

GLOBAL
EDITION



C++ How to Program

TENTH EDITION

Paul Deitel • Harvey Deitel

Introducing the New C++14 Standard



Pearson

C++

HOW TO PROGRAM

This page intentionally left blank

C++



HOW TO PROGRAM

Introducing
the New C++14
Standard

TENTH EDITION
GLOBAL EDITION

Paul Deitel
Deitel & Associates, Inc.

Harvey Deitel
Deitel & Associates, Inc.

PEARSON

Boston Columbus Hoboken Indianapolis New York San Francisco
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal
Toronto Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Vice President, Editorial Director: *Marcia Horton*
Acquisitions Editor: *Tracy Johnson*
Editorial Assistant: *Kristy Alaura*
Acquisitions Editor, Global Editions: *Sourabh Maheshwari*
VP of Marketing: *Christy Lesko*
Director of Field Marketing: *Tim Galligan*
Product Marketing Manager: *Bram Van Kempen*
Field Marketing Manager: *Demetrius Hall*
Marketing Assistant: *Jon Bryant*
Director of Product Management: *Erin Gregg*
Team Lead, Program and Project Management: *Scott Disanno*
Program Manager: *Carole Snyder*
Project Manager: *Robert Engelhardt*
Project Editor, Global Editions: *K.K. Neelakantan*
Senior Manufacturing Controller, Global Editions: *Trudy Kimber*
Senior Specialist, Program Planning and Support: *Maura Zaldivar-Garcia*
Media Production Manager, Global Editions: *Vikram Kumar*
Cover Art: *Finevector / Shutterstock*
Cover Design: *Lumina Datamatics*
R&P Manager: *Rachel Youdelman*
R&P Project Manager: *Timothy Nicholls*
Inventory Manager: *Meredith Maresca*

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on page 6.

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsonglobaleditions.com

© Pearson Education Limited 2017

The rights of Paul Deitel and Harvey Deitel to be identified as the authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

Authorized adaptation from the United States edition, entitled C++ How to Program, 10th Edition, ISBN 9780134448237, by Paul Deitel and Harvey Deitel published by Pearson Education © 2017.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a license permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, Saffron House, 6–10 Kirby Street, London EC1N 8TS.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

10 9 8 7 6 5 4 3 2 1

ISBN 10: 1-292-15334-2

ISBN 13: 978-1-292-15334-6

Typeset by GEX Publishing Services

Printed and bound in Malaysia

*In memory of Marvin Minsky,
a founding father of the
field of artificial intelligence.*

*It was a privilege to be your student in two graduate
courses at M.I.T. Every lecture you gave inspired
your students to think beyond limits.*

Harvey Deitel

Trademarks

DEITEL and the double-thumbs-up bug are registered trademarks of Deitel and Associates, Inc.

Carnegie Mellon Software Engineering Institute™ is a trademark of Carnegie Mellon University.

CERT® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

UNIX is a registered trademark of The Open Group.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided “as is” without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. Screen shots and icons reprinted with permission from the Microsoft Corporation. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

Throughout this book, trademarks are used. Rather than put a trademark symbol in every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner, with no intention of infringement of the trademark.



Contents

Chapters 23–26 and Appendices F–J are PDF documents posted online at the book’s Companion Website, which is accessible from

<http://www.pearsonglobaleditions.com/deitel>

See the inside front cover for more information.

Preface **23**

Before You Begin **39**

I Introduction to Computers and C++ **41**

1.1	Introduction	42
1.2	Computers and the Internet in Industry and Research	43
1.3	Hardware and Software	45
1.3.1	Moore’s Law	45
1.3.2	Computer Organization	46
1.4	Data Hierarchy	47
1.5	Machine Languages, Assembly Languages and High-Level Languages	50
1.6	C and C++	51
1.7	Programming Languages	52
1.8	Introduction to Object Technology	54
1.9	Typical C++ Development Environment	57
1.10	Test-Driving a C++ Application	60
1.10.1	Compiling and Running an Application in Visual Studio 2015 for Windows	60
1.10.2	Compiling and Running Using GNU C++ on Linux	65
1.10.3	Compiling and Running with Xcode on Mac OS X	67
1.11	Operating Systems	72
1.11.1	Windows—A Proprietary Operating System	72
1.11.2	Linux—An Open-Source Operating System	72
1.11.3	Apple’s OS X; Apple’s iOS for iPhone®, iPad® and iPod Touch® Devices	73
1.11.4	Google’s Android	73
1.12	The Internet and the World Wide Web	74
1.13	Some Key Software Development Terminology	76
1.14	C++11 and C++14: The Latest C++ Versions	78

1.15	Boost C++ Libraries	79
1.16	Keeping Up to Date with Information Technologies	79

2 Introduction to C++ Programming, Input/Output and Operators 84

2.1	Introduction	85
2.2	First Program in C++: Printing a Line of Text	85
2.3	Modifying Our First C++ Program	89
2.4	Another C++ Program: Adding Integers	90
2.5	Memory Concepts	94
2.6	Arithmetic	95
2.7	Decision Making: Equality and Relational Operators	99
2.8	Wrap-Up	103

3 Introduction to Classes, Objects, Member Functions and Strings 113

3.1	Introduction	114
3.2	Test-Driving an Account Object	115
3.2.1	Instantiating an Object	115
3.2.2	Headers and Source-Code Files	116
3.2.3	Calling Class <code>Account</code> 's <code>getName</code> Member Function	116
3.2.4	Inputting a <code>string</code> with <code>getline</code>	117
3.2.5	Calling Class <code>Account</code> 's <code>setName</code> Member Function	117
3.3	<code>Account</code> Class with a Data Member and <i>Set</i> and <i>Get</i> Member Functions	118
3.3.1	<code>Account</code> Class Definition	118
3.3.2	Keyword <code>class</code> and the Class Body	119
3.3.3	Data Member name of Type <code>string</code>	119
3.3.4	<code>setName</code> Member Function	120
3.3.5	<code>getName</code> Member Function	122
3.3.6	Access Specifiers <code>private</code> and <code>public</code>	122
3.3.7	<code>Account</code> UML Class Diagram	123
3.4	<code>Account</code> Class: Initializing Objects with Constructors	124
3.4.1	Defining an <code>Account</code> Constructor for Custom Object Initialization	125
3.4.2	Initializing <code>Account</code> Objects When They're Created	126
3.4.3	<code>Account</code> UML Class Diagram with a Constructor	128
3.5	Software Engineering with <i>Set</i> and <i>Get</i> Member Functions	128
3.6	<code>Account</code> Class with a Balance; Data Validation	129
3.6.1	Data Member <code>balance</code>	129
3.6.2	Two-Parameter Constructor with Validation	131
3.6.3	<code>deposit</code> Member Function with Validation	131
3.6.4	<code>getBalance</code> Member Function	131
3.6.5	Manipulating <code>Account</code> Objects with Balances	132
3.6.6	<code>Account</code> UML Class Diagram with a Balance and Member Functions <code>deposit</code> and <code>getBalance</code>	134
3.7	Wrap-Up	134

4	Algorithm Development and Control Statements: Part I	143
4.1	Introduction	144
4.2	Algorithms	145
4.3	Pseudocode	145
4.4	Control Structures	146
4.4.1	Sequence Structure	146
4.4.2	Selection Statements	148
4.4.3	Iteration Statements	148
4.4.4	Summary of Control Statements	149
4.5	if Single-Selection Statement	149
4.6	if...else Double-Selection Statement	150
4.6.1	Nested if...else Statements	151
4.6.2	Dangling-else Problem	153
4.6.3	Blocks	153
4.6.4	Conditional Operator (?:)	154
4.7	Student Class: Nested if...else Statements	155
4.8	while Iteration Statement	157
4.9	Formulating Algorithms: Counter-Controlled Iteration	159
4.9.1	Pseudocode Algorithm with Counter-Controlled Iteration	159
4.9.2	Implementing Counter-Controlled Iteration	160
4.9.3	Notes on Integer Division and Truncation	162
4.9.4	Arithmetic Overflow	162
4.9.5	Input Validation	163
4.10	Formulating Algorithms: Sentinel-Controlled Iteration	163
4.10.1	Top-Down, Stepwise Refinement: The Top and First Refinement	164
4.10.2	Proceeding to the Second Refinement	164
4.10.3	Implementing Sentinel-Controlled Iteration	166
4.10.4	Converting Between Fundamental Types Explicitly and Implicitly	169
4.10.5	Formatting Floating-Point Numbers	170
4.10.6	Unsigned Integers and User Input	170
4.11	Formulating Algorithms: Nested Control Statements	171
4.11.1	Problem Statement	171
4.11.2	Top-Down, Stepwise Refinement: Pseudocode Representation of the Top	172
4.11.3	Top-Down, Stepwise Refinement: First Refinement	172
4.11.4	Top-Down, Stepwise Refinement: Second Refinement	172
4.11.5	Complete Second Refinement of the Pseudocode	173
4.11.6	Program That Implements the Pseudocode Algorithm	174
4.11.7	Preventing Narrowing Conversions with List Initialization	175
4.12	Compound Assignment Operators	176
4.13	Increment and Decrement Operators	177
4.14	Fundamental Types Are Not Portable	180
4.15	Wrap-Up	180

5	Control Statements: Part 2; Logical Operators	199
5.1	Introduction	200
5.2	Essentials of Counter-Controlled Iteration	200
5.3	for Iteration Statement	201
5.4	Examples Using the for Statement	205
5.5	Application: Summing Even Integers	206
5.6	Application: Compound-Interest Calculations	207
5.7	Case Study: Integer-Based Monetary Calculations with Class DollarAmount	211
5.7.1	Demonstrating Class DollarAmount	212
5.7.2	Class DollarAmount	215
5.8	do...while Iteration Statement	219
5.9	switch Multiple-Selection Statement	220
5.10	break and continue Statements	226
5.10.1	break Statement	226
5.10.2	continue Statement	227
5.11	Logical Operators	228
5.11.1	Logical AND (&&) Operator	228
5.11.2	Logical OR () Operator	229
5.11.3	Short-Circuit Evaluation	230
5.11.4	Logical Negation (!) Operator	230
5.11.5	Logical Operators Example	231
5.12	Confusing the Equality (==) and Assignment (=) Operators	232
5.13	Structured-Programming Summary	234
5.14	Wrap-Up	239
6	Functions and an Introduction to Recursion	251
6.1	Introduction	252
6.2	Program Components in C++	253
6.3	Math Library Functions	254
6.4	Function Prototypes	255
6.5	Function-Prototype and Argument-Coercion Notes	258
6.5.1	Function Signatures and Function Prototypes	259
6.5.2	Argument Coercion	259
6.5.3	Argument-Promotion Rules and Implicit Conversions	259
6.6	C++ Standard Library Headers	260
6.7	Case Study: Random-Number Generation	262
6.7.1	Rolling a Six-Sided Die	263
6.7.2	Rolling a Six-Sided Die 60,000,000 Times	264
6.7.3	Randomizing the Random-Number Generator with srand	265
6.7.4	Seeding the Random-Number Generator with the Current Time	267
6.7.5	Scaling and Shifting Random Numbers	267
6.8	Case Study: Game of Chance; Introducing Scoped enums	268
6.9	C++11 Random Numbers	272
6.10	Scope Rules	273

6.11	Function-Call Stack and Activation Records	277
6.12	Inline Functions	281
6.13	References and Reference Parameters	282
6.14	Default Arguments	285
6.15	Unary Scope Resolution Operator	287
6.16	Function Overloading	288
6.17	Function Templates	291
6.18	Recursion	294
6.19	Example Using Recursion: Fibonacci Series	297
6.20	Recursion vs. Iteration	300
6.21	Wrap-Up	303

7 Class Templates array and vector; Catching Exceptions 323

7.1	Introduction	324
7.2	arrays	324
7.3	Declaring arrays	326
7.4	Examples Using arrays	326
7.4.1	Declaring an array and Using a Loop to Initialize the array's Elements	327
7.4.2	Initializing an array in a Declaration with an Initializer List	328
7.4.3	Specifying an array's Size with a Constant Variable and Setting array Elements with Calculations	329
7.4.4	Summing the Elements of an array	330
7.4.5	Using a Bar Chart to Display array Data Graphically	331
7.4.6	Using the Elements of an array as Counters	332
7.4.7	Using arrays to Summarize Survey Results	333
7.4.8	Static Local arrays and Automatic Local arrays	336
7.5	Range-Based for Statement	338
7.6	Case Study: Class GradeBook Using an array to Store Grades	340
7.7	Sorting and Searching arrays	346
7.7.1	Sorting	346
7.7.2	Searching	346
7.7.3	Demonstrating Functions sort and binary_search	346
7.8	Multidimensional arrays	347
7.9	Case Study: Class GradeBook Using a Two-Dimensional array	351
7.10	Introduction to C++ Standard Library Class Template vector	357
7.11	Wrap-Up	363

8 Pointers 379

8.1	Introduction	380
8.2	Pointer Variable Declarations and Initialization	381
8.2.1	Declaring Pointers	381
8.2.2	Initializing Pointers	382
8.2.3	Null Pointers Prior to C++11	382

12 Contents

8.3	Pointer Operators	382
8.3.1	Address (&) Operator	382
8.3.2	Indirection (*) Operator	383
8.3.3	Using the Address (&) and Indirection (*) Operators	384
8.4	Pass-by-Reference with Pointers	385
8.5	Built-In Arrays	389
8.5.1	Declaring and Accessing a Built-In Array	389
8.5.2	Initializing Built-In Arrays	390
8.5.3	Passing Built-In Arrays to Functions	390
8.5.4	Declaring Built-In Array Parameters	391
8.5.5	C++11: Standard Library Functions <code>begin</code> and <code>end</code>	391
8.5.6	Built-In Array Limitations	391
8.5.7	Built-In Arrays Sometimes Are Required	392
8.6	Using <code>const</code> with Pointers	392
8.6.1	Nonconstant Pointer to Nonconstant Data	393
8.6.2	Nonconstant Pointer to Constant Data	393
8.6.3	Constant Pointer to Nonconstant Data	394
8.6.4	Constant Pointer to Constant Data	395
8.7	<code>sizeof</code> Operator	396
8.8	Pointer Expressions and Pointer Arithmetic	398
8.8.1	Adding Integers to and Subtracting Integers from Pointers	399
8.8.2	Subtracting Pointers	400
8.8.3	Pointer Assignment	401
8.8.4	Cannot Dereference a <code>void*</code>	401
8.8.5	Comparing Pointers	401
8.9	Relationship Between Pointers and Built-In Arrays	401
8.9.1	Pointer/Offset Notation	402
8.9.2	Pointer/Offset Notation with the Built-In Array's Name as the Pointer	402
8.9.3	Pointer/Subscript Notation	402
8.9.4	Demonstrating the Relationship Between Pointers and Built-In Arrays	403
8.10	Pointer-Based Strings (Optional)	404
8.11	Note About Smart Pointers	407
8.12	Wrap-Up	407

9 Classes: A Deeper Look 425

9.1	Introduction	426
9.2	Time Class Case Study: Separating Interface from Implementation	427
9.2.1	Interface of a Class	428
9.2.2	Separating the Interface from the Implementation	428
9.2.3	Time Class Definition	428
9.2.4	Time Class Member Functions	430
9.2.5	Scope Resolution Operator (<code>::</code>)	431
9.2.6	Including the Class Header in the Source-Code File	431

9.2.7	Time Class Member Function setTime and Throwing Exceptions	432
9.2.8	Time Class Member Function toUniversalString and String Stream Processing	432
9.2.9	Time Class Member Function toStdString	433
9.2.10	Implicitly Inlining Member Functions	433
9.2.11	Member Functions vs. Global Functions	433
9.2.12	Using Class Time	434
9.2.13	Object Size	436
9.3	Compilation and Linking Process	436
9.4	Class Scope and Accessing Class Members	438
9.5	Access Functions and Utility Functions	439
9.6	Time Class Case Study: Constructors with Default Arguments	439
9.6.1	Constructors with Default Arguments	439
9.6.2	Overloaded Constructors and C++11 Delegating Constructors	444
9.7	Destructors	445
9.8	When Constructors and Destructors Are Called	445
9.8.1	Constructors and Destructors for Objects in Global Scope	446
9.8.2	Constructors and Destructors for Non-static Local Objects	446
9.8.3	Constructors and Destructors for static Local Objects	446
9.8.4	Demonstrating When Constructors and Destructors Are Called	446
9.9	Time Class Case Study: A Subtle Trap—Returning a Reference or a Pointer to a private Data Member	449
9.10	Default Memberwise Assignment	451
9.11	const Objects and const Member Functions	453
9.12	Composition: Objects as Members of Classes	455
9.13	friend Functions and friend Classes	461
9.14	Using the this Pointer	463
9.14.1	Implicitly and Explicitly Using the this Pointer to Access an Object's Data Members	464
9.14.2	Using the this Pointer to Enable Cascaded Function Calls	465
9.15	static Class Members	469
9.15.1	Motivating Classwide Data	469
9.15.2	Scope and Initialization of static Data Members	469
9.15.3	Accessing static Data Members	470
9.15.4	Demonstrating static Data Members	470
9.16	Wrap-Up	473

10 Operator Overloading; Class string 487

10.1	Introduction	488
10.2	Using the Overloaded Operators of Standard Library Class string	489
10.3	Fundamentals of Operator Overloading	493
10.3.1	Operator Overloading Is Not Automatic	493
10.3.2	Operators That You Do Not Have to Overload	493
10.3.3	Operators That Cannot Be Overloaded	494
10.3.4	Rules and Restrictions on Operator Overloading	494

14 Contents

10.4	Overloading Binary Operators	495
10.5	Overloading the Binary Stream Insertion and Stream Extraction Operators	495
10.6	Overloading Unary Operators	499
10.7	Overloading the Increment and Decrement Operators	500
10.8	Case Study: A Date Class	501
10.9	Dynamic Memory Management	506
10.10	Case Study: Array Class	508
	10.10.1 Using the Array Class	509
	10.10.2 Array Class Definition	513
10.11	Operators as Member vs. Non-Member Functions	520
10.12	Converting Between Types	521
10.13	<code>explicit</code> Constructors and Conversion Operators	522
10.14	Overloading the Function Call Operator <code>()</code>	525
10.15	Wrap-Up	525

11 Object-Oriented Programming: Inheritance 537

11.1	Introduction	538
11.2	Base Classes and Derived Classes	539
	11.2.1 <code>CommunityMember</code> Class Hierarchy	539
	11.2.2 <code>Shape</code> Class Hierarchy	540
11.3	Relationship between Base and Derived Classes	541
	11.3.1 Creating and Using a <code>CommissionEmployee</code> Class	541
	11.3.2 Creating a <code>BasePlusCommissionEmployee</code> Class Without Using Inheritance	546
	11.3.3 Creating a <code>CommissionEmployee–BasePlusCommissionEmployee</code> Inheritance Hierarchy	551
	11.3.4 <code>CommissionEmployee–BasePlusCommissionEmployee</code> Inheritance Hierarchy Using <code>protected</code> Data	555
	11.3.5 <code>CommissionEmployee–BasePlusCommissionEmployee</code> Inheritance Hierarchy Using <code>private</code> Data	559
11.4	Constructors and Destructors in Derived Classes	563
11.5	<code>public</code> , <code>protected</code> and <code>private</code> Inheritance	565
11.6	Wrap-Up	566

12 Object-Oriented Programming: Polymorphism 571

12.1	Introduction	572
12.2	Introduction to Polymorphism: Polymorphic Video Game	573
12.3	Relationships Among Objects in an Inheritance Hierarchy	574
	12.3.1 Invoking Base-Class Functions from Derived-Class Objects	574
	12.3.2 Aiming Derived-Class Pointers at Base-Class Objects	577
	12.3.3 Derived-Class Member-Function Calls via Base-Class Pointers	578
12.4	Virtual Functions and Virtual Destructors	580
	12.4.1 Why <code>virtual</code> Functions Are Useful	580
	12.4.2 Declaring <code>virtual</code> Functions	580

12.4.3	Invoking a virtual Function Through a Base-Class Pointer or Reference	581
12.4.4	Invoking a virtual Function Through an Object's Name	581
12.4.5	virtual Functions in the CommissionEmployee Hierarchy	581
12.4.6	virtual Destructors	586
12.4.7	C++11: final Member Functions and Classes	586
12.5	Type Fields and switch Statements	587
12.6	Abstract Classes and Pure virtual Functions	587
12.6.1	Pure virtual Functions	588
12.6.2	Device Drivers: Polymorphism in Operating Systems	589
12.7	Case Study: Payroll System Using Polymorphism	589
12.7.1	Creating Abstract Base Class Employee	590
12.7.2	Creating Concrete Derived Class SalariedEmployee	593
12.7.3	Creating Concrete Derived Class CommissionEmployee	596
12.7.4	Creating Indirect Concrete Derived Class BasePlusCommissionEmployee	598
12.7.5	Demonstrating Polymorphic Processing	600
12.8	(Optional) Polymorphism, Virtual Functions and Dynamic Binding "Under the Hood"	603
12.9	Case Study: Payroll System Using Polymorphism and Runtime Type Information with Downcasting, dynamic_cast, typeid and type_info	607
12.10	Wrap-Up	610

13 Stream Input/Output: A Deeper Look 617

13.1	Introduction	618
13.2	Streams	619
13.2.1	Classic Streams vs. Standard Streams	619
13.2.2	iostream Library Headers	620
13.2.3	Stream Input/Output Classes and Objects	620
13.3	Stream Output	621
13.3.1	Output of char* Variables	621
13.3.2	Character Output Using Member Function put	622
13.4	Stream Input	622
13.4.1	get and getline Member Functions	623
13.4.2	istream Member Functions peek, putback and ignore	626
13.4.3	Type-Safe I/O	626
13.5	Unformatted I/O Using read, write and gcount	626
13.6	Stream Manipulators: A Deeper Look	627
13.6.1	Integral Stream Base: dec, oct, hex and setbase	628
13.6.2	Floating-Point Precision (precision, setprecision)	628
13.6.3	Field Width (width, setw)	630
13.6.4	User-Defined Output Stream Manipulators	631
13.7	Stream Format States and Stream Manipulators	632
13.7.1	Trailing Zeros and Decimal Points (showpoint)	633
13.7.2	Justification (left, right and internal)	634

13.7.3	Padding (<code>fill</code> , <code>setfill</code>)	635
13.7.4	Integral Stream Base (<code>dec</code> , <code>oct</code> , <code>hex</code> , <code>showbase</code>)	637
13.7.5	Floating-Point Numbers; Scientific and Fixed Notation (<code>scientific</code> , <code>fixed</code>)	637
13.7.6	Uppercase/Lowercase Control (<code>uppercase</code>)	638
13.7.7	Specifying Boolean Format (<code>boolalpha</code>)	639
13.7.8	Setting and Resetting the Format State via Member Function flags	640
13.8	Stream Error States	641
13.9	Tying an Output Stream to an Input Stream	644
13.10	Wrap-Up	645

14 File Processing

655

14.1	Introduction	656
14.2	Files and Streams	656
14.3	Creating a Sequential File	657
14.3.1	Opening a File	658
14.3.2	Opening a File via the <code>open</code> Member Function	659
14.3.3	Testing Whether a File Was Opened Successfully	659
14.3.4	Overloaded <code>bool</code> Operator	660
14.3.5	Processing Data	660
14.3.6	Closing a File	660
14.3.7	Sample Execution	661
14.4	Reading Data from a Sequential File	661
14.4.1	Opening a File for Input	662
14.4.2	Reading from the File	662
14.4.3	File-Position Pointers	662
14.4.4	Case Study: Credit Inquiry Program	663
14.5	C++14: Reading and Writing Quoted Text	666
14.6	Updating Sequential Files	667
14.7	Random-Access Files	668
14.8	Creating a Random-Access File	669
14.8.1	Writing Bytes with <code>ostream</code> Member Function <code>write</code>	669
14.8.2	Converting Between Pointer Types with the <code>reinterpret_cast</code> Operator	669
14.8.3	Credit-Processing Program	670
14.8.4	Opening a File for Output in Binary Mode	673
14.9	Writing Data Randomly to a Random-Access File	673
14.9.1	Opening a File for Input and Output in Binary Mode	675
14.9.2	Positioning the File-Position Pointer	675
14.10	Reading from a Random-Access File Sequentially	675
14.11	Case Study: A Transaction-Processing Program	677
14.12	Object Serialization	683
14.13	Wrap-Up	684

15	Standard Library Containers and Iterators	695
15.1	Introduction	696
15.2	Introduction to Containers	698
15.3	Introduction to Iterators	702
15.4	Introduction to Algorithms	707
15.5	Sequence Containers	707
15.5.1	vector Sequence Container	708
15.5.2	list Sequence Container	715
15.5.3	deque Sequence Container	720
15.6	Associative Containers	721
15.6.1	multiset Associative Container	722
15.6.2	set Associative Container	725
15.6.3	multimap Associative Container	727
15.6.4	map Associative Container	729
15.7	Container Adapters	730
15.7.1	stack Adapter	731
15.7.2	queue Adapter	733
15.7.3	priority_queue Adapter	734
15.8	Class bitset	735
15.9	Wrap-Up	737
16	Standard Library Algorithms	747
16.1	Introduction	748
16.2	Minimum Iterator Requirements	748
16.3	Lambda Expressions	750
16.3.1	Algorithm for_each	751
16.3.2	Lambda with an Empty Introducer	751
16.3.3	Lambda with a Nonempty Introducer—Capturing Local Variables	752
16.3.4	Lambda Return Types	752
16.4	Algorithms	752
16.4.1	fill, fill_n, generate and generate_n	752
16.4.2	equal, mismatch and lexicographical_compare	755
16.4.3	remove, remove_if, remove_copy and remove_copy_if	758
16.4.4	replace, replace_if, replace_copy and replace_copy_if	761
16.4.5	Mathematical Algorithms	763
16.4.6	Basic Searching and Sorting Algorithms	766
16.4.7	swap, iter_swap and swap_ranges	771
16.4.8	copy_backward, merge, unique and reverse	772
16.4.9	inplace_merge, unique_copy and reverse_copy	775
16.4.10	Set Operations	777
16.4.11	lower_bound, upper_bound and equal_range	780
16.4.12	min, max, minmax and minmax_element	782
16.5	Function Objects	784
16.6	Standard Library Algorithm Summary	787
16.7	Wrap-Up	789

17	Exception Handling: A Deeper Look	797
17.1	Introduction	798
17.2	Exception-Handling Flow of Control; Defining an Exception Class	799
17.2.1	Defining an Exception Class to Represent the Type of Problem That Might Occur	799
17.2.2	Demonstrating Exception Handling	800
17.2.3	Enclosing Code in a try Block	801
17.2.4	Defining a catch Handler to Process a DivideByZeroException	802
17.2.5	Termination Model of Exception Handling	802
17.2.6	Flow of Program Control When the User Enters a Nonzero Denominator	803
17.2.7	Flow of Program Control When the User Enters a Denominator of Zero	803
17.3	Rethrowing an Exception	804
17.4	Stack Unwinding	806
17.5	When to Use Exception Handling	807
17.6	noexcept: Declaring Functions That Do Not Throw Exceptions	808
17.7	Constructors, Destructors and Exception Handling	808
17.7.1	Destructors Called Due to Exceptions	808
17.7.2	Initializing Local Objects to Acquire Resources	809
17.8	Processing new Failures	809
17.8.1	new Throwing bad_alloc on Failure	809
17.8.2	new Returning nullptr on Failure	810
17.8.3	Handling new Failures Using Function set_new_handler	811
17.9	Class unique_ptr and Dynamic Memory Allocation	812
17.9.1	unique_ptr Ownership	814
17.9.2	unique_ptr to a Built-In Array	815
17.10	Standard Library Exception Hierarchy	815
17.11	Wrap-Up	817
18	Introduction to Custom Templates	823
18.1	Introduction	824
18.2	Class Templates	825
18.2.1	Creating Class Template Stack<T>	826
18.2.2	Class Template Stack<T>'s Data Representation	827
18.2.3	Class Template Stack<T>'s Member Functions	827
18.2.4	Declaring a Class Template's Member Functions Outside the Class Template Definition	828
18.2.5	Testing Class Template Stack<T>	828
18.3	Function Template to Manipulate a Class-Template Specialization Object	830
18.4	Nontype Parameters	832
18.5	Default Arguments for Template Type Parameters	832
18.6	Overloading Function Templates	833
18.7	Wrap-Up	833

19	Custom Templated Data Structures	837
19.1	Introduction	838
19.1.1	Always Prefer the Standard Library's Containers, Iterators and Algorithms, if Possible	839
19.1.2	Special Section: Building Your Own Compiler	839
19.2	Self-Referential Classes	839
19.3	Linked Lists	840
19.3.1	Testing Our Linked List Implementation	842
19.3.2	Class Template <code>ListNode</code>	845
19.3.3	Class Template <code>List</code>	846
19.3.4	Member Function <code>insertAtFront</code>	849
19.3.5	Member Function <code>insertAtBack</code>	850
19.3.6	Member Function <code>removeFromFront</code>	850
19.3.7	Member Function <code>removeFromBack</code>	851
19.3.8	Member Function <code>print</code>	852
19.3.9	Circular Linked Lists and Double Linked Lists	853
19.4	Stacks	854
19.4.1	Taking Advantage of the Relationship Between <code>Stack</code> and <code>List</code>	855
19.4.2	Implementing a Class Template <code>Stack</code> Class Based By Inheriting from <code>List</code>	855
19.4.3	Dependent Names in Class Templates	856
19.4.4	Testing the <code>Stack</code> Class Template	857
19.4.5	Implementing a Class Template <code>Stack</code> Class With Composition of a <code>List</code> Object	858
19.5	Queues	859
19.5.1	Applications of Queues	859
19.5.2	Implementing a Class Template <code>Queue</code> Class Based By Inheriting from <code>List</code>	860
19.5.3	Testing the <code>Queue</code> Class Template	861
19.6	Trees	863
19.6.1	Basic Terminology	863
19.6.2	Binary Search Trees	864
19.6.3	Testing the <code>Tree</code> Class Template	864
19.6.4	Class Template <code>TreeNode</code>	866
19.6.5	Class Template <code>Tree</code>	867
19.6.6	<code>Tree</code> Member Function <code>insertNodeHelper</code>	869
19.6.7	<code>Tree</code> Traversal Functions	869
19.6.8	Duplicate Elimination	870
19.6.9	Overview of the Binary Tree Exercises	870
19.7	Wrap-Up	871
20	Searching and Sorting	881
20.1	Introduction	882
20.2	Searching Algorithms	883
20.2.1	Linear Search	883

20 Contents

20.2.2	Binary Search	886
20.3	Sorting Algorithms	890
20.3.1	Insertion Sort	891
20.3.2	Selection Sort	893
20.3.3	Merge Sort (A Recursive Implementation)	895
20.4	Wrap-Up	902

21 Class `string` and String Stream Processing: A Deeper Look 909

21.1	Introduction	910
21.2	<code>string</code> Assignment and Concatenation	911
21.3	Comparing strings	913
21.4	Substrings	916
21.5	Swapping strings	916
21.6	<code>string</code> Characteristics	917
21.7	Finding Substrings and Characters in a <code>string</code>	920
21.8	Replacing Characters in a <code>string</code>	921
21.9	Inserting Characters into a <code>string</code>	923
21.10	Conversion to Pointer-Based <code>char*</code> Strings	924
21.11	Iterators	926
21.12	String Stream Processing	927
21.13	C++11 Numeric Conversion Functions	930
21.14	Wrap-Up	932

22 Bits, Characters, C Strings and `structs` 939

22.1	Introduction	940
22.2	Structure Definitions	940
22.3	<code>typedef</code> and <code>using</code>	942
22.4	Example: Card Shuffling and Dealing Simulation	942
22.5	Bitwise Operators	945
22.6	Bit Fields	954
22.7	Character-Handling Library	958
22.8	C String-Manipulation Functions	963
22.9	C String-Conversion Functions	970
22.10	Search Functions of the C String-Handling Library	975
22.11	Memory Functions of the C String-Handling Library	979
22.12	Wrap-Up	983

Chapters on the Web 999

A Operator Precedence and Associativity 1001

B ASCII Character Set 1003

C	Fundamental Types	1005
D	Number Systems	1007
D.1	Introduction	1008
D.2	Abbreviating Binary Numbers as Octal and Hexadecimal Numbers	1011
D.3	Converting Octal and Hexadecimal Numbers to Binary Numbers	1012
D.4	Converting from Binary, Octal or Hexadecimal to Decimal	1012
D.5	Converting from Decimal to Binary, Octal or Hexadecimal	1013
D.6	Negative Binary Numbers: Two's Complement Notation	1015
E	Preprocessor	1021
E.1	Introduction	1022
E.2	<code>#include</code> Preprocessing Directive	1022
E.3	<code>#define</code> Preprocessing Directive: Symbolic Constants	1023
E.4	<code>#define</code> Preprocessing Directive: Macros	1023
E.5	Conditional Compilation	1025
E.6	<code>#error</code> and <code>#pragma</code> Preprocessing Directives	1027
E.7	Operators <code>#</code> and <code>##</code>	1027
E.8	Predefined Symbolic Constants	1027
E.9	Assertions	1028
E.10	Wrap-Up	1028
	Appendices on the Web	1033
	Index	1035

Chapters 23–26 and Appendices F–J are PDF documents posted online at the book's Companion Website, which is accessible from

<http://www.pearsonglobaleditions.com/deitel>

See the inside front cover for more information.

23 Other Topics

24 C++11 and C++14: Additional Features

25 ATM Case Study, Part 1: Object-Oriented Design with the UM

26 ATM Case Study, Part 2: Implementing an Object-Oriented Design

F C Legacy Code Topics

G UML: Additional Diagram Types

H Using the Visual Studio Debugger

I Using the GNU C++ Debugger

J Using the Xcode Debugger



Preface

Welcome to the C++ computer programming language and *C++ How to Program, Tenth Edition*. We believe that this book and its support materials will give you an informative, challenging and entertaining introduction to C++. The book presents leading-edge computing technologies in a friendly manner appropriate for introductory college course sequences, based on the curriculum recommendations of two key professional organizations—the ACM and the IEEE.¹

If you haven't already done so, please read the back cover and check out the additional reviewer comments on the inside back cover and the facing page—these capture the essence of the book concisely. In this Preface we provide more detail for students, instructors and professionals.

At the heart of the book is the Deitel signature *live-code approach*—we present most concepts in the context of complete working programs followed by sample executions, rather than in code snippets. Read the **Before You Begin** section to learn how to set up your Linux-based, Windows-based or Apple OS X-based computer to run the hundreds of code examples. All the source code is available at

<http://www.pearsonglobaleditions.com/deitel>

Use the source code we provide to run each program as you study it.

Contacting the Authors

As you read the book, if you have questions, we're easy to reach at

deitel@deitel.com

We'll respond promptly. For book updates, visit

<http://www.deitel.com/books/cpphttp10>

Join the Deitel & Associates, Inc. Social Media Communities

Join the Deitel social media communities on

- Facebook®—<http://facebook.com/DeitelFan>
- LinkedIn®—<http://bit.ly/DeitelLinkedIn>

1. *Computer Science Curricula 2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, December 20, 2013, The Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM), IEEE Computer Society.

- Twitter[®]—<http://twitter.com/deitel>
- Google+[™]—<http://google.com/+DeitelFan>
- YouTube[®]—<http://youtube.com/DeitelTV>

and subscribe to the *Deitel[®] Buzz Online* newsletter

<http://www.deitel.com/newsletter/subscribe.html>

The C++11 and C++14 Standards

These are exciting times in the programming languages community with each of the major languages striving to keep pace with compelling new programming technologies. In the three decades of C++’s development prior to 2011, only a few new versions of the language were released. Now the ISO C++ Standards Committee is committed to releasing a new standard every three years and the compiler vendors are building in the new features promptly. *C++ How to Program, 10/e* is based on the C++11 and C++14 standards published in 2011 and 2014, respectively. C++17 is already under active development. Throughout the book, C++11 and C++14 features are marked with the “11” and “14” icons, respectively, that you see here in the margin. Fig. 1 lists the book’s first references to the 77 C++11 and C++14 features we discuss.

11
14

C++11 and C++14 features in *C++ How to Program, 10/e*

<i>Chapter 3</i> In-class initializers	<i>Chapter 8</i> begin/end functions nullptr	<i>Chapter 13</i> operator bool for streams
<i>Chapter 4</i> Keywords new in C++11	<i>Chapter 9</i> Delegating constructors	<i>Chapter 14</i> quoted stream manipulator (C++14)
<i>Chapter 5</i> long long int type	<i>Chapter 10</i> deleted member functions explicit conversion operators List initializing a dynamically allocated array	string objects for file names
<i>Chapter 6</i> Non-deterministic random number generation Scoped enums Specifying the type of an enum's constants Unsigned long long int Using ' to separate groups of digits in a numeric literals (C++14)	List initializers in constructor calls string object literals (C++14)	<i>Chapter 15</i> cbegin/cend container member functions Compiler fix for >> in template types crbegin/crend container member functions forward_list container Global functions cbegin/ cend, rbegin/rend and crbegin/crend (C++14)
<i>Chapter 7</i> array container auto for type inference List initializing a vector Range-based for statement	Inheriting base-class constructors	Heterogeneous lookup in associative containers (C++14)
	<i>Chapter 12</i> defaulted member functions override keyword	Immutable keys in associative containers

Fig. 1 | First references to C++11 and C++14 features in *C++ How to Program, 10/e*. (Part 1 of 2.)

C++11 and C++14 features in *C++ How to Program, 10/e*

<p><i>Chapter 15 (cont.)</i> insert container member functions return iterators List initialization of key–value pairs List initialization of pairs Return value list initialization shrink_to_fit vector/deque member function</p> <p><i>Chapter 16</i> all_of algorithm any_of algorithm copy_if algorithm copy_n algorithm equal algorithm that accepts two ranges (C++14) find_if_not algorithm Generic lambdas (C++14) Lambda expressions min and max algorithms with initializer_list parameters minmax algorithm</p>	<p><i>Chapter 16 (cont.)</i> minmax_element algorithm mismatch algorithm that accepts two ranges (C++14) none_of algorithm random_shuffle is deprecated (C++14)—replaced with shuffle and C++11 random-number generation swap non-member function</p> <p><i>Chapter 17</i> make_unique to create a unique_ptr (C++14) noexcept unique_ptr smart pointer</p> <p><i>Chapter 18</i> Default type arguments in function templates</p> <p><i>Chapter 21</i> Numeric conversion functions</p> <p><i>Chapter 22</i> Binary literals (C++14)</p>	<p><i>Chapter 24</i> Aggregate member initialization (C++14) auto and decltype(auto) on return types(C++14) constexpr updated(C++14) decltype move algorithm Move assignment operators move_backward algorithm Move constructors Regular expressions Rvalue references shared_ptr smart pointer static_assert objects for file names Trailing return types for functions tuple variadic template tuple addressing via type (C++14) weak_ptr smart pointer</p>
--	---	---

Fig. 1 | First references to C++11 and C++14 features in *C++ How to Program, 10/e*. (Part 2 of 2.)

Key Features of C++ How to Program, 10/e

- *Conforms to the C++11 standard and the new C++14 standard.*
- *Code thoroughly tested on three popular industrial-strength C++14 compilers.* We tested the code examples on GNU™ C++ 5.2.1, Microsoft® Visual Studio® 2015 Community edition and Apple® Clang/LLVM in Xcode® 7.
- *Smart pointers.* Smart pointers help you avoid dynamic memory management errors by providing additional functionality beyond that of built-in pointers. We discuss unique_ptr in Chapter 17, and shared_ptr and weak_ptr in Chapter 24.
- *Early coverage of Standard Library containers, iterators and algorithms, enhanced with C++11 and C++14 capabilities.* The treatment of Standard Library containers, iterators and algorithms in Chapters 15 and 16 has been enhanced with additional C++11 and C++14 features. The vast majority of your data structure needs can be fulfilled by *reusing* these Standard Library capabilities. We'll show you how to build your own *custom* data structures in Chapter 19.
- *Online Chapter 24, C++11 and C++14 Additional Topics.* This chapter includes discussions of regular expressions, shared_ptr and weak_ptr smart pointers, move semantics, multithreading, tuples, decltype, constexpr and more (see Fig. 1).

- *Random-number generation, simulation and game playing.* To help make programs more secure, we include a treatment of C++11's non-deterministic random-number generation capabilities.
- *Pointers.* We provide thorough coverage of the built-in pointer capabilities and the intimate relationship among built-in pointers, C strings and built-in arrays.
- *Visual presentation of searching and sorting, with a simple explanation of Big O.*
- *Printed book contains core content; additional content is online.* Several online chapters and appendices are included. These are available in searchable PDF format on the book's password-protected Companion Website—see the access card information on the inside front cover.
- *Debugger appendices.* On the book's Companion Website we provide Appendix H, Using the Visual Studio Debugger, Appendix I, Using the GNU C++ Debugger and Appendix J, Using the Xcode Debugger.

New in This Edition

- Discussions of the new C++14 capabilities.
- Further integration of C++11 capabilities into the code examples, because the latest compilers are now supporting these features.
- Uniform initialization with list initializer syntax.
- Always using braces in control statements, even for single-statement bodies:

```
if (condition) {
    single-statement or multi-statement body
}
```

- Replaced the `Gradebook` class with `Account`, `Student` and `DollarAmount` class case studies in Chapters 3, 4 and 5, respectively. `DollarAmount` processes monetary amounts precisely for business applications.
- C++14 digit separators in large numeric literals.
- `Type &x` is now `Type& x` in accordance with industry idiom.
- `Type *x` is now `Type* x` in accordance with industry idiom.
- Using C++11 scoped enums rather than traditional C enums.
- We brought our terminology in line with the C++ standard.
- Key terms in summaries now appear in bold for easy reference.
- Removed extra spaces inside `[]`, `()`, `<>` and `{}` delimiters.
- Replaced most `print` member functions with `toString` member functions to make classes more flexible—for example, returning a `string` gives the client code the option of displaying it on the screen, writing it to a file, concatenating it with other strings, etc.

- Now using `ostringstream` to create formatted strings for items like the string representations of a `Time`, rather than outputting formatted data directly to the standard output.
- For simplicity, we deferred using the three-file architecture from Chapter 3 to Chapter 9, so all early class examples define the entire class in a header.
- We reimplement Chapter 10’s Array class operator-overloading example with `unique_ptr`s in Chapter 24. Using raw pointers and dynamic-memory allocation with `new` and `delete` is a source of subtle programming errors, especially “memory leaks”—`unique_ptr` and the other smart pointer types help prevent such errors.
- Using lambdas rather than function pointers in Chapter 16, Standard Library Algorithms. This will get readers comfortable with lambdas, which can be combined with various Standard Library algorithms to perform functional programming in C++. We’re planning a more in-depth treatment of functional programming for *C++ How to Program, 11/e*.
- Enhanced Chapter 24 with additional C++14 features.

Object-Oriented Programming

- **Early-objects approach.** The book introduces the basic concepts and terminology of object technology in Chapter 1. You’ll develop your first customized classes and objects in Chapter 3. We worked hard to make this chapter especially accessible to novices. Presenting objects and classes early gets you “thinking about objects” immediately and mastering these concepts more thoroughly.²
- **C++ Standard Library string.** C++ offers *two* types of strings—`string` class objects (which we begin using in Chapter 3) and C-style pointer-based strings. We’ve replaced most occurrences of C strings with instances of C++ class `string` to make programs more robust and eliminate many of the security problems of C strings. We continue to discuss C strings later in the book to prepare you for working with the legacy code in industry. In new development, you should favor `string` objects.
- **C++ Standard Library array.** C++ offers *three* types of arrays—arrays and vectors (which we start using in Chapter 7) and C-style, pointer-based arrays which we discuss in Chapter 8. Our primary treatment of arrays uses the Standard Library’s array and vector class templates instead of built-in, C-style, pointer-based arrays. We still cover built-in arrays because they remain useful in C++ and so that you’ll be able to read legacy code. In new development, you should favor class template array and vector objects.
- **Crafting valuable classes.** A key goal of this book is to prepare you to build valuable reusable classes. Chapter 10 begins with a test-drive of class template `string` so you can see an elegant use of operator overloading before you implement your own customized class with overloaded operators. In the Chapter 10 case study,

2. For courses that require a late-objects approach, consider our pre-C++11 book *C++ How to Program, Late Objects Version*, which begins with six chapters on programming fundamentals (including two on control statements) and continues with seven chapters that gradually introduce object-oriented programming concepts.

you'll build your own custom Array class, then in the Chapter 18 exercises you'll convert it to a class template. You will have truly crafted valuable classes.

- *Case studies in object-oriented programming.* We provide several well-engineered real-world case studies, including the Account class in Chapter 3, Student class in Chapter 4, DollarAmount class in Chapter 5, GradeBook class in Chapter 7, the Time class in Chapter 9, the Employee class in Chapters 11–12 and more.
- *Optional case study: Using the UML to develop an object-oriented design and C++ implementation of an ATM.* The UML™ (Unified Modeling Language™) is the industry-standard graphical language for modeling object-oriented systems. We introduce the UML in the early chapters. Online Chapters 25 and 26 include an *optional* object-oriented design case study using the UML. We design and fully implement the software for a simple automated teller machine (ATM). We analyze a typical requirements document that specifies the system to be built. We determine the classes needed to implement that system, the attributes the classes need to have, the behaviors the classes need to exhibit and we specify how objects of the classes must interact with one another to meet the system requirements. From the design we produce a complete C++ implementation. Students often report that the case study helps them “tie it all together” and truly understand object orientation.
- *Understanding how polymorphism works.* Chapter 12 contains a detailed diagram and explanation of how C++ typically implements polymorphism, virtual functions and dynamic binding “under the hood.”
- *Object-oriented exception handling.* We integrate basic exception handling early in the book (Chapter 7). Instructors can easily pull more detailed material forward from Chapter 17, Exception Handling: A Deeper Look.
- *Custom template-based data structures.* We provide a rich multi-chapter treatment of data structures—see the Data Structures module in the chapter dependency chart (Fig. 5).
- *Three programming paradigms.* We discuss *structured programming*, *object-oriented programming* and *generic programming*.

Hundreds of Code Examples

We include a broad range of example programs selected from computer science, information technology, business, simulation, game playing and other topics. The examples are accessible to students in novice-level and intermediate-level C++ courses (Fig. 2).

Examples	
Account class	BasePlusCommissionEmployee class
Array class case study	Binary tree creation and traversal
Author class	BinarySearch test program
Bank account program	Card shuffling and dealing
Bar chart printing program	ClientData class

Fig. 2 | A sampling of the book's examples. (Part I of 2.)

Examples

CommissionEmployee class	Poll analysis program
Comparing strings	Polymorphism demonstration
Compilation and linking process	Preincrementing and postincrementing
Compound interest calculations with for	priority_queue adapter class
Converting string objects to C strings	queue adapter class
Counter-controlled repetition	Random-access files
Dice game simulation	Random number generation
DollarAmount class	Recursive function factorial
Credit inquiry program	Rolling a six-sided die 60,000,000 times
Date class	SalariedEmployee class
Downcasting and runtime type information	SalesPerson class
Employee class	Searching and sorting algorithms of the Standard Library
explicit constructor	Sequential files
fibonacci function	set class template
fill algorithms	shared_ptr program
Specializations of function template printArray	stack adapter class
generate algorithms	Stack class
GradeBook Class	Stack unwinding
Initializing an array in a declaration	Standard Library string class program
Input from an istream object	Stream manipulator showbase
Iterative factorial solution	string assignment and concatenation
Lambda expressions	string member function substr
Linked list manipulation	Student class
map class template	Summing integers with the for statement
Mathematical algorithms of the Standard Library	Time class
maximum function template	unique_ptr object managing dynamically allocated memory
Merge sort program	Validating user input with regular expressions
multiset class template	vector class class
new throwing bad_alloc on failure	
PhoneNumber class	

Fig. 2 | A sampling of the book's examples. (Part 2 of 2.)

Exercises

- *Self-Review Exercises and Answers.* Extensive self-review exercises *and* answers are included for self-study.
- *Interesting, entertaining and challenging exercises.* Each chapter concludes with a substantial set of exercises, including simple recall of important terminology and concepts, identifying the errors in code samples, writing individual program statements, writing small portions of C++ classes and member and non-member functions, writing complete programs and implementing major projects. Figure 3 lists a sampling of the book's exercises, including our *Making a Difference* exercises, which encourage you to use computers and the Internet to research and work on significant social problems. We hope you'll approach these exercises with your own values, politics and beliefs.

Exercises		
Airline Reservations System	Credit Limits	Phishing Scanner
Advanced String-Manipulation	Crossword Puzzle Generator	Pig Latin
Bubble Sort	Cryptograms	Polymorphic Banking Program
Building Your Own Compiler	De Morgan's Laws	Using Account Hierarchy
Building Your Own Computer	Dice Rolling	Pythagorean Triples
Calculating Salaries	Eight Queens	Salary Calculator
CarbonFootprint Abstract	Emergency Response	Sieve of Eratosthenes
Class: Polymorphism	Enforcing Privacy with Cryptography	Simple Decryption
Card Shuffling and Dealing	Facebook User Base Growth	Simple Encryption
Computer-Assisted Instruction	Fibonacci Series	SMS Language
Computer-Assisted Instruction: Difficulty Levels	Gas Mileage	Spam Scanner
Computer-Assisted Instruction: Monitoring Student Performance	Global Warming Facts Quiz	Spelling Checker
Computer-Assisted Instruction: Reducing Student Fatigue	Guess the Number Game	Target-Heart-Rate Calculator
Computer-Assisted Instruction: Varying the Types of Problems	Hangman Game	Tax Plan Alternatives; The "Fair Tax"
Cooking with Healthier Ingredients	Health Records	Telephone number word generator
Craps Game Modification	Knight's Tour	"The Twelve Days of Christmas" Song
	Limericks	Tortoise and the Hare Simulation
	Maze Traversal: Generating Mazes Randomly	Towers of Hanoi
	Morse Code	World Population Growth
	Payroll System Modification	
	Peter Minuit Problem	

Fig. 3 | A sampling of the book's exercises.

Illustrations and Figures

Abundant tables, line drawings, UML diagrams, programs and program outputs are included. A sampling of the book's drawings and diagrams is shown in (Fig. 4).

Drawings and diagrams		
<i>Main text drawings and diagrams</i>		
Account class diagrams	while repetition statement	Pass-by-value and pass-by-reference analysis
Data hierarchy	UML activity diagram	Inheritance hierarchy diagrams
Multiple-source-file compilation and linking	for repetition statement UML activity diagram	Function-call stack and activation records
Order in which a second-degree polynomial is evaluated	do...while repetition statement UML activity diagram	Recursive calls to function fibonacci
if single-selection statement activity diagram	switch multiple-selection statement activity diagram	Pointer arithmetic diagrams
if...else double-selection statement activity diagram	C++'s single-entry/single-exit control statements	CommunityMember Inheritance hierarchy

Fig. 4 | A sampling of the book's drawings and diagrams. (Part 1 of 2.)

Drawings and diagrams

Main text drawings and diagrams (cont.)

Shape inheritance hierarchy public, protected and private inheritance	Graphical representation of a list Operation <code>insertAtFront</code> represented graphically	Operation <code>removeFromBack</code> represented graphically Circular, singly linked list
Employee hierarchy UML class diagram	Operation <code>insertAtBack</code> represented graphically	Doubly linked list Circular, doubly linked list
How virtual function calls work	Operation <code>removeFromFront</code> represented graphically	Graphical representation of a binary tree
Two self-referential class objects linked together		

(Optional) ATM Case Study drawings and diagrams

Use case diagram for the ATM system from the User's perspective	Classes in the ATM system with attributes and operations	Class diagram showing composition relationships of a class <code>Car</code>
Class diagram showing an association among classes	Communication diagram of the ATM executing a balance inquiry	Class diagram for the ATM system model including class <code>Deposit</code>
Class diagram showing composition relationships	Communication diagram for executing a balance inquiry	Activity diagram for a <code>Deposit</code> transaction
Class diagram for the ATM system model	Sequence diagram that models a <code>Withdrawal</code> executing	Sequence diagram that models a <code>Deposit</code> executing
Classes with attributes		
State diagram for the ATM	Use case diagram for a modified version of our ATM system that also allows	
Activity diagram for a <code>BalanceInquiry</code> transaction	users to transfer money between accounts	
Activity diagram for a <code>Withdrawal</code> transaction		

Fig. 4 | A sampling of the book's drawings and diagrams. (Part 2 of 2.)

Dependency Chart

C++ How to Program, 10/e is appropriate for most introductory one-and-two-course programming sequences, often called CS1 and CS2. The chart in Fig. 5 shows the dependencies among the chapters to help instructors plan their syllabi. The chart shows the book's modular organization.

Teaching Approach

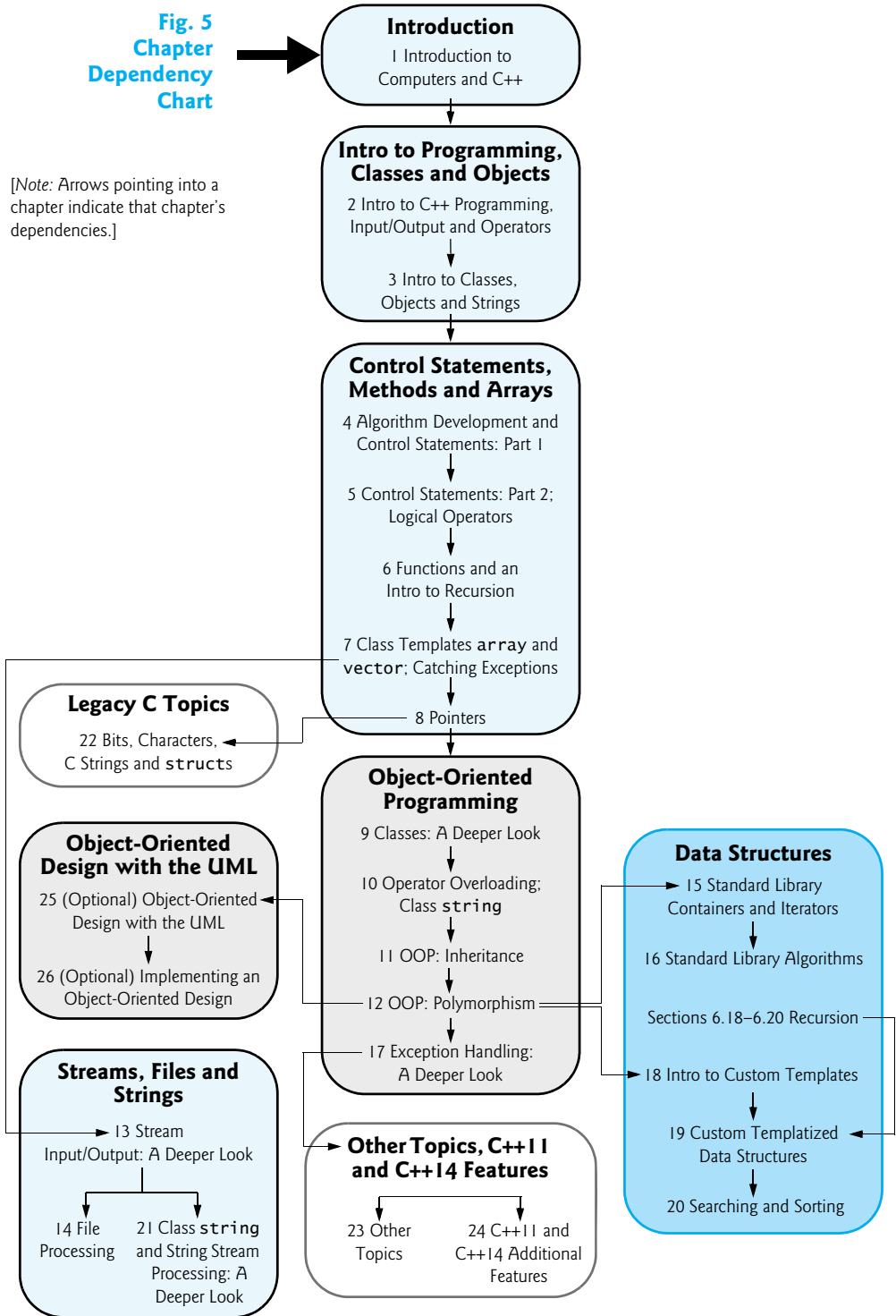
C++ How to Program, 10/e, contains a rich collection of examples. We stress program clarity and concentrate on building well-engineered software.

Live-code approach. The book is loaded with “live-code” examples—most new concepts are presented in *complete working C++ applications*, followed by one or more executions showing program inputs and outputs.

Rich early coverage of C++ fundamentals. Chapter 2 provides a friendly introduction to C++ programming. We include in Chapters 4 and 5 a clear treatment of control statements and algorithm development.

Fig. 5
Chapter
Dependency
Chart

[Note: Arrows pointing into a chapter indicate that chapter's dependencies.]



Syntax coloring. For readability, we syntax color all the C++ code, similar to the way most C++ integrated-development environments and code editors syntax color code. Our coloring conventions are as follows:

```

comments appear like this
keywords appear like this
constants and literal values appear like this
all other code appears in black

```

Code highlighting. We place shaded rectangles around the new features in each program.

Using fonts for emphasis. We color the defining occurrence of each key term in **bold colored** text for easy reference. We emphasize on-screen components in the **bold Helvetica** font (e.g., the **File** menu) and C++ program text in the **Lucida** font (for example, `int x = 5;`).

Objectives. We clearly state the chapter objectives.

Programming tips. We include programming tips to help you focus on key aspects of program development. These tips and practices represent the best we've gleaned from a combined eight decades of teaching and industry experience.



Good Programming Practices

The Good Programming Practices call attention to techniques that will help you produce programs that are clearer, more understandable and more maintainable.



Common Programming Errors

Pointing out these Common Programming Errors reduces the likelihood that you'll make them.



Error-Prevention Tips

These tips contain suggestions for exposing and removing bugs from your programs; many describe aspects of C++ that prevent bugs from getting into programs in the first place.



Performance Tips

These tips highlight opportunities for making your programs run faster or minimizing the amount of memory that they occupy.



Portability Tips

These tips help you write code that will run on a variety of platforms.



Software Engineering Observations

These tips highlight architectural and design issues that affect the construction of software systems, especially large-scale systems.

Summary Bullets. We present a section-by-section, bullet-list summary of each chapter. Each key term is in **bold** followed by the page number of the term's defining occurrence.

Index. For convenient reference, we've included an extensive index, with defining occurrences of key terms highlighted with a **bold** page number.

Secure C++ Programming

It's difficult to build industrial-strength systems that stand up to attacks from viruses, worms, and other forms of "malware." Today, via the Internet, such attacks can be instantaneous and global in scope. Building security into software from the beginning of the development cycle can greatly reduce vulnerabilities.

The CERT[®] Coordination Center (www.cert.org) was created to analyze and respond promptly to attacks. CERT—the Computer Emergency Response Team—is a government-funded organization within the Carnegie Mellon University Software Engineering Institute[™]. CERT publishes and promotes secure coding standards for various popular programming languages to help software developers implement industrial-strength systems which avoid the programming practices that leave systems open to attacks.

We'd like to thank Robert C. Seacord, an adjunct professor in the Carnegie Mellon University School of Computer Science and former Secure Coding Manager at CERT. Mr. Seacord was a technical reviewer for our book, *C How to Program, 7/e*, where he scrutinized our C programs from a security standpoint, recommending that we adhere to key guidelines of the *CERT C Secure Coding Standard*.

We've done the same for *C++ How to Program, 10/e*, adhering to key guidelines of the *CERT C++ Secure Coding Standard*, which you can find at:

<http://www.securecoding.cert.org>

We were pleased to discover that we've already been recommending many of these coding practices in our books since the early 1990s. We upgraded our code and discussions to conform to these practices, as appropriate for an introductory/intermediate-level textbook. If you'll be building industrial-strength C++ systems, consider reading *Secure Coding in C and C++, Second Edition* (Robert Seacord, Addison-Wesley Professional, 2013).

Online Chapters, Appendices and Other Content

The book's Companion Website, which is accessible at

<http://www.pearsonglobaleditions.com/deitel>

(see the inside front cover for your access key) contains the following videos as well as chapters and appendices in searchable PDF format:

- *VideoNotes*—The Companion Website (see the inside front cover for your access key) also includes extensive videos. Watch and listen as co-author Paul Deitel discusses in-depth the key code examples from the book's core programming-fundamentals and object-oriented-programming chapters.
- Chapter 23, Other Topics
- Chapter 24, C++11 and C++14 Additional Topics
- Chapter 25, ATM Case Study, Part 1: Object-Oriented Design with the UML
- Chapter 26, ATM Case Study, Part 2: Implementing an Object-Oriented Design
- Appendix F, C Legacy Code Topics
- Appendix G, UML 2: Additional Diagram Types

- Appendix H, Using the Visual Studio Debugger
- Appendix I, Using the GNU C++ Debugger
- Appendix J, Using the Xcode Debugger
- Building Your Own Compiler exercise descriptions from Chapter 19 (posted at the Companion Website.)

Obtaining the Software Used in C++ How to Program, 10/e

We wrote the code examples in *C++ How to Program, 10/e* using the following free C++ development tools:

- Microsoft’s free Visual Studio Community 2015 edition, which includes Visual C++ and other Microsoft development tools. This runs on Windows and is available for download at

<https://www.visualstudio.com/products/visual-studio-community-vs>

- GNU’s free GNU C++ 5.2.1. GNU C++ is already installed on most Linux systems and can also be installed on Mac OS X and Windows systems. There are many versions of Linux—known as Linux distributions—that use different techniques for performing software upgrades. Check your distribution’s online documentation for information on how to upgrade GNU C++ to the latest version. GNU C++ is available at

<http://gcc.gnu.org/install/binaries.html>

- Apple’s free Xcode, which OS X users can download from the Mac App Store—click the app’s icon in the dock at the bottom of your screen, then search for Xcode in the app store.

Instructor Supplements

The following supplements are available to *qualified instructors only* through Pearson Education’s Instructor Resource Center (<http://www.pearsonglobal editions.com/deitel>):

- *Solutions Manual* contains solutions to most of the end-of-chapter exercises. We include *Making a Difference* exercises, many with solutions. **Please do not write to us requesting access to the Pearson Instructor’s Resource Center. Access is restricted to college instructors teaching from the book.** Instructors may obtain access only through their Pearson representatives.

Solutions are *not* provided for “project” exercises. Check out our Programming Projects Resource Center for lots of additional exercise and project possibilities.

<http://www.deitel.com/ProgrammingProjects>

- *Test Item File* of multiple-choice questions.
- *Customizable PowerPoint® slides* containing all the code and figures in the text, plus bulleted items that summarize key points in the text.

Online Practice and Assessment with MyProgrammingLab™

MyProgrammingLab™ helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyProgrammingLab improves the programming competence of beginning students who often struggle with the basic concepts and paradigms of popular high-level programming languages.

An optional self-study and homework tool, a MyProgrammingLab course consists of hundreds of small practice problems organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted by students for review.

For a full demonstration, to see feedback from instructors and students or to get started using MyProgrammingLab in your course, visit

<http://www.myprogramminglab.com>

Acknowledgments

We'd like to thank Barbara Deitel of Deitel & Associates, Inc. for long hours devoted to this project. She painstakingly researched the new capabilities of C++11 and C++14.

We're fortunate to have worked with the dedicated team of publishing professionals at Pearson Higher Education. We appreciate the guidance, wisdom and energy of Tracy Johnson, Executive Editor, Computer Science. Kristy Alaura did an extraordinary job recruiting the book's reviewers and managing the review process. Bob Engelhardt did a wonderful job bringing the book to publication.

Finally, thanks to Abbey Deitel, former President of Deitel & Associates, Inc., and a graduate of Carnegie Mellon University's Tepper School of Management where she received a B.S. in Industrial Management. Abbey managed the business operations of Deitel & Associates, Inc. for 17 years, along the way co-authoring a number of our publications, including the previous C++ *How to Program* editions' versions of Chapter 1.

Reviewers

We wish to acknowledge the efforts of our reviewers. Over its ten editions, the book has been scrutinized by academics teaching C++ courses, current and former members of the C++ standards committee and industry experts using C++ to build industrial-strength, high-performance systems. They provided countless suggestions for improving the presentation. Any remaining flaws in the book are our own.

Tenth Edition reviewers: Chris Aburime, Minnesota State Colleges and Universities System; Gašper Ažman, A9.com Search Technologies and Co-Author of *C++ Today: The Beast is Back*; Danny Kalev, Intel and Former Member of the C++ Standards Committee; Renato Golin, LLVM Tech Lead at Linaro and Co-Owner for the ARM Target in LLVM; Gordon Hogenson, Microsoft, Author of *Foundations of C++/CLI: The Visual C++ Language for .NET 3*; Jonathan Wakely, Redhat, ISO C++ Committee Secretary; José Antonio González Seco, Parliament of Andalusia; Dean Michael Berris, Google, Maintainer of `cpp-netlib` and Former ISO C++ Committee Member.

Ninth Edition post-publication academic reviewers: Stefano Basagni, Northeastern University; Amr Elkady, Diablo Valley College; Chris Aburime, Minnesota State Colleges and Universities System.

Other recent edition reviewers: Virginia Bailey (Jackson State University), Ed James-Beckham (Borland), Thomas J. Borrelli (Rochester Institute of Technology), Ed Brey (Kohler Co.), Chris Cox (Adobe Systems), Gregory Dai (eBay), Peter J. DePasquale (The College of New Jersey), John Dibling (SpryWare), Susan Gauch (University of Arkansas), Doug Gregor (Apple, Inc.), Jack Hagemester (Washington State University), Williams M. Higdon (University of Indiana), Anne B. Horton (Lockheed Martin), Terrell Hull (Logicalis Integration Solutions), Linda M. Krause (Elmhurst College), Wing-Ning Li (University of Arkansas), Dean Mathias (Utah State University), Robert A. McLain (Tide-water Community College), James P. McNellis (Microsoft Corporation), Robert Myers (Florida State University), Gavin Osborne (Saskatchewan Institute of Applied Science and Technology), Amar Raheja (California State Polytechnic University, Pomona), April Reagan (Microsoft), Robert C. Seacord (Secure Coding Manager at SEI/CERT, author of *Secure Coding in C and C++*), Raymond Stephenson (Microsoft), Dave Topham (Ohlone College), Anthony Williams (author and C++ Standards Committee member) and Chad Willwerth (University Washington, Tacoma).

As you read the book, we'd sincerely appreciate your comments, criticisms and suggestions for improving the text. Please address all correspondence to:

`deitel@deitel.com`

We'll respond promptly. We enjoyed writing *C++ How to Program, Tenth Edition*. We hope you enjoy reading it!

Paul Deitel
Harvey Deitel

Acknowledgments for the Global Edition

Pearson would like to thank and acknowledge the following people for their contributions to the Global Edition.

Contributors

Rosanne Els, University of Kawazulu Natal; Almasi S. Maguya, Mzumbe University; Khyat Sharma.

Reviewers

T.V. Gopal, Anna University; Subrajeet Mohapatra, Birla Institute of Technology; Prabhat Verma, Harcourt Butler Technological Institute.

About the Authors

Paul Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., has over 30 years of experience in computing. He is a graduate of MIT, where he studied Information Technology. He holds the Java Certified Programmer and Java Certified Developer designations and is an Oracle Java Champion. Paul was also named as a Microsoft® Most Valuable Professional (MVP) for C# in 2012–2014. Through Deitel & Associates, Inc., he has delivered hundreds of programming courses worldwide to clients, including Cisco, IBM, Siemens, Sun Microsystems, Dell, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey Deitel, are the world's best-selling programming-language textbook/professional book/video authors.

Dr. Harvey Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has over 50 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University—he studied computing in each of these programs before they spun off Computer Science programs. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., in 1991 with his son, Paul. The Deitels' publications have earned international recognition, with translations published in Japanese, German, Russian, Spanish, French, Polish, Italian, Simplified Chinese, Traditional Chinese, Korean, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to academic, corporate, government and military clients.

About Deitel & Associates, Inc.

Deitel & Associates, Inc., founded by Paul Deitel and Harvey Deitel, is an internationally recognized authoring and corporate training organization, specializing in computer programming languages, object technology, Internet and web software technology, and Android and iOS app development. The company's clients include academic institutions, many of the world's largest corporations, government agencies and branches of the military. The company offers instructor-led training courses delivered at client sites worldwide on major programming languages and platforms, including C++, C, Java™, Android app development, iOS app development, Swift™, Visual C#®, Visual Basic®, Internet and web programming and a growing list of additional programming and software-development courses.



Before You Begin

This section contains information you should review before using this book and instructions to ensure that your computer is set up properly to compile the example programs.

Font and Naming Conventions

We use fonts to distinguish between features, such as menu names, menu items, and other elements that appear in your IDE (Integrated Development Environment), such as Microsoft's Visual Studio. Our convention is to emphasize IDE features in a sans-serif bold **Helvetica** font (for example, **File** menu) and to emphasize program text in a sans-serif **Lucida** font (for example, `bool x = true;`).

Obtaining the Software Used in *C++ How to Program, 10/e*

Before reading this book, you should download and install a C++ compiler. We wrote *C++ How to Program, 10/e*'s code examples using the following free C++ development tools:

- Microsoft's free Visual Studio Community 2015 edition, which includes the Visual C++ compiler and other Microsoft development tools. This runs on Windows and is available for download at

<https://www.visualstudio.com/products/visual-studio-community-vs>

- GNU's free GNU C++ 5.2.1 compiler. GNU C++ is already installed on most Linux systems and also can be installed on Mac OS X and Windows systems. There are many versions of Linux, known as Linux distributions, that use different techniques for performing software upgrades. Check your distribution's online documentation for information on how to upgrade GNU C++ to the latest version. GNU C++ is available at

<http://gcc.gnu.org/install/binaries.html>

- Apple's free Xcode, which OS X users can download from the Mac App Store—click the app's icon in the dock at the bottom of your Mac screen, then search for Xcode in the app store.

We also provide links to our getting-started videos for each of these C++ tools at:

<http://www.deitel.com/books/cpphttp10>

Obtaining the Code Examples

The examples for *C++ How to Program, 10/e* are available for download at

<http://www.pearsonglobal editions.com/deitel>

Click the **Download Code Examples** link to download the ZIP archive file to your computer. Write down the location where you saved the file—most browsers will save the file into your user account's **Downloads** folder.

Throughout the book, steps that require you to access our example code on your computer assume that you've extracted the examples from the ZIP file and placed them in `C:\examples` on Windows or in your user account's Documents directory on other platforms. You can extract them anywhere you like, but if you choose a different location, you'll need to update our steps accordingly.

Creating Projects

In Section 1.10, we demonstrate how to compile and run programs with

- Microsoft Visual Studio Community 2015 edition on Windows (Section 1.10.1)
- GNU C++ 5.2.1 on Linux (Section 1.10.2)
- Apple Xcode on OS X (Section 1.10.3)

For GNU C++ and Xcode, you must compile your programs with C++14. To do so in GNU C++, include the option `-std=c++14` when you compile your code, as in:

```
g++ -std=c++14 YourFileName.cpp
```

For Xcode, after following Section 1.10.3's steps to create a project:

1. Select the root node at the top of the Xcode **Project** navigator.
2. Click the **Build Settings** tab in the **Editors** area.
3. Scroll down to the **Apple LLVM 7.0 - Language - C++** section.
4. For the **C++ Language Dialect** option, select **C++14 [-std=c++14]**.

Getting Your C++ Questions Answered

As you read the book, if you have questions, we're easy to reach at

```
deitel@deitel.com
```

We'll respond promptly.

In addition, the web is loaded with programming information. An invaluable resource for nonprogrammers and programmers alike is the website

```
http://stackoverflow.com
```

on which you can:

- Search for answers to most common programming questions
- Search for error messages to see what causes them
- Ask programming questions to get answers from programmers worldwide
- Gain valuable insights about programming in general

Online C++ Documentation

For documentation on the C++ Standard Library, visit

```
http://cppreference.com
```

and be sure to check out the C++ FAQ at

```
https://isocpp.org/faq
```

Introduction to Computers and C++

1



Objectives

In this chapter you'll learn:

- Exciting recent developments in the computer field.
- Computer hardware, software and networking basics.
- The data hierarchy.
- The different types of programming languages.
- Basic object-technology concepts.
- Some basics of the Internet and the World Wide Web.
- A typical C++ program development environment.
- To test-drive a C++ application.
- Some key recent software technologies.
- How computers can help you make a difference.

- 1.1 Introduction
- 1.2 Computers and the Internet in Industry and Research
- 1.3 Hardware and Software
 - 1.3.1 Moore's Law
 - 1.3.2 Computer Organization
- 1.4 Data Hierarchy
- 1.5 Machine Languages, Assembly Languages and High-Level Languages
- 1.6 C and C++
- 1.7 Programming Languages
- 1.8 Introduction to Object Technology
- 1.9 Typical C++ Development Environment
- 1.10 Test-Driving a C++ Application
 - 1.10.1 Compiling and Running an Application in Visual Studio 2015 for Windows
 - 1.10.2 Compiling and Running Using GNU C++ on Linux
 - 1.10.3 Compiling and Running with Xcode on Mac OS X
- 1.11 Operating Systems
 - 1.11.1 Windows—A Proprietary Operating System
 - 1.11.2 Linux—An Open-Source Operating System
 - 1.11.3 Apple's OS X; Apple's iOS for iPhone®, iPad® and iPod Touch® Devices
 - 1.11.4 Google's Android
- 1.12 The Internet and the World Wide Web
- 1.13 Some Key Software Development Terminology
- 1.14 C++11 and C++14: The Latest C++ Versions
- 1.15 Boost C++ Libraries
- 1.16 Keeping Up to Date with Information Technologies

Self-Review Exercises | *Answers to Self-Review Exercises* | *Exercises* | *Making a Difference* | *Making a Difference Resources*

1.1 Introduction

Welcome to 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're already familiar with the powerful tasks computers perform. Using this textbook, 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* that model *things* in the real world.

C++ is one of today's most popular software development languages. This text provides an introduction to programming in C++11 and C++14—the latest versions standardized through the **International Organization for Standardization (ISO)** and the **International Electrotechnical Commission (IEC)**.

As of 2008 there were more than a billion general-purpose computers in use. Today, various websites say that number is approximately two billion, and according to the real-time tracker at gsmaintelligence.com, there are now more mobile devices than there are people in the world. According to the International Data Corporation (IDC), the number of mobile Internet users will top two billion in 2016.¹ Smartphone sales surpassed personal computer sales in 2011.² Tablet sales were expected to overtake personal-computer sales

11
14

1. <https://www.idc.com/getdoc.jsp?containerId=prUS40855515>.

2. <http://www.mashable.com/2012/02/03/smartphone-sales-overtake-pcs/>.

by 2015.³ By 2017, the smartphone/tablet app market is expected to exceed \$77 billion.⁴ This explosive growth is creating significant opportunities for programming mobile applications.

1.2 Computers and the Internet in Industry and Research

These are exciting times in the computer field. 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. At the time of this writing, Apple was the most valuable company in the world. Figure 1.1 provides a few examples of the ways in which computers are improving people's lives in research, industry and society.

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.
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.

Fig. 1.1 | A few uses for computers. (Part 1 of 3.)

3. <http://www.forbes.com/sites/louiscolombus/2014/07/18/gartner-forecasts-tablet-shipments-will-overtake-pcs-in-2015/>.

4. <http://www.entrepreneur.com/article/236832>.

Name	Description
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.
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.
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 2 of 3.)

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 <i>Fallout 4</i> 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 3 of 3.)

1.3 Hardware and Software

Computers can perform calculations and make logical decisions phenomenally faster than human beings can. Many of 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! China’s National University of Defense Technology’s Tianhe-2 supercomputer can perform over 33 quadrillion calculations per second (33.86 *petaflops*)!⁵ To put that in perspective, *the Tianhe-2 supercomputer can perform in one second about 3 million calculations for every person on the planet!* And supercomputing “upper limits” are growing quickly.

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**. In this book, you’ll learn a key programming methodology that’s enhancing programmer productivity, thereby reducing software development costs—*object-oriented programming*.

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*, owing 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. Ironically, silicon is one of the most abundant materials on Earth—it’s an ingredient in common sand. Silicon-chip technology has made computing so economical that computers have become a commodity.

1.3.1 Moore’s Law

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

5. <http://www.top500.org>.

Every year or two, the capacities of computers have approximately *doubled* inexpensively. This remarkable trend often is called **Moore’s Law**, named for the person who identified it in the 1960s, Gordon Moore, co-founder of Intel—a leading manufacturer of the processors in today’s computers and embedded systems. 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, which puts greater demands on programming-language designers to innovate.

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

1.3.2 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 <i>accelerometer</i> (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).
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 1 of 2.)

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 <i>volatile</i> —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 <i>calculations</i> , such as addition, subtraction, multiplication and division. It also contains the <i>decision</i> 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.
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 <i>dual-core processor</i> has two CPUs, a <i>quad-core processor</i> has four and an <i>octa-core processor</i> 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.
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 <i>hard drive</i>) until they’re again needed, possibly hours, days, months or even years later. Information on secondary storage devices is <i>persistent</i> —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 2 of 2.)

1.4 Data Hierarchy

Data items processed by computers form a **data hierarchy** that becomes larger and more complex in structure as we progress from the simplest data items (called “bits”) to richer ones, such as characters and fields. Figure 1.3 illustrates a portion of the data hierarchy.

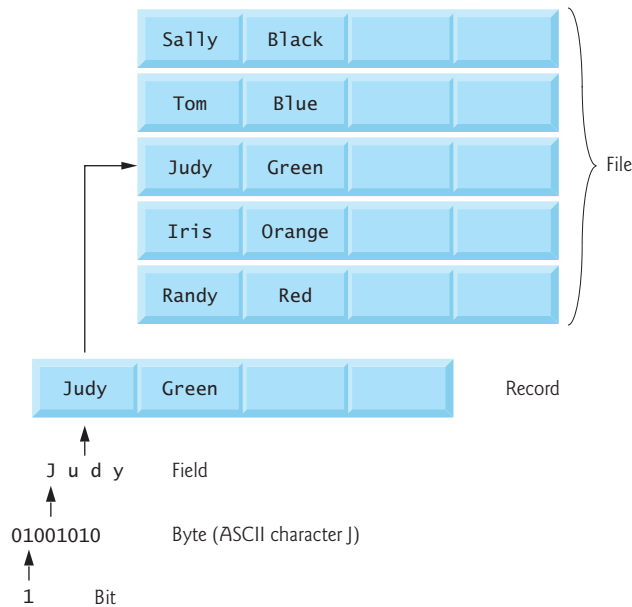


Fig. 1.3 | Data hierarchy.

Bits

The smallest data item in a computer can assume the value 0 or the value 1. It's called a **bit** (short for “binary digit”—a digit that can assume one of *two* values). Remarkably, the impressive functions performed by computers involve only the simplest manipulations of 0s and 1s—*examining a bit's value*, *setting a bit's value* and *reversing a bit's value* (from 1 to 0 or from 0 to 1).

Characters

It's tedious for people to work with data in the low-level form of bits. Instead, they prefer to work with *decimal digits* (0–9), *letters* (A–Z and a–z), and *special symbols* (e.g., \$, @, %, &, *, (,), -, +, ", :, ? and /). Digits, letters and special symbols are known as **characters**. The computer's **character set** is the set of all the characters used to write programs and represent data items. Computers process only 1s and 0s, so a computer's character set represents every character as a pattern of 1s and 0s. C++ supports various character sets (including **Unicode**[®]), with some requiring more than one byte per character. Unicode supports many of the world's languages. See Appendix B for more information on the **ASCII (American Standard Code for Information Interchange)** character set—the popular subset of Unicode that represents uppercase and lowercase letters, digits and some common special characters.

Fields

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

Records

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

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

Thus, a record is a group of related fields. In the preceding example, all the fields belong to the *same* employee. A company might have many employees and a payroll record for each.

Files

A **file** is a group of related records. [Note: More generally, a file contains arbitrary data in arbitrary formats. In some operating systems, a file is viewed simply as a *sequence of bytes*—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer.] It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information.

Database

A **database** is a collection of data organized for easy access and manipulation. The most popular model is the *relational database*, in which data is stored in simple *tables*. A table includes *records* and *fields*. For example, a table of students might include first name, last name, major, year, student ID number and grade-point-average fields. The data for each student is a record, and the individual pieces of information in each record are the fields. You can *search*, *sort* and otherwise manipulate the data based on its relationship to multiple tables or databases. For example, a university might use data from the student database in combination with data from databases of courses, on-campus housing, meal plans, etc.

Big Data

The amount of data being produced worldwide is enormous and growing quickly. According to IBM, approximately 2.5 quintillion bytes (2.5 *exabytes*) of data are created daily⁶ and according to Salesforce.com, 90% of the world's data was created in just the past 12 months!⁷ According to an IDC study, the global data supply will reach 40 *zettabytes* (equal to 40 trillion gigabytes) annually by 2020.⁸ Figure 1.4 shows some common byte measurements. **Big data** applications deal with massive amounts of data and this field is growing quickly, creating lots of opportunity for software developers. According to a study by Gartner Group, over 4 million IT jobs globally were expected to support big data in 2015.⁹

6. <http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html>.

7. <https://www.salesforce.com/blog/2015/10/salesforce-channel-ifttt.html>.

8. <http://recode.net/2014/01/10/stuffed-why-data-storage-is-hot-again-really/>.

9. <http://tech.fortune.cnn.com/2013/09/04/big-data-employment-boom/>.