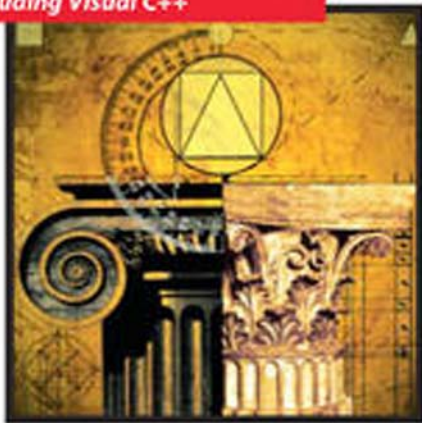


C++

from the **GROUND UP**

Works with all C++ compilers,
including Visual C++



THIRD EDITION

Covers the International
Standard for C++

Teaches the entire C++
language, from the basics
to advanced features

Packed with insider tips
and techniques, and
hundreds of examples

Includes the Standard
Template
Library (STL)

FREE
CODE
ONLINE

BONUS!
www.osborne.com

Herbert Schildt

Best-selling programming author with more than 3 million books sold!

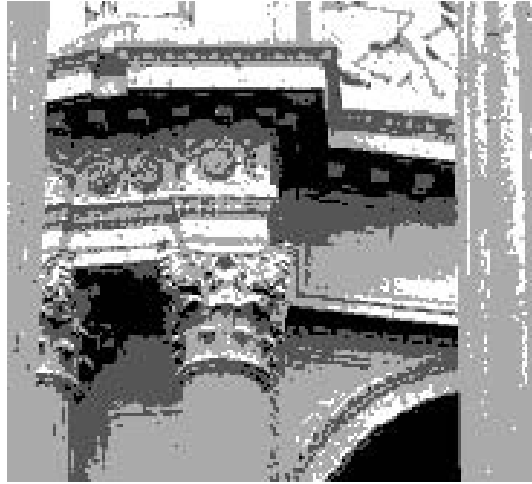


C++ from the Ground Up

Third Edition

About the Author

Herbert Schildt is the world's leading programming author. He is an authority on the C, C++, Java, and C# languages, and is a master Windows programmer. His programming books have sold more than 3 million copies worldwide and have been translated into all major foreign languages. He is the author of numerous bestsellers, including *C++: The Complete Reference*, *C#: The Complete Reference*, *Java 2: The Complete Reference*, *C: The Complete Reference*, *C++ From the Ground Up*, *C++: A Beginner's Guide*, *C#: A Beginner's Guide*, and *Java 2: A Beginner's Guide*. Schildt holds a master's degree in computer science from the University of Illinois. He can be reached at his consulting office at (217) 586-4683.



C++ from the Ground Up

Third Edition

Herbert Schildt

McGraw-Hill/Osborne

New York Chicago San Francisco
Lisbon London Madrid Mexico City
Milan New Delhi San Juan
Seoul Singapore Sydney Toronto

McGraw-Hill/Osborne
2600 Tenth Street
Berkeley, California 94710
U.S.A.

To arrange bulk purchase discounts for sales promotions, premiums, or fund-raisers, please contact **McGraw-Hill/Osborne** at the above address. For information on translations or book distributors outside the U.S.A., please see the International Contact Information page immediately following the index of this book.

C++ from the Ground Up, Third Edition

Copyright © 2003 by The McGraw-Hill Companies. All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

1234567890 DOC DOC 019876543

ISBN 0-07-222897-0

Publisher

Brandon A. Nordin

**Vice President &
Associate Publisher**

Scott Rogers

Acquisitions Editor

Lisa McClain

Project Editors

Jenn Tust, Elizabeth Seymour

Proofreader

Marian M. Selig

Indexer

Sheryl Schildt

Computer Designers

Tabitha M. Cagan, Tara A. Davis,
John Patrus, Lucie Ericksen

Illustrators

Michael Mueller, Lyssa Wald,
Melinda Lytle

Cover Series Design

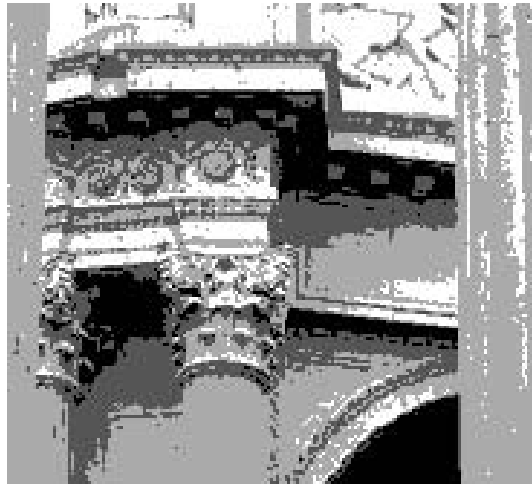
John Nedwidek, emdesign

Cover Illustration

Lance Ravella

This book was composed with Corel VENTURA™ Publisher.

Information has been obtained by **McGraw-Hill/Osborne** from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, **McGraw-Hill/Osborne**, or others, **McGraw-Hill/Osborne** does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.



Contents

<i>Preface</i>	<i>xvii</i>
1 ■ The Story of C++	1
The Origins of C++	2
The Creation of C	2
Understanding the Need for C++	4
C++ Is Born	5
The Evolution of C++	6
What Is Object-Oriented Programming?	6
Encapsulation	7
Polymorphism	7
Inheritance	8
C++ Implements OOP	8
How C++ Relates to Java and C#	8
2 ■ An Overview of C++	11
Your First C++ Program	12
Entering the Program	12
Compiling the Program	13
Run the Program	14
A Line-by-Line Explanation	14

Handling Syntax Errors	16
A Second Simple Program	17
A More Practical Example	18
A New Data Type	19
A Quick Review	20
Functions	20
A Program with Two Functions	21
Function Arguments	22
Functions Returning Values	24
The main() Function	25
The General Form of C++ Functions	26
Some Output Options	26
Two Simple Commands	27
The if Statement	27
The for Loop	28
Blocks of Code	29
Semicolons and Positioning	30
Indentation Practices	31
C++ Keywords	31
Identifiers in C++	32
The Standard C++ Library	32
3 ■ The Basic Data Types	33
Declaration of Variables	35
Local Variables	35
Formal Parameters	36
Global Variables	37
Some Type Modifiers	38
Literals	41
Hexadecimal and Octal Literals	43
String Literals	43
Character Escape Sequences	44
Variable Initializations	45
Operators	46
Arithmetic Operators	46
Increment and Decrement	48
How C++ Got Its Name	49
Relational and Logical Operators	50
Expressions	53
Type Conversion in Expressions	53
Converting to and from bool	53
Casts	54
Spacing and Parentheses	55

4	Program Control Statements	57
	The if Statement	58
	The Conditional Expression	59
	Nested ifs	60
	The if-else-if Ladder	61
	The for Loop	62
	Some Variations on the for Loop	64
	Missing Pieces	66
	The Infinite Loop	66
	Time Delay Loops	67
	The switch Statement	67
	Nested switch Statements	71
	The while Loop	71
	The do-while Loop	73
	Using continue	74
	Using break to Exit Loops	75
	Nested Loops	76
	Using the goto Statement	77
	Putting Together the Pieces	78
5	Arrays and Strings	81
	One-Dimensional Arrays	82
	No Bounds Checking	84
	Sorting an Array	85
	Strings	86
	Reading a String from the Keyboard	87
	Some String Library Functions	89
	strcpy	89
	strcat	89
	strcmp	90
	strlen	91
	Using the Null Terminator	93
	Two-Dimensional Arrays	94
	Multidimensional Arrays	96
	Array Initialization	96
	Unsize Array Initializations	100
	Arrays of Strings	101
	An Example Using String Arrays	102
6	Pointers	105
	What Are Pointers?	106
	The Pointer Operators	107
	The Base Type Is Important	108
	Assigning Values Through a Pointer	110

Pointer Expressions	110
Pointer Arithmetic	111
Pointer Comparisons	112
Pointers and Arrays	112
Indexing a Pointer	115
Are Pointers and Arrays Interchangeable?	116
Pointers and String Literals	117
A Comparison Example	117
Arrays of Pointers	118
The Null Pointer Convention	121
Multiple Indirection	122
Pointers and 16-bit Environments	122
Problems with Pointers	124
Uninitialized Pointers	124
Invalid Pointer Comparisons	124
Forgetting to Reset a Pointer	125
7 ■ Functions, Part One: The Fundamentals	127
Scope Rules of Functions	128
Local Variables	128
Formal Parameters	134
Global Variables	134
Passing Pointers and Arrays	136
Calling Functions with Pointers	136
Calling Functions with Arrays	137
Passing Strings	140
argc and argv: Arguments to main()	141
Passing Numeric Command Line Arguments	144
Converting Numeric Strings to Numbers	145
The return Statement	145
Returning from a Function	146
Returning Values	147
void Functions	149
Functions That Return Pointers	149
Function Prototypes	151
Headers: A Closer Look	152
Old-Style versus Modern Function Parameter Declarations	153
Recursion	153
8 ■ Functions, Part Two: References, Overloading, and Default Arguments	157
Two Approaches to Argument Passing	158
How C++ Passes Arguments	158
Using a Pointer to Create a Call-by-Reference	159

Reference Parameters	160
Declaring Reference Parameters	163
Returning References	164
Creating a Bounded Array	167
Independent References	168
A Few Restrictions When Using References	169
Function Overloading	170
The overload Anachronism	173
Default Function Arguments	173
Default Arguments versus Overloading	175
Using Default Arguments Correctly	177
Function Overloading and Ambiguity	177
9 More Data Types and Operators	181
The const and volatile Qualifiers	182
const	182
volatile	184
Storage Class Specifiers	185
auto	185
extern	186
static Variables	187
Register Variables	191
The Origins of the register Modifier	192
Enumerations	193
typedef	197
More Operators	197
Bitwise Operators	197
AND, OR, XOR, and NOT	198
The Shift Operators	202
The ? Operator	203
Compound Assignment	205
The Comma Operator	205
Multiple Assignments	206
Using sizeof	206
Dynamic Allocation Using new and delete	207
Initializing Dynamically Allocated Memory	210
Allocating Arrays	210
C's Approach to Dynamic Allocation: malloc() and free()	211
Precedence Summary	213
10 Structures and Unions	215
Structures	216
Accessing Structure Members	218
Arrays of Structures	219

A Simple Inventory Example	219
Passing Structures to Functions	226
Assigning Structures	227
Pointers to Structures and the Arrow Operator	228
References to Structures	232
Arrays and Structures Within Structures	233
C Structure Versus C++ Structures	234
Bit-Fields	235
Unions	237
Anonymous Unions	242
Using sizeof to Ensure Portability	243
Moving On to Object-Oriented Programming	243
11 ■ Introducing the Class	245
Class Fundamentals	246
The General Form of a class	250
A Closer Look at Class Member Access	250
Constructors and Destructors	252
Parameterized Constructors	255
An Initialization Alternative	259
Classes and Structures Are Related	260
Structures versus Classes	262
Unions and Classes Are Related	263
Inline Functions	264
Creating Inline Functions Inside a Class	265
Arrays of Objects	267
Initializing Object Arrays	268
Pointers to Objects	270
Object References	272
12 ■ A Closer Look at Classes	273
Friend Functions	274
Overloading Constructors	278
Dynamic Initialization	280
Applying Dynamic Initialization to Constructors	280
Assigning Objects	282
Passing Objects to Functions	283
Constructors, Destructors, and Passing Objects	284
A Potential Problem When Passing Objects	285
Returning Objects	288
A Potential Problem When Returning Objects	289
Creating and Using a Copy Constructor	291
Copy Constructors and Parameters	292
Copy Constructors and Initializations	294

Using Copy Constructors When an Object Is Returned . . .	295
Copy Constructors—Is There a Simpler Way?	296
The this Keyword	297
13 ■ Operator Overloading	299
Operator Overloading Using Member Functions	300
Using Member Functions to Overload Unary Operators . .	303
Operator Overloading Tips and Restrictions	308
Nonmember Operator Functions	309
Order Matters	309
Using a Friend to Overload a Unary Operator	313
Overloading the Relational and Logical Operators	316
A Closer Look at the Assignment Operator	317
Overloading []	320
Overloading ()	324
Overloading Other Operators	325
Another Example of Operator Overloading	325
14 ■ Inheritance	331
Introducing Inheritance	332
Base Class Access Control	335
Using protected Members	337
Using protected for Inheritance of a Base Class	340
Reviewing public, protected, and private	342
Inheriting Multiple Base Classes	342
Constructors, Destructors, and Inheritance	343
When Constructors and Destructors Are Executed	343
Passing Parameters to Base Class Constructors	346
Granting Access	350
Reading C++ Inheritance Graphs	352
Virtual Base Classes	352
15 ■ Virtual Functions and Polymorphism	357
Pointers to Derived Types	358
References to Derived Types	360
Virtual Functions	360
Virtual Functions Are Inherited	363
Why Virtual Functions?	365
A Simple Application of Virtual Functions	366
Pure Virtual Functions and Abstract Classes	370
Early versus Late Binding	372
Polymorphism and the Purist	373
16 ■ Templates	375
Generic Functions	376
A Function with Two Generic Types	378
Explicitly Overloading a Generic Function	379

Overloading a Function Template	381
Using Standard Parameters with Template Functions ...	382
Generic Function Restrictions	383
Creating a Generic abs() Function	383
Generic Classes	384
An Example with Two Generic Data Types	387
Creating a Generic Array Class	388
Using Non-Type Arguments with Generic Classes	389
Using Default Arguments with Template Classes	391
Explicit Class Specializations	393
17 ■ Exception Handling	395
Exception Handling Fundamentals	396
exit() and abort()	398
Catching Class Types	401
Using Multiple catch Statements	402
Options for Exception Handling	404
Catching All Exceptions	404
Restricting Exceptions Thrown by a Function	406
Rethrowing an Exception	408
Handling Exceptions Thrown by new	409
The nothrow Alternative	410
Overloading new and delete	411
Overloading the nothrow Version of new	415
18 ■ The C++ I/O System	417
Old VS Modern C++ I/O	418
C++ Streams	418
The C++ Predefined Streams	419
The C++ Stream Classes	419
Overloading the I/O Operators	420
Creating Inserters	421
Using Friend Functions to Overload Inserters	423
Overloading Extractors	424
C I/O Versus C++ I/O	426
Formatted I/O	426
Formatting with the ios Member Functions	426
Using I/O Manipulators	431
Creating Your Own Manipulator Functions	433
File I/O	435
Opening and Closing a File	435
Reading and Writing Text Files	438
Unformatted Binary I/O	439
Reading and Writing Blocks of Data	441
Detecting EOF	442
A File Comparison Example	443

More Binary I/O Functions	444
Random Access	446
Checking I/O Status	448
Customized I/O and Files	449
19 ■ Run-Time Type ID and the Casting Operators	451
Run-Time Type Identification (RTTI)	452
A Simple Application of Run-Time Type ID	456
typeid Can Be Applied to Template Classes	458
The Casting Operators	462
dynamic_cast	462
const_cast	467
static_cast	468
reinterpret_cast	469
The Traditional Cast Versus the Four Casting Operators ..	470
20 ■ Namespaces and Other Advanced Topics	471
Namespaces	472
Namespace Fundamentals	472
using	475
Unnamed Namespaces	477
The std Namespace	478
Pointers to Functions	480
Finding the Address of an Overloaded Function	483
Static Class Members	484
const Member Functions and mutable	486
Explicit Constructors	488
An Interesting Benefit from Implicit Constructor Conversion	490
The Member Initialization Syntax	490
Using the asm Keyword	493
Linkage Specification	493
The .* and ->* Pointer-to-Member Operators	495
Creating Conversion Functions	497
21 ■ Introducing the Standard Template Library	499
An Overview of the STL	500
The Container Classes	502
Vectors	504
Accessing a Vector Through an Iterator	508
Inserting and Deleting Elements in a Vector	509
Storing Class Objects in a Vector	510
The Power of Iterators	513
Lists	514
Sort a List	519
Merging One List with Another	520
Storing Class Objects in a List	521

Maps	523
Storing Class Objects in a Map	528
Algorithms	529
Counting	532
Removing and Replacing Elements	533
Reversing a Sequence	535
Transforming a Sequence	535
Exploring the Algorithms	537
The string Class	537
Some string Member Functions	541
Putting Strings into Other Containers	545
Final Thoughts on the STL	545
22 ■ The C++ Preprocessor	547
#define	548
Function-Like Macros	550
#error	552
#include	552
Conditional Compilation Directives	553
#if, #else, #elif, and #endif	553
#ifdef and #ifndef	555
#undef	556
Using defined	557
The Diminishing Role of the Preprocessor	557
#line	558
#pragma	559
The # and ## Preprocessor Operators	559
Predefined Macro Names	560
Final Thoughts	561
A ■ C-Based I/O	563
C I/O Uses Streams	564
Understanding printf() and scanf()	565
printf()	565
scanf()	567
The C File System	572
fopen()	573
fputc()	574
fgetc()	574
feof()	575
fclose()	575
Using fopen(), fgetc(), fputc(), and fclose()	575
ferror() and rewind()	576
fread() and fwrite()	577

fseek() and Random-Access I/O	578
fprintf() and fscanf()	579
Erasing Files	580
B ■ Working with an Older C++ Compiler	581
Two Simple Changes	583
C ■ The .NET Managed Extensions to C++	585
The .NET Keyword Extensions	586
__abstract	586
__box	587
__delegate	587
__event	587
__finally	587
__gc	587
__identifier	587
__interface	587
__nogc	587
__pin	588
__property	588
__sealed	588
__try_cast	588
__typeof	588
__value	588
Preprocessor Extensions	588
The attribute Attribute	589
Compiling Managed C++	589
■ Index	591

This page intentionally left blank





Preface

This book teaches you how to program in C++ — the most powerful computer language in use today. No previous programming experience is required. The book starts with the basics, covers the fundamentals, moves on to the core of the language, and concludes with its more advanced features. By the time you finish, you will be an accomplished C++ programmer.

C++ is your gateway to modern, object-oriented programming. It is the preeminent language for the development of high-performance software and is the choice of programmers worldwide. Simply put, to be a top-flight, professional programmer today implies competency in C++.

C++ is more than just a popular language. C++ provides the conceptual substrata that underlie the design of several other languages, and much of modern computing. It is no accident that two other important languages, Java and C#, are descended from C++. There is little in programming that has not been influenced by the syntax, style, and philosophy of C++.

Because C++ was designed for professional programming, C++ is not the easiest programming language to learn. It is, however, the *best* programming language to learn. Once you have mastered C++, you will be able to write professional-quality, high-performance programs. You will also be able to easily learn languages like Java or C# because they share the same basic syntax and design as C++.

What Is New in the Third Edition

In the time that has passed since the previous edition of this book, there have been no changes to the C++ language. There have, however, been big changes to the computing environment. For example, Java became the dominant language for Web programming, the .NET Framework was released, and C# was invented. Through all the changes of the past few years, one thing has remained constant: the staying

power of C++. C++ has been, is, and will remain the dominant language of “power programmers” well into the foreseeable future.

The overall structure and organization of the third edition is similar to the second edition. Most of the changes involve updating and expanding the coverage throughout. In some cases, additional details were added. In other cases, the presentation of a topic was improved. In still other situations, descriptions were modernized to reflect the current programming environment. Several new sections were also added.

Two appendices were added. One describes the extended keywords defined by Microsoft that are used for creating managed code for the .NET Framework. The second explains how to adapt the code in this book for use with an older, non-standard C++ compiler.

Finally, all code examples were retested against the current crop of compilers, including Microsoft’s Visual Studio .NET and Borland’s C++ Builder.

What Version of C++

The material in this book describes Standard C++. This is the version of C++ defined by the ANSI/ISO Standard for C++, and it is the one that is currently supported by all major compilers. Therefore, using this book, you can be confident that what you learn today will also apply tomorrow.

How to Use This Book

The best way to learn any programming language, including C++, is by doing. Therefore, after you have read through a section, try the sample programs. Make sure that you understand why they do what they do before moving on. You should also experiment with the programs, changing one or two lines at a time and observing the results. The more you program, the better you become at programming.

If You’re Using Windows

If your computer uses Windows and your goal is to write Windows-based programs, then you have chosen the right language to learn. C++ is completely at home with Windows programming. However, none of the programs in this book use the Windows graphical user interface (GUI). Instead, they are console-based programs that can be run under a Windows console session, such as that provided by the Command Prompt window. The reason for this is easy to understand: GUI-based Windows programs are, by their nature, large and complex. They also use many techniques not directly related to the C++ language. Thus, they are not well-suited for teaching a programming language. However, you can still use a Windows-based compiler to compile the programs in this book because the compiler will automatically create a console session in which to execute your program.

Once you have mastered C++, you will be able to apply your knowledge to Windows programming. In fact, Windows programming using C++ allows the use of class libraries such as MFC or the newer .NET Framework, which can greatly simplify the development of a Windows program.

Don’t Forget: Code on the Web

Remember, the source code for all of the programs in this book is available free of charge on the Web at <http://www.osborne.com>. Downloading this code prevents you from having to type in the examples.

For Further Study

C++*from the Ground Up* is your gateway to the Herb Schildt series of programming books. Here are some others that you will find of interest.

To learn more about C++, try

C++: The Complete Reference

C++: A Beginner's Guide

Teach Yourself C++

STL Programming From the Ground Up

C++ Programmer's Reference

To learn about Java programming, we recommend the following:

Java 2: A Beginner's Guide

Java 2: The Complete Reference

Java 2 Programmer's Reference

To learn about C#, Herb offers these books:

C#: A Beginner's Guide

C#: The Complete Reference

To learn about Windows programming we suggest the following Schildt books:

Windows 98 Programming From the Ground Up

Windows 2000 Programming From the Ground Up

MFC Programming From the Ground Up

The Windows Programming Annotated Archives

If you want to learn about the C language, which is the foundation of all modern programming, then the following titles will be of interest.

C: The Complete Reference

Teach Yourself C

**When you need solid answers, fast, turn to Herbert Schildt,
the recognized authority on programming.**

This page intentionally left blank





CHAPTER 1

The Story of C++

C++ is the single most important language that any programmer can learn. This is a strong statement, but it is not an exaggeration. C++ is the center of gravity around which all of modern programming revolves. Its syntax and design philosophy define the essence of object-oriented programming. Moreover, C++ charts the course for future language development. For example, both Java and C# are directly descended from C++. C++ is also the universal language of programming; it is the language in which programmers share ideas with one another. To be a professional programmer today implies competency in C++. It is that fundamental and that important. C++ is the gateway to all of modern programming.

Before beginning your study of C++, it is important for you to know how C++ fits into the historical context of computer languages. Understanding the forces that drove its creation, the design philosophy it represents, and the legacy that it inherits makes it easier to appreciate the many innovative and unique features of C++. With this in mind, this chapter presents a brief history of the C++ programming language, its origins, its relationship to its predecessor (C), its uses, and the programming philosophies that it supports. It also puts C++ into perspective relative to other programming languages.

The Origins of C++

The story of C++ begins with C. The reason for this is simple: C++ is built upon the foundation of C. In fact, C++ is a superset of C. (Indeed, all C++ compilers can also be used to compile C programs!) Specifically, C++ is an expanded and enhanced version of C that embodies the philosophy of object-oriented programming (which is described later in this chapter). C++ also includes several other improvements to the C language, including an extended set of library routines. However, much of the spirit and flavor of C++ is inherited directly from C. To fully understand and appreciate C++, you need to understand the “how and why” behind C.

The Creation of C

The C language shook the computer world. Its impact should not be underestimated because it fundamentally changed the way programming was approached and thought about. C is considered by many to be the first modern “programmer’s language.” Prior to the invention of C, computer languages were generally designed either as academic exercises or by bureaucratic committees. C is different. C was designed, implemented, and developed by real, working programmers, and it reflected the way they approached the job of programming. Its features were honed, tested, thought about, and rethought by the people who actually used the language. The result of this process was a language that programmers liked to use. Indeed, C quickly attracted many followers who had a near-religious zeal for it, and it found wide and rapid acceptance in the programmer community. In short, C is a language designed by and for programmers.

C was invented and first implemented by Dennis Ritchie on a DEC PDP-11 using the UNIX operating system. C is the result of a development process that started with an older language called BCPL, which was developed by Martin Richards. BCPL influenced a language called B, invented by Ken Thompson, which led to the development of C in the 1970s.

For many years, the de facto standard for C was the one supplied with the Unix operating system and described in *The C Programming Language*, by Brian Kernighan and Dennis Ritchie (Prentice-Hall, 1978). However, because no formal standard existed, there were discrepancies between different implementations of C. To alter this situation, a committee was established in the beginning of the summer of 1983 to work on the creation of an ANSI (American National Standards Institute) standard that would define—once and for all—the C language. The final version of the standard was adopted in December 1989, the first copies of which became available in early 1990. This version of C is commonly referred to as *C89*, and it is the foundation upon which C++ is built.



NOTE: The C standard was updated in 1999 and this version of C is usually referred to as *C99*. This version contains some new features, including a few borrowed from C++, but, overall, it is compatible with the original C89 standard. At the time of this writing, no widely available compiler supports C99 and it is still C89 that defines what is commonly thought of as the C language. Furthermore, it is C89 that is the basis for C++. It is possible that a future standard for C++ will include the features added by C99, but they are not part of C++ at this time.

It may seem hard to understand at first, but C is often called a “middle-level” computer language. As it is applied to C, middle-level does not have a negative connotation; it does not mean that C is less powerful, harder to use, or less developed than a “high-level” language, or that it is as difficult to use as assembly language. (*Assembly language*, or *assembler*, as it is often called, is simply a symbolic representation of the actual machine code that a computer can execute.) C is thought of as a middle-level language because it combines elements of high-level languages, such as Pascal, Modula-2, or Visual Basic, with the functionality of assembler.

From a theoretical point of view, a *high-level language* attempts to give the programmer everything he or she could possibly want, already built into the language. A *low-level language* provides nothing other than access to the actual machine instructions. A *middle-level language* gives the programmer a concise set of tools and allows the programmer to develop higher-level constructs on his or her own. A middle-level language offers the programmer built-in power, coupled with flexibility.

Being a middle-level language, C allows you to manipulate bits, bytes, and addresses—the basic elements with which a computer functions. Thus, C does not attempt to buffer the hardware of the machine from your program to any significant extent. For example, the size of an integer in C is directly related to the word size of the CPU. In most high-level languages there are built-in statements for reading and writing disk files. In C, all of these procedures are performed by calls to library routines and not by keywords defined by the language. This approach increases C’s flexibility.

C allows—indeed, needs—the programmer to define routines for performing high-level operations. These routines are called *functions*, and they are very important to the C language. In fact, functions are the building blocks of both C and C++. You can easily tailor a library of functions to perform various tasks that are used by your program. In this sense, you can personalize C to fit your needs.

- [read online El caso del martillo blanco pdf, azw \(kindle\), epub](#)
- [Teaching in the 21st Century: Adapting Writing Pedagogies to the College Curriculum here](#)
- [Chasing Clayoquot: A Wilderness Almanac pdf, azw \(kindle\), epub, doc, mobi](#)
- [download Valmiki's Daughter pdf](#)
- [click Bear Attacks: Their Causes and Avoidance \(Revised Edition\) pdf](#)
- [download Unraveling \(Unraveling Series, Book 1\)](#)

- <http://pittiger.com/lib/Happiness--A-Guide-to-Developing-Life-s-Most-Important-Skill.pdf>
- <http://patrickvincitore.com/?ebooks/Teaching-in-the-21st-Century--Adapting-Writing-Pedagogies-to-the-College-Curriculum.pdf>
- <http://aseasonedman.com/ebooks/Seed-of-Avraham--The-4000-Year-History-of-the-Jewish-Family.pdf>
- <http://www.celebritychat.in/?ebooks/Hemp-Bound--Dispatches-from-the-Front-Lines-of-the-Next-Agricultural-Revolution.pdf>
- <http://econtact.webschaefer.com/?books/Guide-to-the-Traditions--Mage--The-Ascension-.pdf>
- <http://academialanguagebar.com/?ebooks/The-Uses-and-Abuses-of-History.pdf>