009992: Problem Solving and Lab: C++

Introduction to C++ and Object-Oriented Programming (OOP)

Man is still the most extraordinary computer of all.

—John F. Kennedy

Good design is good business.

—Thomas J. Watson, Founder of IBM

How wonderful it is that nobody need wait a single moment before starting to improve the world.

—Anne Frank

1.1 Introduction

- Introduction to C++
- Concepts of OOP
- Typical C++ Development Environment
- SW technologies and some terminologies

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.
- This text provides an introduction to programming in C++11—the latest version standardized through the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).
- In use today are more than a billion general-purpose computers and billions more cell phones, smartphones and handheld devices (such as tablet computers).

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 (Cont.)

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 (Cont.)

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 (Cont.)

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 (Cont.)

- 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 than compiled code.

1.6 C++

- C++ evolved from C, which was developed by Dennis Ritchie at Bell Laboratories.
- C
  - Available for most computers and is hardware independent.
  - It's possible to write C programs that are portable to most computers.
  - 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 and is referred to as *ANSI/ISO 9899: 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, but more importantly, it provides capabilities for object-oriented programming.

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, 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 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.

1.8 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.)

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 member functions that perform the class's tasks.
- 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 methods 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—an object's attributes and member functions are intimately related.
- Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented—implementation details are *hidden* within the objects themselves.
- 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 (note that `C` is in uppercase) which indicate that a file contains C++ source code.

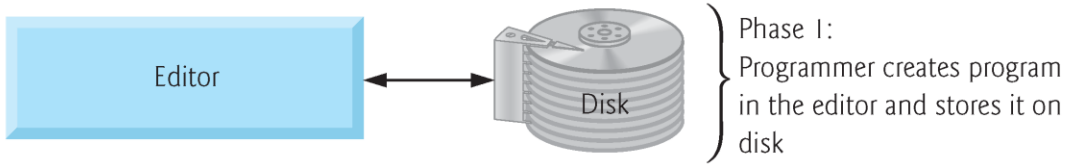

Fig. 1.6 | Typical C++ development environment—editing phase.

1.9 Typical C++ Development Environment (Cont.)

- Linux editors: `vi` and `emacs`.
- C++ software packages for Microsoft Windows such as Microsoft Visual C++ (`microsoft.com/express`) have editors integrated into the programming environment.
- 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 2012 Express Edition
  - Dev C++
  - NetBeans
  - Eclipse
  - Apple's Xcode
  - CodeLite
  - **Code::Blocks**

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 other text files to be compiled, and perform various text replacements.
  - The most common preprocessing directives are discussed in the early chapters; a detailed discussion of preprocessor features appears later.

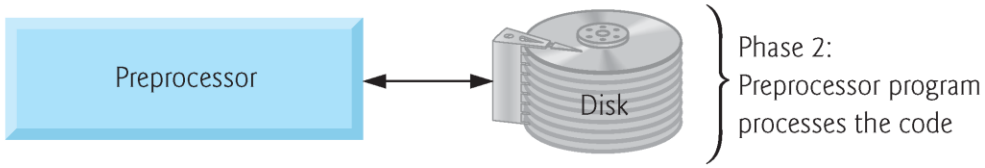

Fig. 1.7 | Typical C++ development environment—preprocessor phase.

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.

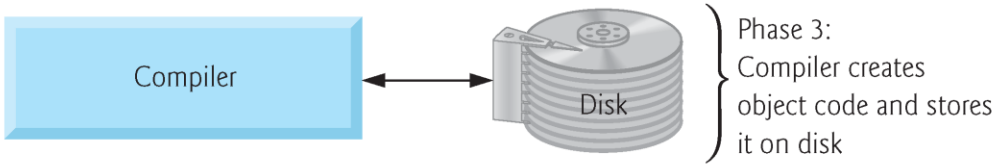

Fig. 1.8 | Typical C++ development environment—compilation phase.

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.

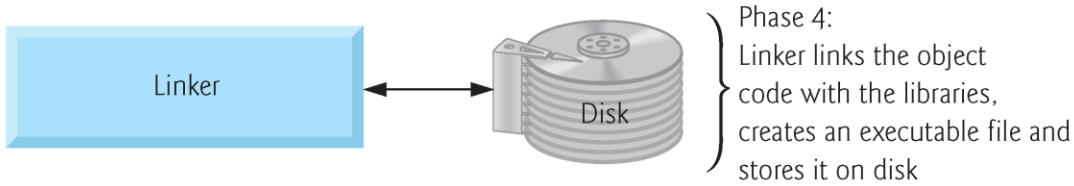

Fig. 1.9 | Typical C++ development environment—linking phase.

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.

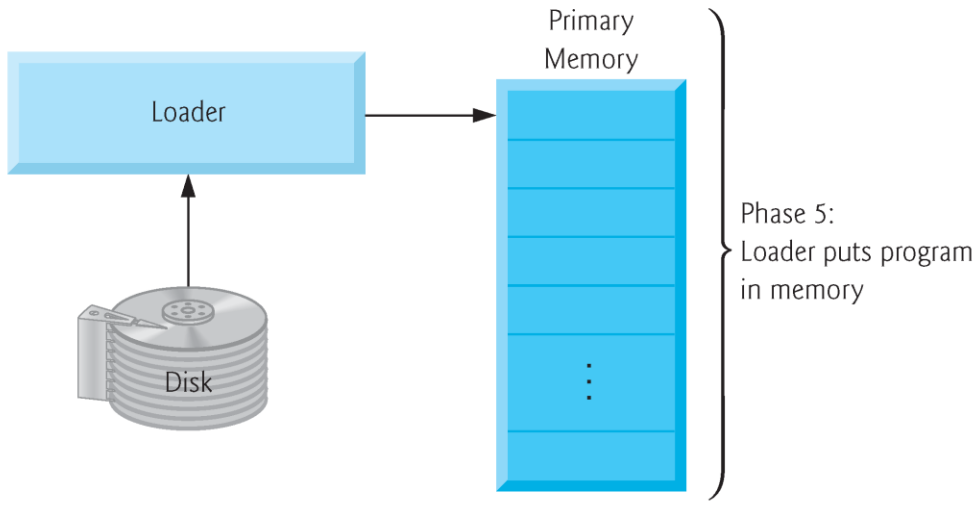

Fig. 1.10 | Typical C++ development environment—loading phase.

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 can execute several instructions in parallel.

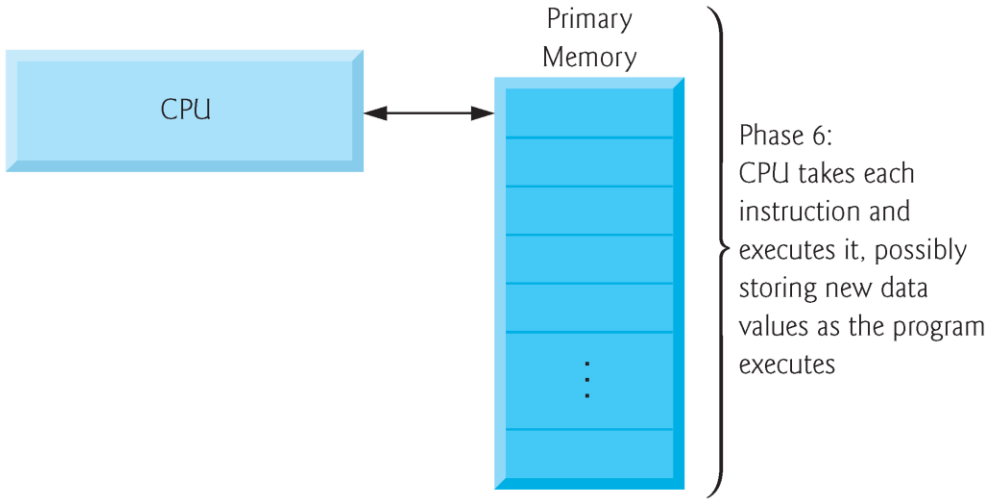

Fig. 1.11 | Typical C++ development environment—execution phase.

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.
  - 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 and hardcopy printers.
- 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.

Technology	Description
Ajax	Ajax is one of the premier Web 2.0 software technologies. Ajax helps Internet-based applications perform like desktop applications—a difficult task, given that such applications suffer transmission delays as data is shuttled back and forth between your computer and servers on the Internet.
Agile software development	Agile software development is a set of methodologies that try to get software implemented faster and using fewer resources than previous methodologies. 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 include *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. 1.27 | Software technologies. (Part 1 of 4.)

Technology	Description
LAMP	LAMP is an acronym for the set of open-source technologies that many developers use to build web applications—it stands for Linux, Apache, MySQL and PHP (or Perl or Python—two other languages used for similar purposes). MySQL is an open-source database management system. PHP is the most popular open-source server-side Internet "scripting" language for developing Internet-based applications.

Fig. 1.27 | 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 of the software appear, you upgrade your software, often requiring significant time and at considerable expense. This process can become cumbersome for organizations with tens of thousands of systems that must be maintained 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, and Microsoft's Office Live and Windows Live all offer SaaS. SaaS is a capability of cloud computing.

Fig. 1.27 | Software technologies. (Part 3 of 4.)

Technology	Description
Platform as a Service (PaaS)	Platform as a Service (PaaS), another capability of cloud computing, provides a computing platform for developing and running applications as a service over the web, rather than installing the tools on your computer. PaaS providers include Google App Engine, Amazon EC2, Bungee Labs and more.
Software Development Kit (SDK)	Software Development Kits (SDKs) include the tools and documentation developers use to program applications.

Fig. 1.27 | 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	An *alpha* version 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* and (supposedly) 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. Any bugs that appear 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 generally does not have version numbers (for example, Google search or Gmail). The software, which is hosted in the cloud (not installed on your computer), is constantly evolving so that users always have the latest version.

Fig. 1.28 | Software product-release terminology.

1.14 C++11 and the Open Source Boost Libraries

- C++11 (formerly called C++0x)—the latest C++ programming language standard—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.
- The new standard extends the C++ Standard Library and includes several features and enhancements to improve performance and security.

1.14 C++11 and the Open Source Boost Libraries (Cont.)

- The major C++ compiler vendors have already implemented many of the new C++11 features (Fig. 1.29).
- For more information, visit the C++ Standards Committee website at

  www.open-std.org/jtc1/sc22/wg21/ and isocpp.org.
- Copies of the C++11 language specification (ISO/IEC 14882:2011) can be purchased at:
  http://bit.ly/CPlusPlus11Standard

1.14 C++11 and the Open Source Boost Libraries (Cont.)

- 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.

1.14 C++11 and the Open Source Boost Libraries (Cont.)

- 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.

C++ Compiler	URL of C++11 feature descriptions
C++11 features implemented in each of the major C++ compilers.	`wiki.apache.org/stdcxx/C%2B%2B0xCompilerSupport`
Microsoft® Visual C++	`msdn.microsoft.com/en-us/library/hh567368.aspx`
GNU Compiler Collection (g++)	`gcc.gnu.org/projects/cxx0x.html`
Intel® C++ Compiler	`software.intel.com/en-us/articles/c0x-features-supported-by-intel-c-compiler/`
IBM® XL C/C++	`www.ibm.com/developerworks/mydeveloperworks/blogs/5894415f-be62-4bc0-81c5-3956e82276f3/entry/xlc_compiler_s_c_11_support50?lang=en`
Clang	`clang.llvm.org/cxx_status.html`
EDG ecpp	`www.edg.com/docs/edg_cpp.pdf`

Fig. 1.29 | C++ compilers that have implemented major portions of C++11.