complex mathematical computations. It's still widely used and its latest versions support object-oriented programming.
COBOL	COBOL (COmmon Business Oriented Language) was developed in the late 1950s by computer manufacturers, the U.S. government and industrial computer users, based on a language developed by Grace Hopper, a career U.S. Navy officer and computer scientist. COBOL is still widely used for commercial applications that require precise and efficient manipulation of large amounts of data. Its latest version supports object-oriented programming.
Pascal	Research in the 1960s resulted in *structured programming*—a disciplined approach to writing programs that are clearer, easier to test and debug and easier to modify than programs produced with previous techniques. The Pascal language developed by Professor Niklaus Wirth in 1971 grew out of this research. It was popular for teaching structured programming for several decades.

Fig. 1.5 | Some other programming languages. (Part 1 of 5.)

Programming language	Description
Ada	Ada, based on Pascal, was developed under the sponsorship of the U.S. Department of Defense (DOD) during the 1970s and early 1980s. The DOD wanted a single language that would fill most of its needs. The Pascal-based language was named after Lady Ada Lovelace, daughter of the poet Lord Byron. She's credited with writing the world's first computer program in the early 1800s (for the Analytical Engine mechanical computing device designed by Charles Babbage). Ada also supports object-oriented programming.
Basic	Basic was developed in the 1960s at Dartmouth College to familiarize novices with programming techniques. Many of its latest versions are object oriented.
Objective-C	Objective-C is an object-oriented language based on C. It was developed in the early 1980s and later acquired by NeXT, which in turn was acquired by Apple. It became the key programming language for the OS X operating system and all iOS-powered devices (such as iPods, iPhones and iPads).

Fig. 1.5 | Some other programming languages. (Part 2 of 5.)

Programming language	Description
Swift	Swift, which was introduced in 2014, is Apple's programming language of the future for developing iOS and OS X applications (apps). Swift is a contemporary language that includes popular programming-language features from languages such as Objective-C, Java, C#, Ruby, Python and others. In 2015, Apple released Swift 2 with new and updated features. According to the Tiobe Index, Swift has already become one of the most popular programming languages. Swift is now *open source* (Section 1.11.2), so it can be used on non-Apple platforms as well.
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 goal of Java is to enable developers to write programs that will run on a great variety of computer systems and computer-controlled devices. This is sometimes called "write once, run anywhere." Java is used to develop large-scale enterprise applications, to enhance the functionality of web servers (the computers that provide the content we see in our web browsers), to provide applications for consumer devices (e.g., smartphones, tablets, television set-top boxes, appliances, automobiles and more) and for many other purposes. Java is also the key language for developing Android smartphone and tablet apps.

Fig. 1.5 | Some other programming languages. (Part 3 of 5.)

Programming language	Description
Visual Basic	Microsoft's Visual Basic language was introduced in the early 1990s to simplify the development of Microsoft Windows applications. Its latest versions support object-oriented programming.
C#	Microsoft's three primary object-oriented programming languages are C# (based on C++ and Java), Visual C++ (based on C++) and Visual Basic (based on the original Basic). C# was developed to integrate the web into computer applications, and is now widely used to develop enterprise applications and for mobile application development.
PHP	PHP is an object-oriented, *open-source* (see Section 1.11.2) "scripting" language supported by a community of developers and used by numerous websites. PHP is platform independent—implementations exist for all major UNIX, Linux, Mac and Windows operating systems.
Python	Python, another object-oriented scripting language, was released publicly in 1991. Developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam (CWI), Python draws heavily from Modula-3—a systems programming language. Python is "extensible"—it can be extended through classes and programming interfaces.

Fig. 1.5 | Some other programming languages. (Part 4 of 5.)

Programming language	Description
JavaScript	JavaScript is the most widely used scripting language. It's primarily used to add programmability to web pages—for example, animations and interactivity with the user. It's provided with all major web browsers.
Ruby on Rails	Ruby—created in the mid-1990s by Yukihiro Matsumoto—is an open-source, object-oriented programming language with a simple syntax that's similar to Python. Ruby on Rails combines the scripting language Ruby with the Rails web application framework developed by the company 37Signals. Their book, *Getting Real* (`http://gettingreal.37signals.com/toc.php`), is a must read for web developers. Many Ruby on Rails developers have reported productivity gains over other languages when developing database-intensive web applications.
Scala	Scala (`www.scala-lang.org/node/273`)—short for "scalable language"—was designed by Martin Odersky, a professor at École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland. Released in 2003, Scala uses both the object-oriented programming and functional programming paradigms and is designed to integrate with Java. Programming in Scala can reduce the amount of code in your applications significantly.

Fig. 1.5 | Some other programming languages. (Part 5 of 5.)

1.8 Introduction to Object Technology

- *Objects*, or more precisely—as we'll see in Chapter 3—the classes objects come from, are essentially *reusable* software components.
  - There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc.
  - Almost any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating).
- Using a modular, object-oriented design-and-implementation approach can make software-development groups much more productive than was possible with earlier techniques—object-oriented programs are often easier to understand, correct and modify.

1.8 Introduction to Object Technology (Cont.)

- The Automobile as an Object
  - Let's begin with a simple analogy.
  - Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal.*
  - Before you can drive a car, someone has to *design* it.
  - A car typically begins as engineering drawings, similar to the *blueprints* that describe the design of a house.
  - Drawings include the design for an accelerator pedal.
  - Pedal *hides* from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel hides the mechanisms that turn the car.

1.8 Introduction to Object Technology (Cont.)

- Enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.
- Before you can drive a car, it must be *built* from the engineering drawings that describe it.
- A completed car has an *actual* accelerator pedal to make the car go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must *press* the pedal to accelerate the car.

1.8 Introduction to Object Technology (Cont.)

Functions, Member Functions and Classes

- Performing a task in a program requires a member function
- Houses the program statements that actually perform its task.
- Hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster.

1.8 Introduction to Object Technology (Cont.)

- In C++, we create a program unit called a class to house the set of functions that perform the class's tasks—these are the class's member functions.
- A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

1.8 Introduction to Object Technology (Cont.)

Instantiation

- Just as someone has to *build* a car from its engineering drawings before you can actually drive a car, you must *build an object* from a class before a program can perform the tasks that the class's member functions define.
- An object is then referred to as an instance of its class.

1.8 Introduction to Object Technology (Cont.)

Reuse

- Just as a car's engineering drawings can be *reused* many times to build many cars, you can *reuse* a class many times to build many objects.
- Reuse of existing classes when building new classes and programs saves time and effort.

1.8 Introduction to Object Technology (Cont.)

- Reuse also helps you build more reliable and effective systems, because existing classes and components often have gone through extensive *testing, debugging* and *performance* tuning.
- Just as the notion of *interchangeable parts* was crucial to the Industrial Revolution, reusable classes are crucial to the software revolution that has been spurred by object technology.

1.8 Introduction to Object Technology (Cont.)

Messages and Member-Function Calls

- When you drive a car, pressing its gas pedal sends a *message* to the car to perform a task—that is, to go faster.
- Similarly, you *send messages to an object.*
- Each message is implemented as a member-function call that tells a member function of the object to perform its task.

1.8 Introduction to Object Technology (Cont.)

Attributes and Data Members

- A car has *attributes*
- Color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading).
- The car's attributes are represented as part of its design in its engineering diagrams.
- Every car maintains its *own* attributes.
- Each car knows how much gas is in its own gas tank, but *not* how much is in the tanks of other cars.

1.8 Introduction to Object Technology (Cont.)

- An object has attributes that it carries along as it's used in a program.
- Specified as part of the object's class.
- A bank account object has a *balance attribute* that represents the amount of money in the account.
- Each bank account object knows the balance in the account it represents, but *not* the balances of the *other* accounts in the bank.
- Attributes are specified by the class's data members.

1.8 Introduction to Object Technology (Cont.)

Encapsulation

- Classes encapsulate (i.e., wrap) attributes and member functions into objects created from those classes
  - An object's attributes and member functions are intimately related.
- Objects may communicate with one another
  - They're normally not allowed to know how other objects are implemented—implementation details are *hidden*.
- Information hiding is crucial to good software engineering.

1.8 Introduction to Object Technology (Cont.)

Inheritance

- A new class of objects can be created quickly and conveniently by inheritance—the new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own.
- In our car analogy, an object of class "convertible" certainly *is an* object of the more *general* class "automobile," but more *specifically*, the roof can be raised or lowered.

1.8 Introduction to Object Technology (Cont.)

Object-Oriented Analysis and Design (OOAD)

- How will you create the code (i.e., the program instructions) for your programs?
- Follow a detailed analysis process for determining your project's requirements (i.e., defining *what* the system is supposed to do)
- Develop a design that satisfies them (i.e., deciding *how* the system should do it).
- Carefully review the design (and have your design reviewed by other software professionals) before writing any code.

1.8 Introduction to Object Technology (Cont.)

- If this process involves analyzing and designing your system from an object-oriented point of view, it's called an object-oriented analysis and design (OOAD) process.
- Languages like C++ are object oriented.
- Object-oriented programming (OOP) allows you to implement an object-oriented design as a working system.

1.8 Introduction to Object Technology (Cont.)

The UML (Unified Modeling Language)

- The Unified Modeling Language (UML) is now the most widely used graphical scheme for modeling object-oriented systems.

1.9 Typical C++ Development Environment (Cont.)

- C++ systems generally consist of three parts: a program development environment, the language and the C++ Standard Library.
- C++ programs typically go through six phases: edit, preprocess, compile, link, load and execute.

1.9 Typical C++ Development Environment (Cont.)

- Phase 1 consists of editing a file with an *editor* program, normally known simply as an editor.
  - Type a C++ program (source code) using the editor.
  - Make any necessary corrections.
  - Save the program.
  - C++ source code filenames often end with the `.cpp`, `.cxx`, `.cc` or `.C` extensions, which indicate that a file contains C++ source code.

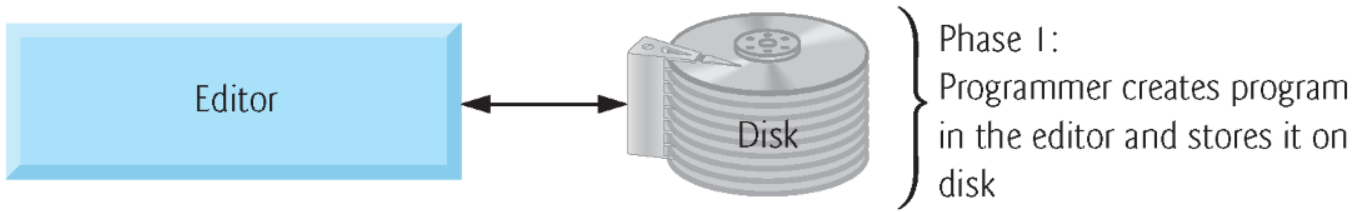


Fig. 1.6 | Typical C++ development environment—editing phase.

1.9 Typical C++ Development Environment (Cont.)

- Linux editors: `vi` and `emacs`.
- You can also use a simple text editor, such as Notepad in Windows, to write your C++ code.
- integrated development environments (IDEs)
  - Provide tools that support the software-development process, including editors for writing and editing programs and debuggers for locating logic errors—errors that cause programs to execute incorrectly.

1.9 Typical C++ Development Environment (Cont.)

- Popular IDEs
  - Microsoft® Visual Studio 2017 Community Edition
  - NetBeans
  - Eclipse
  - Apple's Xcode
  - CodeLite
  - Clion

1.9 Typical C++ Development Environment (Cont.)

- In phase 2, you give the command to compile the program.
  - A **preprocessor** program executes automatically before the compiler's translation phase begins (so we call preprocessing Phase 2 and compiling Phase 3).
  - The C++ preprocessor obeys commands called preprocessing directives, which indicate that certain manipulations are to be performed on the program before compilation.
  - These manipulations usually include (copy into the program file) other text files to be compiled, and perform various text replacements.

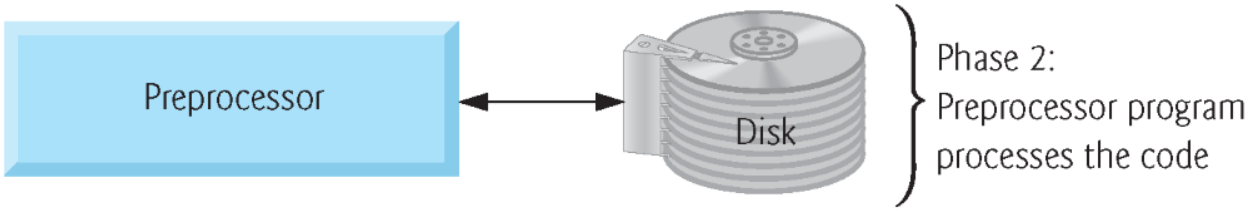


Fig. 1.7 | Typical C++ development environment—preprocessor phase.

1.9 Typical C++ Development Environment (Cont.)

- In Phase 3, the compiler translates the C++ program into machine-language code—also referred to as object code.

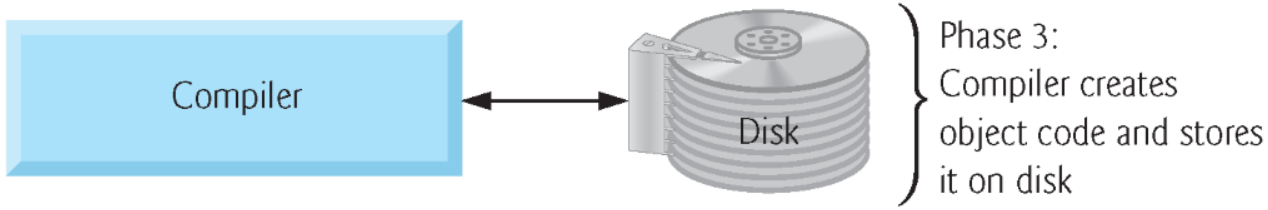


Fig. 1.8 | Typical C++ development environment—compilation phase.

1.9 Typical C++ Development Environment (Cont.)

- Phase 4 is called linking.
  - The object code produced by the C++ compiler typically contains "holes" due to these missing parts.
  - A linker links the object code with the code for the missing functions to produce an executable program.
  - If the program compiles and links correctly, an executable image is produced.

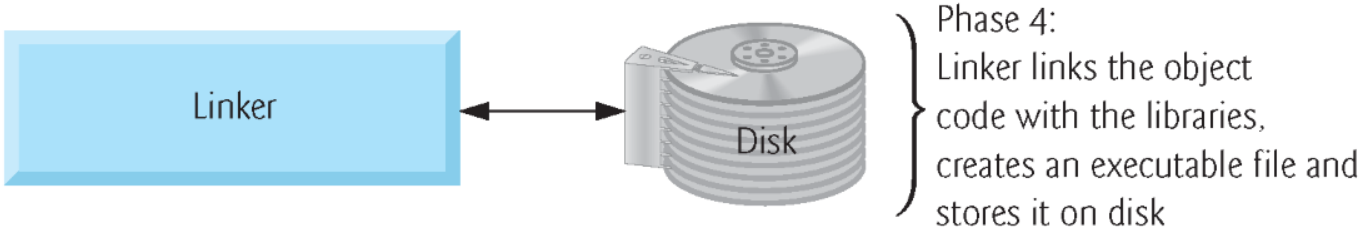


Fig. 1.9 | Typical C++ development environment—linking phase.

1.9 Typical C++ Development Environment (Cont.)

- Phase 5 is called loading.
  - Before a program can be executed, it must first be placed in memory.
  - This is done by the loader, which takes the executable image from disk and transfers it to memory.
  - Additional components from shared libraries that support the program are also loaded.

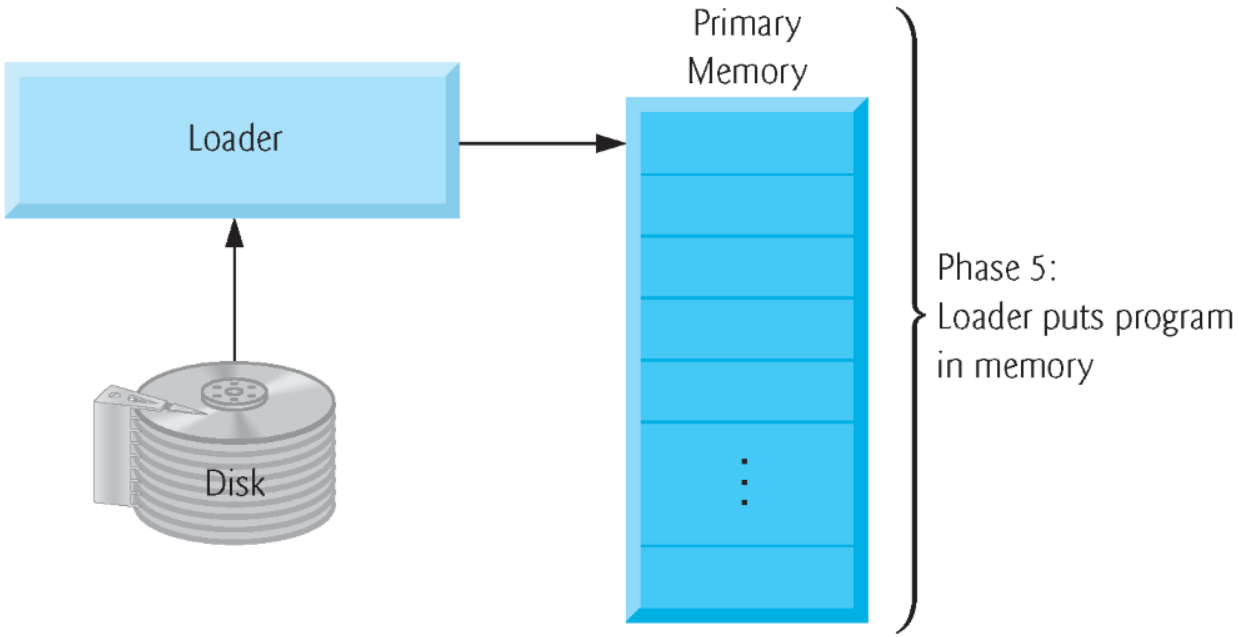


Fig. 1.10 | Typical C++ development environment—loading phase.

1.9 Typical C++ Development Environment (Cont.)

- Phase 6: Execution
  - Finally, the computer, under the control of its CPU, executes the program one instruction at a time.
  - Some modern computer architectures often execute several instructions in parallel.

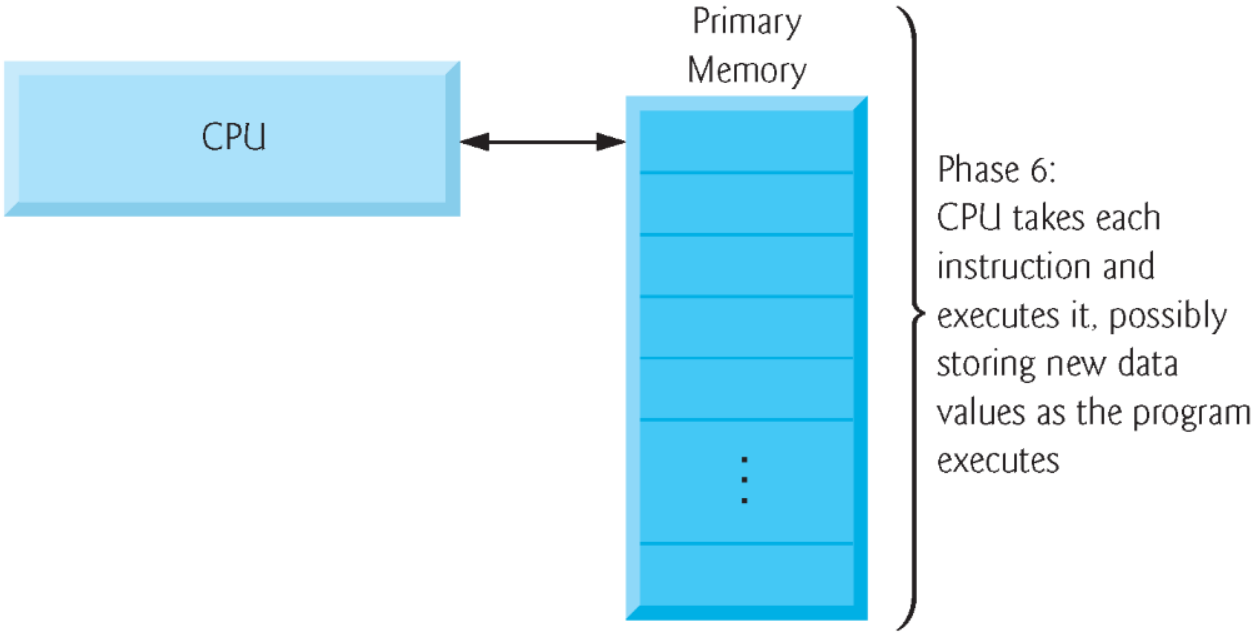


Fig. 1.11 | Typical C++ development environment—execution phase.

1.9 Typical C++ Development Environment (Cont.)

- Problems That May Occur at Execution Time
  - Programs might not work on the first try.
  - Each of the preceding phases can fail because of various errors that we'll discuss throughout this book.
  - If this occurred, you'd have to return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections fixed the problem(s).
  - Most programs in C++ input or output data.

1.9 Typical C++ Development Environment (Cont.)

- Certain C++ functions take their input from `cin` (the standard input stream; pronounced "see-in"), which is normally the keyboard, but `cin` can be redirected to another device.
- Data is often output to `cout` (the standard output stream; pronounced "see-out"), which is normally the computer screen, but `cout` can be redirected to another device.
- When we say that a program prints a result, we normally mean that the result is displayed on a screen.

1.9 Typical C++ Development Environment (Cont.)

- Data may be output to other devices, such as disks, hardcopy printers or even transmitted over the Internet.
- There is also a standard error stream referred to as cerr. The `cerr` stream is used for displaying error messages.

Common Programming Error 1.1

Errors such as division by zero occur as a program runs, so they're called ***runtime errors*** *or* ***execution-time errors****.* ***Fatal runtime errors*** *cause programs to terminate immediately without having successfully performed their jobs.* ***Nonfatal runtime errors*** *allow programs to run to completion, often producing incorrect results.*

1.10 Test-Driving a C++ Application

- In this section, you'll compile, run and interact with your first C++ application.
  - Guess-the-number game, which picks a number from 1 to 1000 and prompts you to guess it.
  - If your guess is correct, the game ends.
  - If your guess is not correct, the application indicates whether your guess is higher or lower than the correct number. There is no limit on the number of guesses you can make.

1.10 Test-Driving a C++ Application

- [Note: For this test drive only, we've modified this application from the exercise you'll be asked to create in Chapter 6, Functions and an Introduction to Recursion. Normally this application randomly selects the correct answer as you execute the program. The modified application uses the same correct answer every time the program executes (though this may vary by compiler), so you can use the same guesses we use in this section and see the same results as we walk you through interacting with your first C++ application.]

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- In this section, you'll run a C++ program on Windows using Microsoft Visual Studio 2015 Community Edition.
- There are several versions of Visual Studio available—on some versions, the options, menus and instructions we present might differ slightly. From this point forward, we'll refer to Visual Studio 2015 Community Edition simply as "Visual Studio" or "the IDE."

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 1: Checking Your Setup
  - It's important to read this book's Before You Begin section to make sure that you've installed Visual Studio and copied the book's examples to your hard drive correctly.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 2: Launching Visual Studio
  - Open Visual Studio from the **Start** menu.
  - The IDE displays the Start Page (Fig. 1.12), which provides links for creating new programs, opening existing programs and learning about the IDE and various programming topics.
  - Close this window for now by clicking the **X** in its tab—you can access this window any time by selecting **View > Start Page**.
    - We use the > character to indicate selecting a menu item from a menu. For example, the notation **File > Open** indicates that you should select the **Open** menu item from the **File** menu.

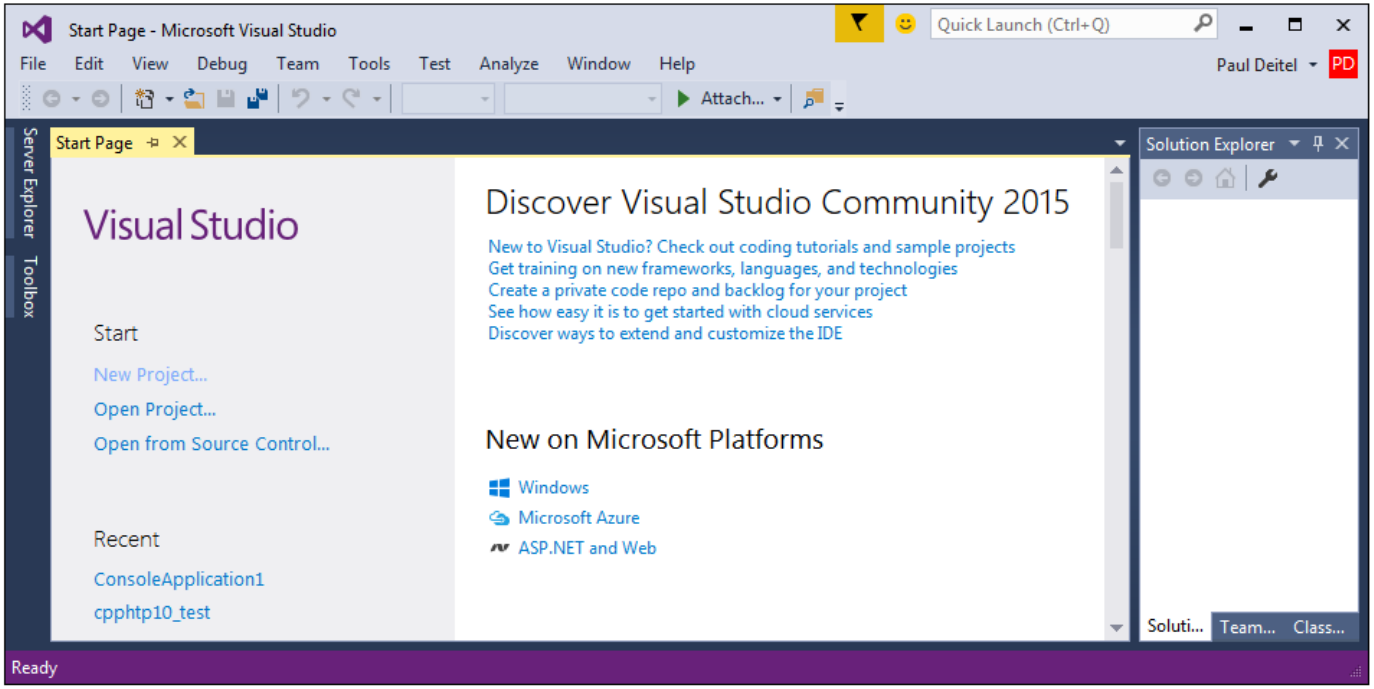


Fig. 1.12 | Visual Studio 2015 Community Edition window showing the **Start Page**.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- 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.
  - Multiple-project solutions are used to create large-scale applications.
  - Each application in this book will be a solution containing a single project.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 3: Creating a Project
  - This book's examples are **Win32 Console Application** projects that you'll execute from the IDE.
  - Select **File > New > Project...**.
  - At the **New Project** dialog's left side, select the category **Installed > Templates > Visual C++ > Win32** (Fig. 1.13).

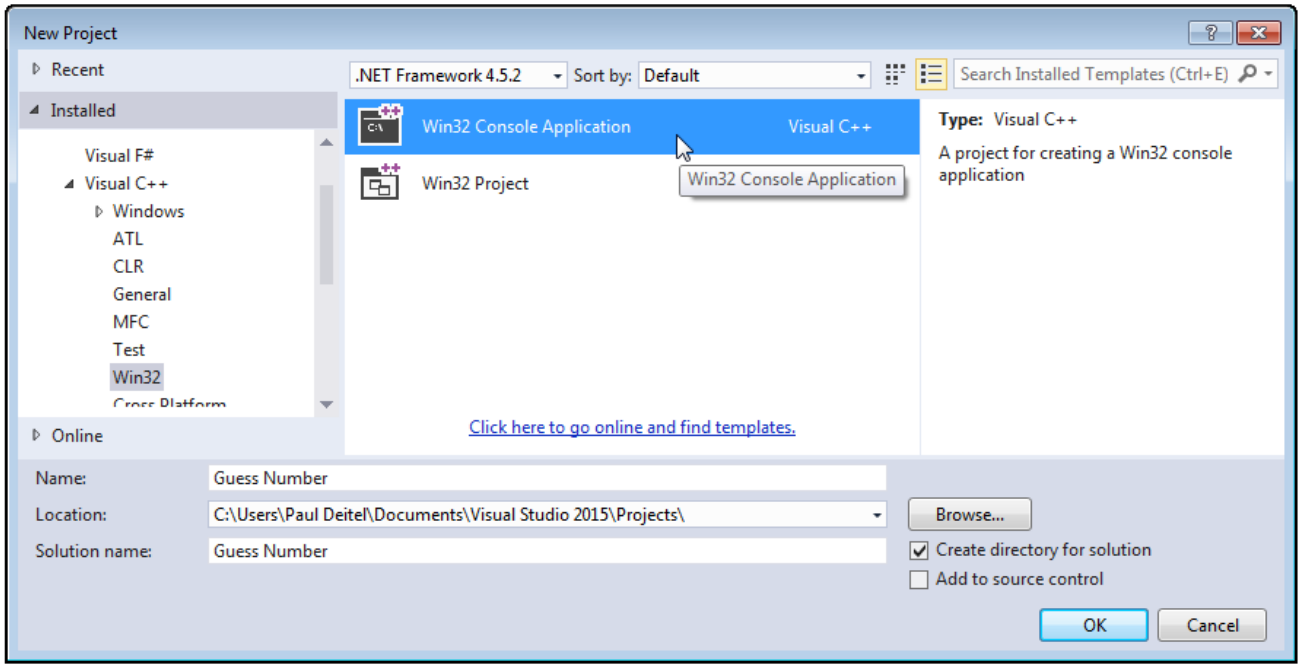


Fig. 1.13 | Visual Studio 2015 Community Edition **New Project** dialog.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 3: Creating a Project
  - In the **New Project** dialog's middle section, select **Win32 Console Application**.
  - Provide a name for your project in the **Name** field—we specified `Guess Number`—then click **OK** to display the **Win32 Application Wizard** window, then click **Next >** to display the **Application Settings** step.
  - Configure the settings as shown in Fig. 1.14 to create a solution containing an empty project, then click **Finish**.

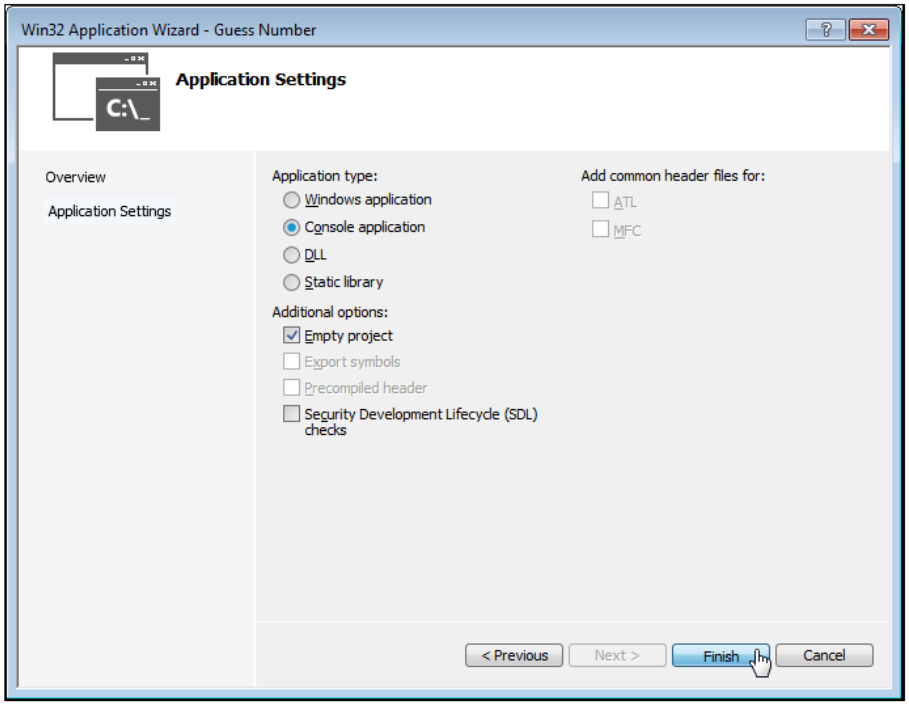


Fig. 1.14 | **Win32 Application Wizard** window's **Application Settings** step.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 3: Creating a Project
  - At this point, the IDE opens the window in Fig. 1.15 and creates your project and places its folder in
    - C:\Users*YourUserAccount*\Documents\Visual Studio 2015\Projects
  - This window displays editors as tabbed windows (one for each file) when you're editing code.
  - Also displayed is the **Solution Explorer** in which you can view and manage your application's files. 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**.

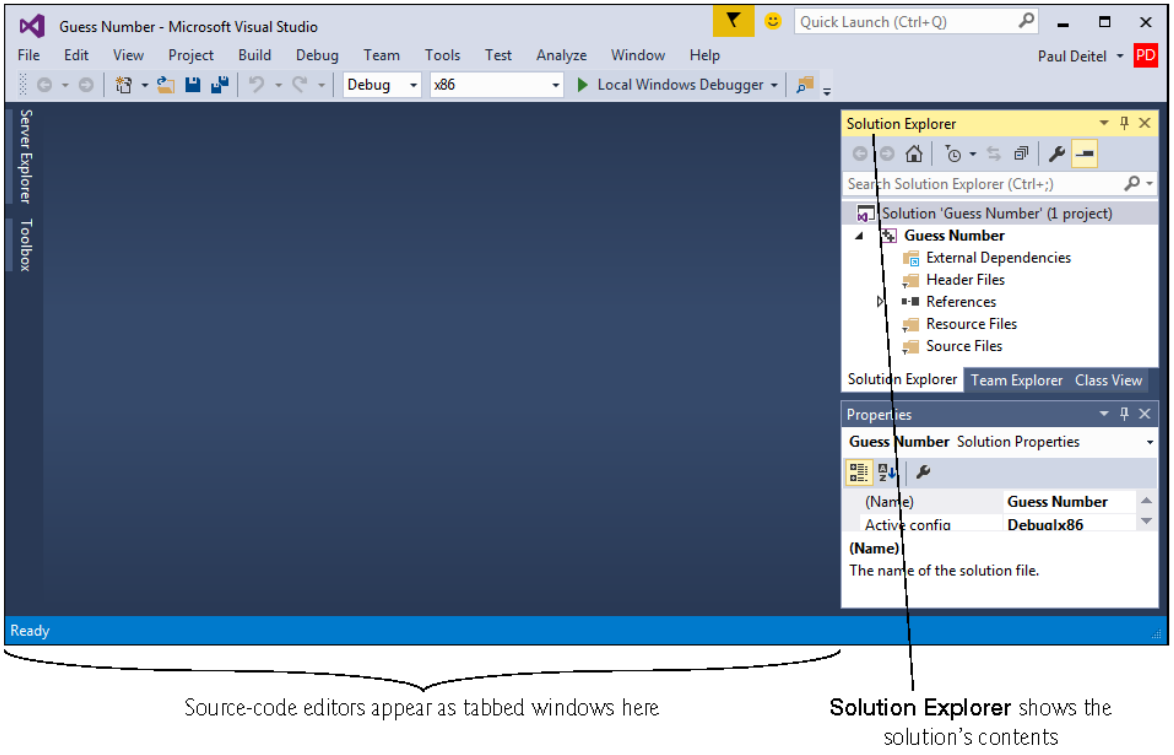


Fig. 1.15 | Visual Studio window after creating the `Guess Number` project.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 4: Adding the GuessNumber.cpp File into the Project
  - Next, add `GuessNumber.cpp` to the project you created in Step 3.
  - In Windows Explorer (Windows 7) or File Explorer (Windows 8 and 10), open the `ch01` folder in the book's `examples` folder, then drag `GuessNumber.cpp` onto the **Source Files** folder in the **Solution Explorer**.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 5: Compiling and Running the Project
  - To compile and run the project so you can test-drive the application, select **Debug > Start** without debugging or simply type *Ctrl* + *F5*.
  - If the program compiles correctly, the IDE opens a **Command Prompt** window and executes the program (Fig. 1.16)
    - We changed the **Command Prompt**'s color scheme to make the screen captures more readable.
  - The application displays "Please type your first guess.", then displays a question mark (?) as a prompt on the next line.

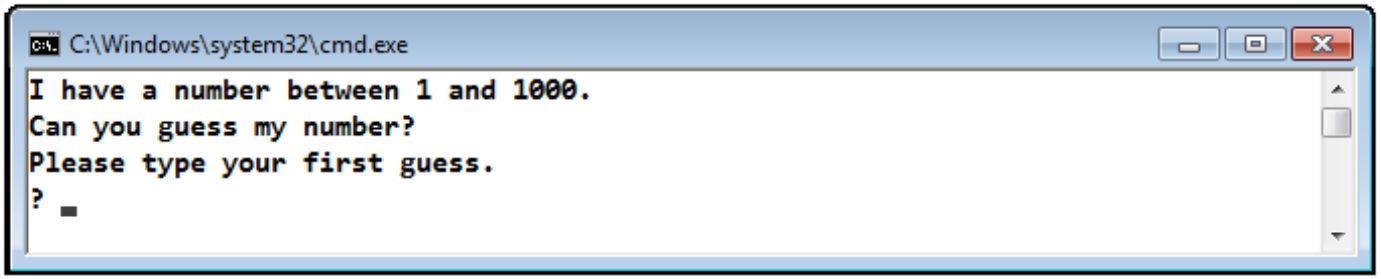


Fig. 1.16 | Command Prompt showing the running program.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 6: Entering Your First Guess
  - Type `500` and press *Enter*.
  - The application displays "Too high. Try again." (Fig. 1.17), meaning that the value you entered is greater than the number the application chose as the correct guess.

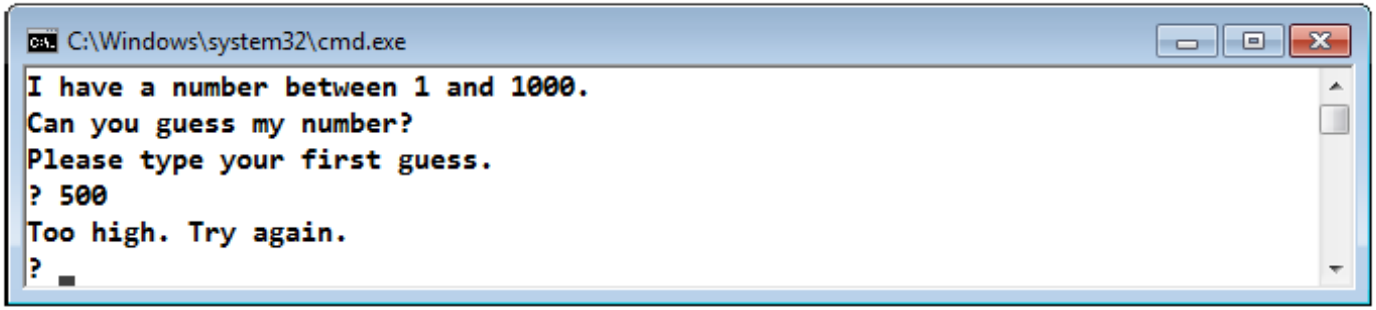


Fig. 1.17 | Entering an initial guess and receiving feedback.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 7: Entering Another Guess
  - At the next prompt, enter `250` (Fig. 1.18).
  - The application displays "Too high. Try again.", because the value you entered once again is greater than the correct guess.

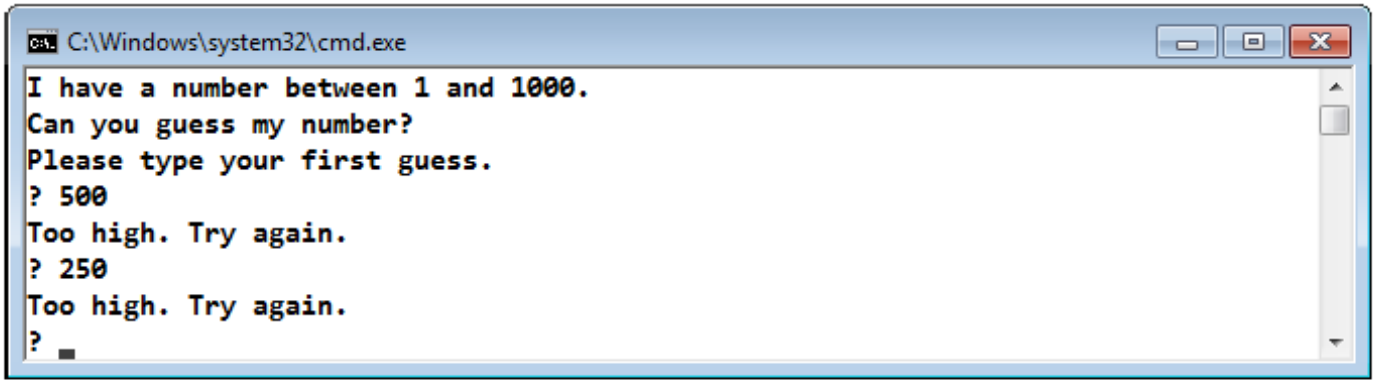


Fig. 1.18 | Entering a second guess and receiving feedback.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 8: Entering Additional Guesses
  - Continue to play the game (Fig. 1.19) by entering values until you guess the correct number.
  - When you guess correctly, the application displays "Excellent! You guessed the number."

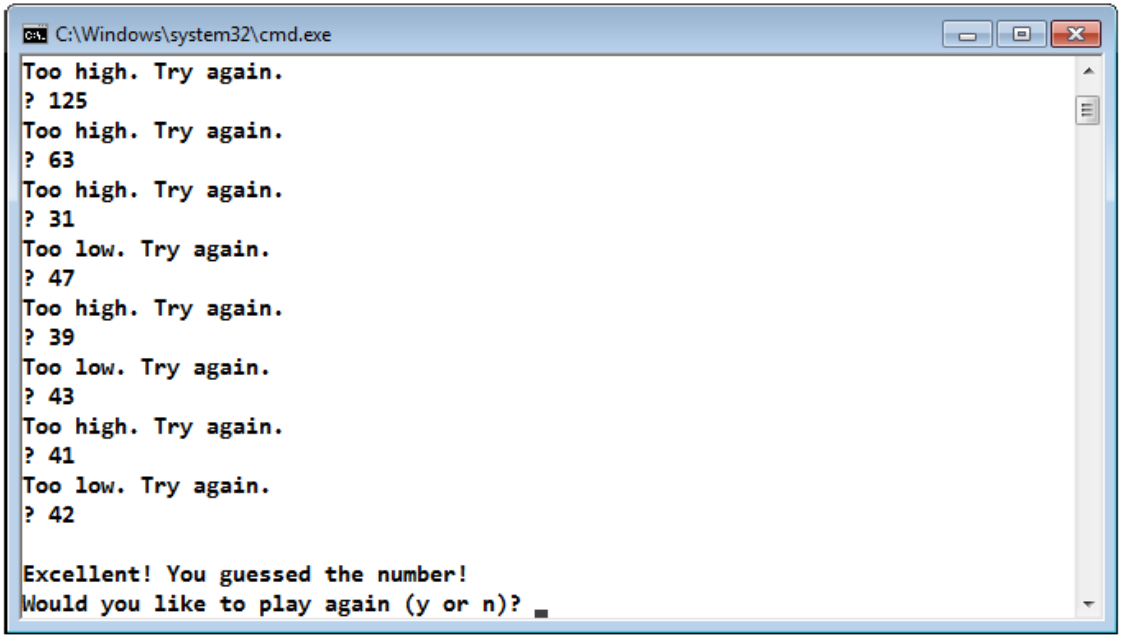


Fig. 1.19 | Entering additional guesses and guessing the correct number.

1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows

- Step 9: 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 "Would you like to play again (y or n)?" prompt, entering the one character y causes the application to choose a new number and displays the message "Please type your first guess." followed by a question-mark prompt so you can make your first guess in the new game.
  - Entering the character n terminates the application.
  - Each time you execute this application from the beginning (Step 5), it will choose the same numbers for you to guess.

1.10.2 Compiling and Running Using GNU C++ 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 home directory on your Linux system.
- The prompt in the shell on our system uses the tilde (~) character to represent the home directory, and each prompt ends with the dollar sign ($) character. The prompt will vary among Linux systems.

1.10.2 Compiling and Running Using GNU C++ on Linux

- Step 1: Locating the Completed Application
  - From a Linux shell, use the command `cd` to change to the completed application directory (Fig. 1.20) by typing
    - `cd examples/ch01`
  - then pressing *Enter*.

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

Fig. 1.20 | Changing to the `GuessNumber` application's directory.

1.10.2 Compiling and Running Using GNU C++ on Linux

- Step 2: Compiling the Application
  - Before running the application, you must first compile it (Fig. 1.21) by typing
    - `g++ -std=c++14 GuessNumber.cpp -o GuessNumber`
  - This command compiles the application for C++14 (the current C++ version) and produces an executable file called `GuessNumber`.

```
~/examples/ch01$ g++ -std=c++14 GuessNumber.cpp -o GuessNumber
~/examples/ch01$
```

Fig. 1.21 | Compiling the **GuessNumber** application using the g++ command.

1.10.2 Compiling and Running Using GNU C++ on Linux

- Step 3: Running the Application
  - To run the executable file `GuessNumber`, type `./GuessNumber` at the next prompt, then press *Enter* (Fig. 1.22).
  - The `./` tells Linux to run from the current directory and is required to indicate that `GuessNumber` is an executable file.

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

Fig. 1.22 | Running the **GuessNumber** application.

1.10.2 Compiling and Running Using GNU C++ on Linux

- 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 (Fig. 1.22).
  - At the prompt, enter `500` (Fig. 1.23). [Note that the outputs may vary based on the compiler you're using.]

```
~/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.
?
```

Fig. 1.23 | Entering an initial guess.

1.10.2 Compiling and Running Using GNU C++ on Linux

- Step 5: Entering Another Guess
  - The application displays "Too high. Try again.", meaning that the value you entered is greater than the number the application chose as the correct guess (Fig. 1.23).
  - At the next prompt, enter `250` (Fig. 1.24).
  - This time the application displays "Too low. Try again.", because the value you entered is less than the correct guess.

```
~/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.
?
```

Fig. 1.24 | Entering a second guess and receiving feedback.

1.10.2 Compiling and Running Using GNU C++ on Linux

- Step 6: Entering Additional Guesses
  - Continue to play the game (Fig. 1.25) 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 (y or n)?
```

Fig. 1.25 | Entering additional guesses and guessing the correct number.

1.10.2 Compiling and Running Using GNU C++ on Linux

- 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 "Would you like to play again (y or n)?" prompt, entering the one character y causes the application to choose a new number and displays the message "Please type your first guess." followed by a question-mark prompt so you can make your first guess in the new game.
  - Entering the character n ends the application, returns you to the shell and awaits your next command.

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 1: Checking Your Setup
  - It's important to read this book's Before You Begin section to make sure that you've installed Apple's Xcode IDE and copied the book's examples to your hard drive correctly.

1.10.3 Compiling and Running with Xcode on Mac OS X

- 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 will appear (Fig. 1.26).
  - Close this window for now—you can access it any time by selecting **Window > Welcome to Xcode**.

Fig. 1.26 | **Welcome to Xcode** window.

1.10.3 Compiling and Running with Xcode on Mac OS X

- 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 **OS X Command Line Tool** projects that you'll execute directly in the IDE. To create a project:

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 3: Creating a Project
  - Select **File > New > Project...**.
  - In the **OS X** subcategory **Application**, select **Command Line Tool** and click **Next**.
  - Provide a name for your project in the **Product Name** field—we specified `Guess Number`.
  - Ensure that the selected **Language** is **C++** and click **Next**.
  - Specify where you want to store your project, then click **Create**. (See the Before You Begin section for information on configuring a project to use C++14.)

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 3: Creating a Project
  - Figure 1.27 shows the workspace window that appears after you create the project.
  - By default, Xcode creates a `main.cpp` source-code file containing a simple program that displays "Hello, World!".
  - The window is divided into four main areas below the toolbar: the **Navigator** area, **Editor** area and **Utilities** area are displayed initially. We'll explain momentarily how to display the **Debug** area in which you'll run and interact with the program.

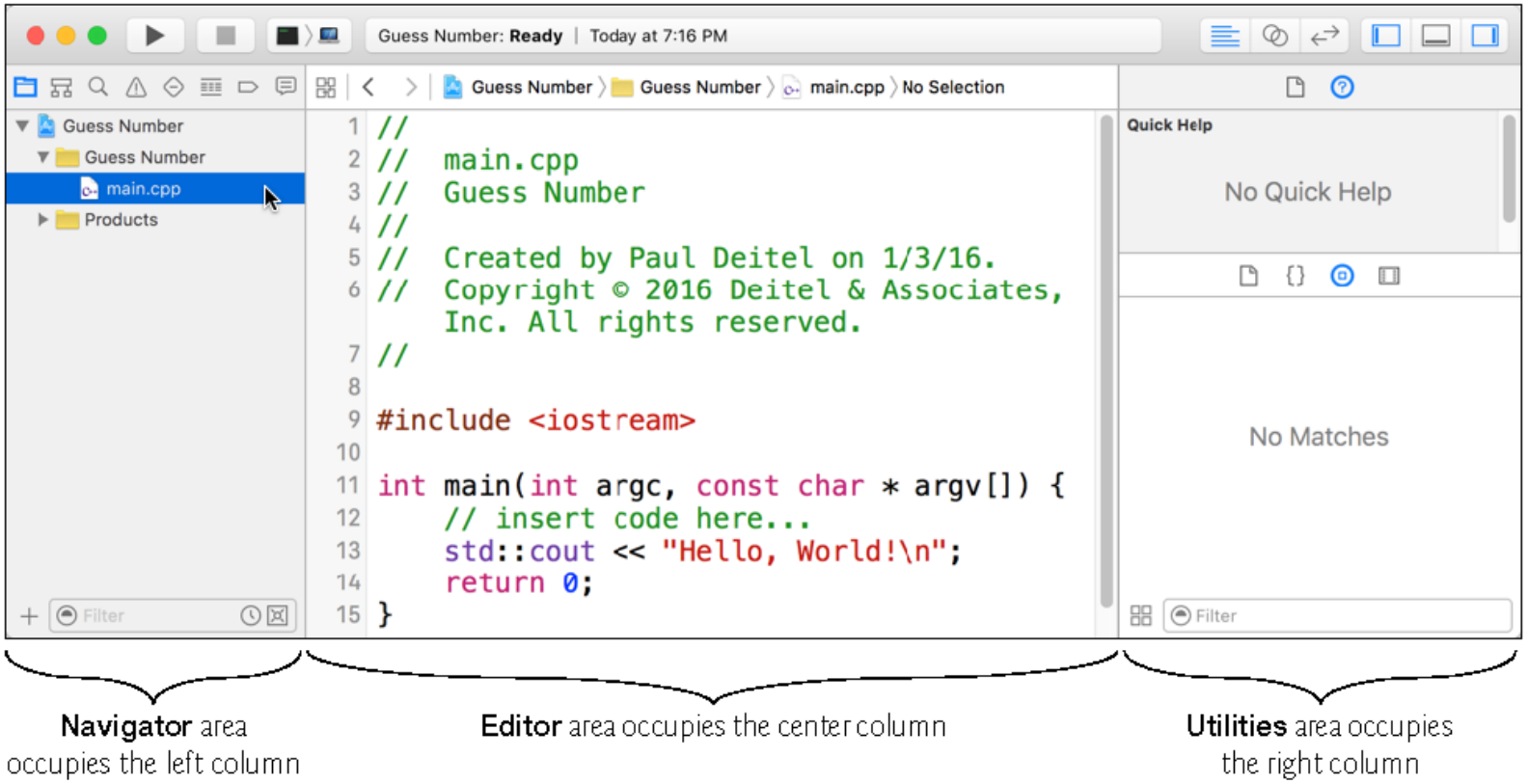


Fig. 1.27 | Sample Xcode C++ project with `main.cpp` selected.

1.10.3 Compiling and Running with Xcode on Mac OS X

- At the left of the workspace window is the **Navigator** area.
- The **Project** navigator shows all the files and folders in your project.
- The Issue navigator shows you warnings and errors generated by the compiler.
- You choose which navigator to display by clicking the corresponding button above the **Navigator** area of the window.

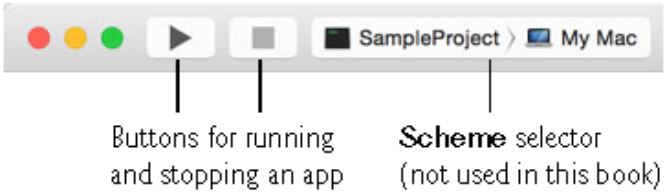
1.10.3 Compiling and Running with Xcode on Mac OS X

- To the right of the **Navigator** area is the **Editor** area for editing source code.
  - When you select a file in the **Project** navigator, the file's contents are displayed in the **Editor** area.
- At the right side of the workspace window is the **Utilities** area, which you will not use in this book.
- The **Debug** area, when displayed, appears below the **Editor** area.

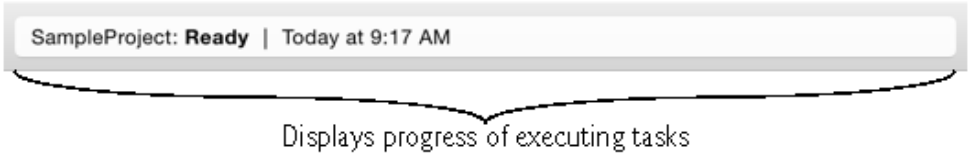
1.10.3 Compiling and Running with Xcode on Mac OS X

- The toolbar contains options for executing a program (Fig. 1.28(a)), a display area (Fig. 1.28(b)) to shows the progress of tasks executing in Xcode (such as the compilation status) and buttons (Fig. 1.28(c)) for hiding and showing areas in the workspace window.

a) The left side of the toolbar



b) The middle of the toolbar



c) The right side of the toolbar

Fig. 1.28 | Xcode 7 toolbar.

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 4: Deleting the main.cpp File from the Project
  - You won't use `main.cpp` in this test-drive, so you should delete the file.
  - In the **Project** navigator, right click the `main.cpp` file and select **Delete**.
  - In the dialog that appears, select **Move to Trash** to delete the file.

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 5: Adding the GuessNumber.cpp File into the Project
  - Add `GuessNumber.cpp` to the project.
  - In a **Finder** window, open the `ch01` folder in the book's `examples` folder, then drag `GuessNumber.cpp` onto the **Guess Number** folder in the **Project** navigator.
  - In the dialog that appears, ensure that **Copy items if needed** is checked, then click **Finish**.

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 6: Compiling and Running the Project
  - To compile and run the project so you can test-drive the application, click the run button at the left side of Xcode's toolbar.
  - If the program compiles correctly, Xcode opens the **Debug** area (at the bottom of the **Editor** area) and executes the program in the right half of the **Debug** area (Fig. 1.29).
  - The application displays "Please type your first guess.", then displays a question mark (?) as a prompt on the next line.

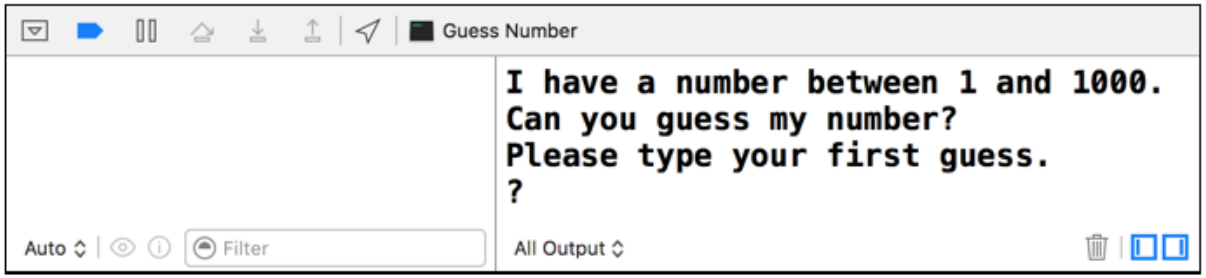


Fig. 1.29 | Debug area showing the running program.

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 7: Entering Your First Guess
  - Click in the **Debug** area, then type `500` and press *Return*.
  - The application displays "Too low. Try again." (Fig. 1.30), meaning that the value you entered is less than the number the application chose as the correct guess.

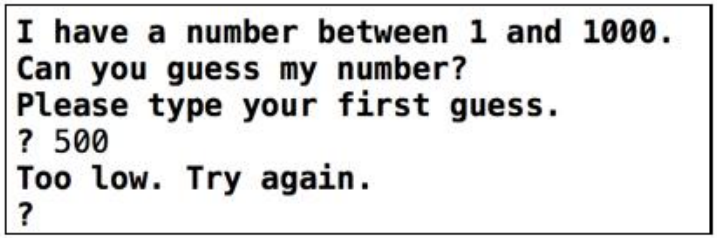


Fig. 1.30 | Entering an initial guess and receiving feedback.

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 8: Entering Another Guess
  - At the next prompt, enter `750` (Fig. 1.31).
  - The application displays "Too low. Try again.", because the value you entered once again is less than the correct guess.

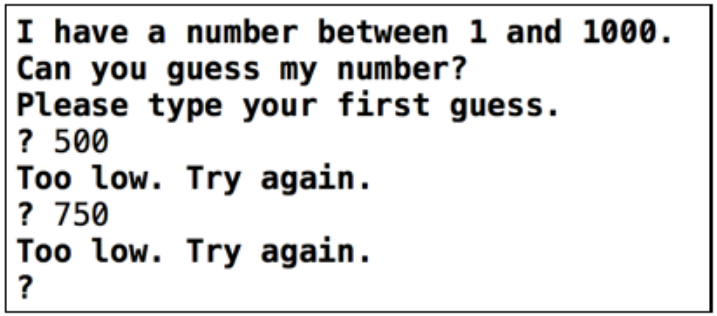


Fig. 1.31 | Entering a second guess and receiving feedback.

1.10.3 Compiling and Running with Xcode on Mac OS X

- Step 9: Entering Additional Guesses
  - Continue to play the game (Fig. 1.32) 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.
? 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 (y or n)?
```

Fig. 1.32 | Entering additional guesses and guessing the correct number.

1.10.3 Compiling and Running with Xcode on Mac OS X

- 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 "Would you like to play again (y or n)?" prompt, entering the character y causes the application to choose a new number and displays the message "Please type your first guess." followed by a question-mark prompt so you can make your first guess in the new game.
  - Entering the character n terminates the application. Each time you execute this application from the beginning (Step 6), it will choose the same numbers for you to guess.

1.11 Operating Systems

- Operating systems
  - Software systems that make using computers more convenient for users, application developers and system administrators.
  - Provide services that allow each application to execute safely, efficiently and *concurrently* (i.e., in parallel) with other applications.
  - The software that contains the core components of the operating system is called the kernel.
  - Popular desktop operating systems include Linux, Windows and OS X (formerly called Mac OS X).
  - Popular mobile operating systems used in smartphones and tablets include Google's Android, Apple's iOS and Windows 10 Mobile.

1.11.1 Windows—A Proprietary Operating System

- Mid-1980s, Microsoft developed the Windows operating system, consisting of a graphical user interface built on top of DOS—an enormously popular personal-computer operating system of the time that users interacted with by *typing* commands.
- Windows borrowed from many concepts (such as icons, menus and windows) developed by Xerox PARC and popularized by early Apple Macintosh operating systems.

1.11.1 Windows—A Proprietary Operating System (Cont.)

- Windows 10 is Microsoft's latest operating system—its features include enhancements to the Start menu and user interface, Cortana personal assistant for voice interactions, Action Center for receiving notifications, Microsoft's new Edge web browser, and more.
- Windows is a proprietary operating system—it's controlled by Microsoft exclusively.
- Windows is by far the world's most widely used desktop operating system

1.11.2 Linux—An Open-Source Operating System

- Open-source software
  - A software development style that departs from the *proprietary* development that dominated software's early years.
  - Individuals and companies *contribute* their efforts in developing, maintaining and evolving software in exchange for the right to use that software for their own purposes, typically at *no charge*.

1.11.2 Linux—An Open-Source Operating System (Cont.)

- Some key organizations in the open-source community are
  - the Eclipse Foundation (the Eclipse Integrated Development Environment helps programmers conveniently develop software)
  - the Mozilla Foundation (creators of the Firefox web browser)
  - the Apache Software Foundation (creators of the Apache web server used to develop web-based applications)
  - GitHub (which provides tools for managing open-source projects—it has millions of them under development).
- Facebook, which was launched from a college dorm room and built with open-source software.

1.11.2 Linux—An Open-Source Operating System (Cont.)

- Linux
  - The most popular open-source operating system.
  - Developed by volunteers
  - Popular in servers, personal computers and embedded systems.
  - Source code is available to the public for examination and modification
  - Free to download and install.
  - Ability to completely customize the operating system to meet specific needs.

1.11.3 Apple's OS X; Apple's iOS for iPhone ®, iPad® and iPod Touch ® Devices

- In 1979, Steve 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 with much fanfare in a memorable Super Bowl ad in 1984.

1.11.3 Apple's OS X; Apple's iOS for iPhone ®, iPad® and iPod Touch ® Devices (Cont.)

- Objective-C Programming Language
  - Created by Brad Cox and Tom Love at Stepstone in the early 1980s, added capabilities for object-oriented programming (OOP) to the C programming language.
  - Steve Jobs left Apple in 1985 and founded NeXT Inc.
  - In 1988, NeXT licensed Objective-C from StepStone and developed an Objective-C compiler and libraries which were used as the platform for the NeXTSTEP operating system's user interface and Interface Builder—used to construct graphical user interfaces.

1.11.3 Apple's OS X; Apple's iOS for iPhone ®, iPad® and iPod Touch ® Devices (Cont.)

- Jobs returned to Apple in 1996 when Apple bought NeXT.
- Apple's OS X operating system is a descendant of NeXTSTEP.
- Apple's proprietary operating system, iOS, is derived from Apple's OS X and is used in the iPhone, iPad and iPod Touch devices.
- In 2014, Apple introduced its new Swift programming language, which became open source in 2015.
- The iOS app-development community is gradually shifting from Objective-C to Swift.

1.11.4 Google's Android

- Android
  - Fastest growing mobile and smartphone operating system
  - Based on the Linux kernel and Java.
  - Open source and free.
  - Developed by Android, Inc., which was acquired by Google in 2005.
  - According to IDC, after the first six months of 2015, Android had 82.8% of the global smartphone market share, compared to 13.9% for Apple, 2.6% for Microsoft and 0.3% for Blackberry.
  - Used in numerous smartphones, e-reader devices, tablets, in-store touch-screen kiosks, cars, robots, multimedia players and more.
  - At the time of this writing there were more than 1.4 billion Android users.

1.12 The Internet and the World Wide Web

- 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 institutions.
- 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 were connecting over telephone lines to computers at a rate of 110 bits per second.
- 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 speeds on the horizon!

1.12 The Internet and the World Wide Web

- 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 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 routed from sender to receiver, arrived intact and were assembled in the correct order.

1.12 The Internet and the World Wide Web

- The Internet: A Network of Networks
  - In parallel with the early evolution of the Internet, organizations worldwide were implementing their own networks for both intraorganization and interorganization 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 "network of networks," the current architecture of the Internet.
  - The combined set of protocols is now called TCP/IP.
  - As a result of business investement, bandwidth—the information-carrying capacity of communications lines—on the Internet has increased tremendously, while hardware costs have plummeted.

1.12 The Internet and the World Wide Web

- 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 multimedia-based documents (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 to develop a technology for sharing information via “hyperlinked” text documents.

1.12 The Internet and the World Wide Web

- Berners-Lee called his invention the HyperText Markup Language (HTML).
- He also wrote communication protocols such as HyperText Transfer Protocol (HTTP) to form the backbone of his new hypertext information system—the World Wide Web.
- In 1994, Berners-Lee founded the World Wide Web Consortium (W3C, http://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.

1.12 The Internet and the World Wide Web

- Web services are software components stored on one computer that can be accessed by an app (or other software component) on another computer over the Internet.
- With web services, you can create mashups, which enable you to rapidly develop apps by combining complementary web services, often from multiple organizations and possibly other forms of information feeds.
- Programmableweb (http://www.programmableweb.com/) provides a directory of over 11,150 APIs and 7,300 mashups, plus how-to guides and sample code for creating your own mashups.

1.12 The Internet and the World Wide Web

- Ajax technology helps Internet-based applications perform like desktop applications.
- Using Ajax, applications like Google Maps have achieved excellent performance and approach the look-and-feel of desktop applications.

1.12 The Internet and the World Wide Web

- The Internet is no longer just a network of computers—it's an Internet of Things.
- A thing is any object with an IP address and the ability to send data automatically over the Internet
- Examples
  - a car with a transponder for paying tolls
  - a heart monitor implanted in a human
  - a smart meter that reports energy usage
  - mobile apps that can track your movement and location
  - smart thermostats that adjust room temperatures based on weather forecasts and activity in the home.

1.13 Some Key Software Development Terminology

- Figure 1.27 lists a number of buzzwords that you'll hear in the software development community.

Technology	Description
Agile software development	**Agile software development** is a set of methodologies that try to get software implemented faster and using fewer resources. Check out the Agile Alliance (`www.agilealliance.org`) and the Agile Manifesto (`www.agilemanifesto.org`).
Refactoring	**Refactoring** involves reworking programs to make them clearer and easier to maintain while preserving their correctness and functionality. It's widely employed with agile development methodologies. Many IDEs contain built-in *refactoring tools* to do major portions of the reworking automatically.
Design patterns	**Design patterns** are 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.

Fig. 1.33 | Software technologies. (Part 1 of 4.)

Technology	Description
LAMP	**LAMP** is an acronym for the open-source technologies that many developers use to build web applications inexpensively—it stands for *Linux*, *Apache*, *MySQL* and *PHP* (or *Perl* or *Python*—two other popular scripting languages). MySQL is an open-source database-management system. PHP is a popular open-source server-side "scripting" language for developing web applications. Apache is the most popular web server software. The equivalent for Windows development is WAMP—*Windows*, *Apache*, *MySQL* and *PHP*.

Fig. 1.33 | Software technologies. (Part 2 of 4.)

Technology	Description
Software as a Service (SaaS)	Software has generally been viewed as a product; most software still is offered this way. If you want to run an application, you buy a software package from a software vendor—often a CD, DVD or web download. You then install that software on your computer and run it as needed. As new versions appear, you upgrade your software, often at considerable cost in time and money. This process can become cumbersome for organizations that must maintain tens of thousands of systems on a diverse array of computer equipment. With **Software as a Service (SaaS)**, the software runs on servers elsewhere on the Internet. When that server is updated, all clients worldwide see the new capabilities—no local installation is needed. You access the service through a browser. Browsers are quite portable, so you can run the same applications on a wide variety of computers from anywhere in the world. Salesforce.com, Google, Microsoft and many other companies offer SaaS.
Platform as a Service (PaaS)	**Platform as a Service (PaaS)** provides a computing platform for developing and running applications as a service over the web, rather than installing the tools on your computer. Some PaaS providers are Google App Engine, Amazon EC2 and Windows Azure™.

Fig. 1.33 | Software technologies. (Part 3 of 4.)

Technology	Description
Cloud computing	SaaS and PaaS are examples of cloud computing. You can use software and data stored in the "cloud"—i.e., accessed on remote computers (or servers) via the Internet and available on demand—rather than having it stored locally on your desktop, notebook computer or mobile device. This 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).
Software Development Kit (SDK)	**Software Development Kits (SDKs)** include the tools and documentation developers use to program applications.

Fig. 1.33 | Software technologies. (Part 4 of 4.)

1.13 Some Key Software Development Terminology (Cont.)

- Figure 1.28 describes software product-release categories.

Version	Description
Alpha	*Alpha* software is the earliest release of a software product that's still under active development. Alpha versions are often buggy, incomplete and unstable and are released to a relatively small number of developers for testing new features, getting early feedback, etc.
Beta	*Beta* versions are released to a larger number of developers later in the development process after most major bugs have been fixed and new features are nearly complete. Beta software is more stable, but still subject to change.
Release candidates	*Release candidates* are generally *feature complete*, (mostly) bug free and ready for use by the community, which provides a diverse testing environment—the software is used on different systems, with varying constraints and for a variety of purposes.
Final release	Any bugs that appear in the release candidate are corrected, and eventually the final product is released to the general public. Software companies often distribute incremental updates over the Internet.
Continuous beta	Software that's developed using this approach (for example, Google search or Gmail) generally does not have version numbers. It's hosted in the *cloud* (not installed on your computer) and is constantly evolving so that users always have the latest version.

Fig. 1.34 | Software product-release terminology.

1.14 C++11 and C++14: The Latest C++ Versions

- C++11 (formerly called C++0x) was published by ISO/IEC in 2011.
- The main goals were to
  - make C++ easier to learn,
  - improve library building capabilities
  - increase compatibility with the C programming language.
- This version of the standard extended the C++ Standard Library and includes several features and enhancements to improve performance and security.

1.14 C++11 and C++14: The Latest C++ Versions

- The current C++ standard, C++14, was published by ISO/IEC in 2014.
- Added several language features and C++ Standard Library enhancements, and fixed bugs from C++11.
- For a list of C++11 and C++14 features and the compilers that support them, visit
  - http://en.cppreference.com/w/cpp/compiler_support
- The next version of the C++ standard, C++17, is currently under development.
  - https://en.wikipedia.org/wiki/C%2B%2B17

1.15 Boost C++ Libraries

- The Boost C++ Libraries are free, open-source libraries created by members of the C++ community.
- Boost has grown to over 100 libraries, with more being added regularly.

1.15 Boost C++ Libraries

- Regular expressions are used to match specific character patterns in text. They can be used to validate data to ensure that it's in a particular format, to replace parts of one string with another, or to split a string.
- Many common bugs in C and C++ code are related to pointers, a powerful programming capability that C++ absorbed from C.
- Smart pointers help you avoid errors associated with traditional pointers.

1.16 Keeping Up-to-Date with Information Technologies

- Figure 1.35 lists key technical and business publications that will help you stay up-to-date with the latest news, trends and technology.

Publication	URL
AllThingsD	allthingsd.com
Bloomberg BusinessWeek	www.businessweek.com
CNET	news.cnet.com
Communications of the ACM	cacm.acm.org
Computerworld	www.computerworld.com
Engadget	www.engadget.com
eWeek	www.eweek.com
Fast Company	www.fastcompany.com
Fortune	fortune.com
GigaOM	gigaom.com
Hacker News	news.ycombinator.com
IEEE Computer Magazine	www.computer.org/portal/web/computingnow/computer
InfoWorld	www.infoworld.com
Mashable	mashable.com
PCWorld	www.pcworld.com

Fig. 1.35 | Technical and business publications. (Part 1 of 2.)

Publication	URL
SD Times	www.sdtimes.com
Slashdot	slashdot.org
Stack Overflow	stackoverflow.com
Technology Review	technologyreview.com
Techcrunch	techcrunch.com
The Next Web	thenextweb.com
The Verge	www.theverge.com
Wired	www.wired.com

Fig. 1.35 | Technical and business publications. (Part 2 of 2.)