



C++ for Lazy Programmers

Quick, Easy, and Fun C++ for Beginners

Will Briggs

Apress®

C++ for Lazy Programmers

**Quick, Easy, and Fun C++ for
Beginners**

Will Briggs

Apress®

C++ for Lazy Programmers: Quick, Easy, and Fun C++ for Beginners

Will Briggs

Lynchburg, VA, USA

ISBN-13 (pbk): 978-1-4842-5186-7

<https://doi.org/10.1007/978-1-4842-5187-4>

ISBN-13 (electronic): 978-1-4842-5187-4

Copyright © 2019 by Will Briggs

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr

Acquisitions Editor: Steve Anglin

Development Editor: Matthew Moodie

Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image by Raw Pixel (www.rawpixel.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail editorial@apress.com; for reprint, paperback, or audio rights, please email bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484251867. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*To my favorite C++ programmer and the love of my life;
To the little one who first inspired her to study at home so
she wouldn't go mommy-crazy;
And to the boy who's already programming.*

Table of Contents

About the Author	xvii
About the Technical Reviewer	xix
Acknowledgments	xxi
Introduction	xxiii
Chapter 1: Getting Started	1
A simple program.....	1
Spacing.....	4
Creating an SSDL project.....	6
...in Visual Studio.....	6
...with g++	17
How not to be miserable	19
Shapes and the functions that draw them.....	21
Antibugging	30
consts and colors.....	31
Text	35
sout, escape sequences, and fonts	35
SSDL_RenderText, SSDL_RenderTextCentered	39
Chapter 2: Images and Sound	43
Images and changing window characteristics	43
Antibugging	49
Multiple images together	51
Adding transparency with GIMP.....	52
Sound.....	57
Antibugging	60

TABLE OF CONTENTS

Chapter 3: Numbers.....	61
Variables	61
Constants	63
When to use constants, not literal values.....	64
Math operators.....	65
Integer division	65
Assignment (=) operators.....	66
A diving board example	67
The no-worries list for math operators.....	70
Built-in functions and casting	71
Antibugging	76
Chapter 4: Mouse, and if.....	79
Mouse functions	79
Antibugging	82
if.....	83
Coercion and if conditions (int's dirty little secret).....	86
Combining conditions with &&, , and !.....	86
Antibugging	87
Boolean values and variables	90
A hidden object game	92
Chapter 5: Loops, Input, and char.....	101
Keyboard input.....	101
Antibugging	103
while and do-while	105
Loops with SSDL.....	107
break and continue.....	108
Antibugging	109
for loops	112
Increment operators	113
An example: averaging numbers	114
Antibugging	116

chars and ctype.....	118
switch	123
Antibugging	124
Chapter 6: Algorithms and the Development Process	127
Adventures in robotic cooking	127
Writing a program, from start to finish	131
Requirements: What do we want to do?	131
Algorithm: How do we do it?	132
Walkthrough: Will it do it?	134
Coding: putting it all into C++ (plus: commenting the lazy way)	134
Chapter 7: Functions	141
Functions that return values	141
Functions that return nothing	148
Global variables.....	152
Antibugging	153
How to write a function in four easy steps (and call it in one).....	156
Why have functions, anyway?	160
Chapter 8: Functions (Continued)	171
Random numbers.....	171
Making a random number generator.....	171
Using the built-in random number generator	174
Antibugging	177
Boolean functions	179
& parameters	180
Antibugging	185
Identifier scope	186
A final note on algorithms.....	189

TABLE OF CONTENTS

Chapter 9: Using the Debugger 191

 A flawed program 191

 Breakpoints and watched variables..... 196

 Visual Studio..... 196

 ddd 198

 gdb 199

 Fixing the stripes 199

 Going into functions 199

 Visual Studio..... 199

 ddd 201

 gdb 202

 Fixing the stars 202

 Wrap-up 203

 Antibugging 203

 Bottom-up testing 204

 More on antibugging..... 205

Chapter 10: Arrays and enum..... 207

 Arrays..... 207

 Arrays’ dirty little secret: using memory addresses 210

 Antibugging 211

 Arrays as function parameters..... 212

 Array parameters that change, or don’t..... 213

 Array parameters and reusability 213

 Antibugging 214

 Enumeration types 215

 enum class 217

 Antibugging 218

 Multidimensional arrays 219

 Displaying the board..... 220

 Arrays of more than two dimensions..... 223

 Antibugging 224

Chapter 11: Animation with structs and Sprites	227
structs.....	227
Cool struct tricks	231
Making a movie with struct and while.....	232
Sprites.....	239
Antibugging	244
Chapter 12: Making an Arcade Game: Input, Collisions, and Putting It All Together	245
Determining input states	245
Mouse.....	245
Keyboard	246
Antibugging	248
Events	248
Cooldowns and lifetimes.....	251
Collisions.....	254
The big game	255
Antibugging	269
Chapter 13: Standard I/O and File Operations	273
Standard I/O programs.....	273
Compiling in Microsoft Visual Studio	274
Compiling with g++.....	278
File I/O (optional).....	280
cin and cout as files	280
Using file names.....	287

TABLE OF CONTENTS

Chapter 14: Character Arrays and Dynamic Memory 293

Character arrays 293

 Antibugging 296

Dynamic allocation of arrays 299

 Antibugging 302

Using the * notation 303

 Antibugging 307

Chapter 15: Classes 309

Writing classes..... 309

Constructors..... 312

 Antibugging 316

const objects, const member functions . . . 318

 Antibugging 319

...and const parameters..... 319

Multiple constructors..... 320

 Copy constructors..... 320

 Default constructors 321

 Conversion constructors..... 322

 Summary 323

 Antibugging 323

Default parameters for code reuse 324

Date program (so far)..... 325

Chapter 16: Classes (Continued) 329

inline functions for efficiency..... 329

Access functions..... 331

Separate compilation and include files..... 332

 What happens in separate compilation 333

 Writing your .h file 334

 Backing up a multi-file project 337

 Antibugging 337

Multiple-file projects in Microsoft Visual Studio	339
Multiple-file projects in g++	340
Command line: more typing, less thinking	340
Makefiles: more thinking, less typing	340
Antibugging	344
Final Date program	344
static members (optional)	350
Chapter 17: Operators	353
The basic string class	353
Destructors	355
Binary and unary operators	356
Assignment operators and *this.....	358
Antibugging	360
Arithmetic operators	361
[] and ()	364
>> and <<: operators that aren't class members.....	365
++ and --	368
Explicit call to constructor	369
Final String program	370
#include <string>	376
Chapter 18: Exceptions, Move Constructors, Recursion, and O Notation	377
Exceptions.....	377
Move constructors and move = (optional)	382
Recursion (optional; used in the next section)	384
Antibugging	388
Algorithm analysis and O notation (optional)	389
Chapter 19: Inheritance	393
The basics of inheritance.....	393
Constructors and destructors	398
Inheritance as a concept.....	400

TABLE OF CONTENTS

Classes for card games	401
An inheritance hierarchy	404
private inheritance	408
Hiding an inherited member function	410
A game of Montana	411
Chapter 20: Templates	423
Function templates	423
Antibugging	425
The Vector class	426
Efficiency (optional)	431
Making Vector a template	432
Antibugging	436
Unusual class templates	437
#include <vector>	439
Chapter 21: Virtual Functions and Multiple Inheritance	441
Virtual functions	441
Behind the scenes	446
Pure virtual functions and abstract base classes	447
Why virtual functions often mean using pointers	447
Virtual destructors	452
Inheritance and move ctor/move = (optional)	454
Antibugging	455
Multiple inheritance	457
Antibugging	459
Chapter 22: Linked Lists	463
What lists are and why have them	463
List<T>::List ()	467
void List<T>::push_front (const T& newElement);	468
void List<T>::pop_front()	469
List<T>::~~List()	472

->: a bit of syntactic sugar.....	472
More friendly syntax: pointers as conditions	472
The linked list template	473
Antibugging	477
#include <list>.....	478
Chapter 23: The Standard Template Library	479
Iterators.....	479
... with vector, too	483
const and reverse iterators	484
Antibugging	485
Getting really lazy: ranges and auto.....	486
initializer_lists (optional).....	487
algorithm (optional).....	489
The erase-remove idiom	490
Antibugging	490
Chapter 24: Building Bigger Projects	493
Namespaces	493
Conditional compilation	494
Libraries	495
g++	496
Microsoft Visual Studio.....	498
Chapter 25: History.....	509
Simula 67.....	509
Smalltalk.....	509
What “object-oriented” is	510
C.....	511
C++.....	511
Standards.....	512

TABLE OF CONTENTS

Chapter 26: Esoterica (Recommended) 513

- sstream: using strings like cin/cout..... 513
- iomanip: formatted output 516
- Command-line arguments 522
 - Debugging with command-line arguments in Visual Studio 525
 - Debugging with command-line arguments in Unix 525
- static_cast et al 527
- Defaulted constructors and = 528
- constexpr and static_assert: moving work to compile time 529
- User-defined literals: automatic conversion between systems of measurement 533
- Lambda functions for one-time use..... 536
 - Lambda captures..... 537
- Structured bindings and tuples: returning multiple values at once 542
- Smart pointers 546
 - unique_ptr 546
 - shared_ptr 552
 - Antibugging 553
- Bit twiddling: &, |, ~, and..... 553
 - Antibugging 559

Chapter 27: Esoterica (Not So Recommended)..... 561

- protected sections, protected inheritance 561
- Template specialization 565
- friends, and why you shouldn't have any..... 566
- User-defined conversions 571

Chapter 28: C 573

- Compiling C..... 574
- I/O 575
 - printf..... 575
 - scanf, and the address-of (&) operator..... 575
 - fprintf and fscanf; fopen and fclose 578
 - sprintf and sscanf; fputs and fgets..... 580

Summary	583
Antibugging	584
Parameter passing with *	584
Antibugging	588
Dynamic memory	588
Chapter 29: Moving On with SDL	591
Writing code	593
Compiling	597
Further resources	598
Appendix A: SDL/SSDL Setup Issues.....	599
Unix	599
g++	599
SDL.....	599
SSDL	600
Making your own Makefiles.....	600
Antibugging.....	600
MinGW.....	601
g++	601
SDL and SSDL	601
Making your own Makefiles.....	601
Antibugging.....	601
Microsoft Visual Studio	602
SDL/SSDL	602
Making your own project files.....	602
Antibugging.....	603
Sound.....	604
Appendix B: Operators.....	605
Associativity	605
Precedence	605
Overloading.....	606

TABLE OF CONTENTS

Appendix C: ASCII Codes 607

Appendix D: Fundamental Types..... 609

Appendix E: Escape Sequences 611

Appendix F: Basic C Standard Library 613

 cmath 613

 ctype 614

 cstdlib 615

Appendix G: Common Debugger Commands 617

 Microsoft Visual Studio 617

 gdb/ddd..... 618

Appendix H: SSDL Reference 619

 Updating the screen..... 619

 Added types 619

 Clearing the screen..... 620

 Colors..... 620

 Drawing..... 620

 Images 621

 Mouse, keyboard, and events 622

 Music 623

 Quit messages 624

 Sounds..... 625

 Sprites..... 626

 Text 628

 Time and synchronization..... 629

 Window 630

References..... 631

Index..... 633

About the Author

Will Briggs, PhD, is a professor of computer science at the University of Lynchburg in Virginia. He has over 20 years of experience teaching C++, 12 of them using earlier drafts of this textbook, and more than that teaching other languages including C, LISP, Pascal, PHP, PROLOG, and Python. His primary focus is teaching of late while also doing research in artificial intelligence.

About the Technical Reviewer

Michael Thomas has worked in software development for over 20 years as an individual contributor, team lead, program manager, and vice president of engineering. He has over 10 years of experience working with mobile devices. His current focus is in the medical sector using mobile devices to accelerate information transfer between patients and health-care providers.

Acknowledgments

Special thanks to

- Dr. Kim McCabe, for advice on publishing
- Dr. Zakaria Kurdi, for advice on publishing
- Apress
- Microsoft
- The makers of GIMP (the GNU Image Manipulation Program)
- Pixabay.com and contributors, especially David Mark/12019 (Chapter 2, beach), Free-Photos (Chapter 2, pug), Andi Caswell/andicaz (Chapter 6, scones), joakant (Chapter 11, tropical fish), Gerhard Janson/Janson_G (Ch12, ufo), 13smok (Chapter 12, alien sign), Prawny (Chapter 12, splat), Elliekha (Chapter 12, haunted house), pencil parker (Chapter 12, candy), and Robert Davis/rescueram3 (Chapter 12, pumpkin photos)
- Wikimedia Commons
- OpenClipArt.org and contributors, especially Firkin (Chapter 2, flamingo)
- Flickr, especially Speedy McZoom (Chapter 12, jack-o-lantern art)
- FreeSounds.org and contributors, especially Razor5 (Chapter 2, techno music), robbo799 (Chapter 2, church bells), alqutis (Chapter 12, hovercar), Berviceps (Chapter 12, splat), mistersherlock (Chapter 12, Hallowe'en graveyard), matypresidente (Chapter 12, water drop), Osiruswaltz (Chapter 12, bump), mrose6 (Chapter 12, echoed scream), and robcro6010 (Chapter 12, circus theme)

ACKNOWLEDGMENTS

- Chad Savage of Sinister Fonts for Werewolf Moon (Chapter 12)
- Lazy Foo' Productions
- StackOverflow.com
- Einar Egilsson of cardgames.io for images of card games, Nicu Buculei (<http://nicubunu.ro/cards>) for card images, and “Nanami Kamimura” for Montana image

Introduction

Surely there's no shortage of C++ intro texts. Why write yet another?

I'm glad you asked.

Ever since moving from Pascal to C++ (back when dinosaurs roamed the Earth), I've been underwhelmed by available resources. I wanted something quirky and fun to read, with sufficient coverage and fun examples, like the old *Oh! Pascal!* text by Cooper and Clancy.

It's about time we had this again. Even a perfectly accurate text with broad coverage gives you nothing if you fall asleep when you read it. Well, nothing but a sore neck.

But the other reason, of course, is to promote laziness.

We all want our projects to be done more quickly, with less wailing and gnashing of teeth. Sometimes, it's said, you have to put your nose to the grindstone. Maybe, but I like my nose too well for that. I'd rather do things the easy way.

But the easy way isn't procrastinating and dragging my feet: it's to find something I love doing and do it so well that it feels relatively effortless. It's producing something robust enough that when it does break down, it tells me exactly what the problem is, so I don't have to spend a week pleading with it to explain itself. It's writing code that I can use again and again, adapting it to a new use in hours instead of days.

You'll benefit from this book if you're a beginning programmer or one who hasn't yet learned C++ or its descendants like Java or C#; if you already know a C++-like language, you can go a little faster.

Here's what you can expect:

- A pleasant reading experience.
- Adequate coverage.
- Games, that is, use of the SDL graphics library, which makes it easy to get graphics programs working quickly and easily. It isn't fair that Python and Visual Basic should get all the eye candy.¹ The SDL

¹"Eye candy": things that look good on the screen. See *The New Hacker's Dictionary*, available at time of writing at www.catb.org/jargon/.

INTRODUCTION

library is used through Chapter 12. After that we'll use more standard (but less visually interesting) I/O, so we can also get practice with the more common console programs.

- ...and an easy introduction to SDL's graphical magic, using the SSDL library (see below).
- Sufficient examples, and they won't all be about actuarial tables or how to organize an address book. (See "pleasant reading experience" earlier.)
- Antibugging sections throughout the text to point out common or difficult-to-trace errors – and how to prevent them.
- Compatibility with g++ and Microsoft Visual Studio.
- Compliance with C++17, the latest standard, and the nice goodies it provides.
- For g++ programmers, instructions on using g++, the ddd/gdb debugger system, and Makefiles; for Visual Studio, use of the debugger and project files.
- An appreciation of laziness.
- A cool title. Maybe I could have tried to write a "For Dummies" book, but after seeing *Bioinformatics for Dummies* I'm not sure I have what it takes.

Why SDL?

It's surely more enjoyable to make programs with graphics and WIMP²-style interaction than to merely type things in and print them out. There's a variety of graphical libraries out there. SDL, or Simple DirectMedia Layer, is popular, relatively easy to learn, portable between platforms, and fast enough for real-world work, as evidenced by its use in actual released games (Figure 1).

²WIMP: window, mouse, icon, pointer. What we're all used to.



Figure 1. A game of Freeciv, which uses the SDL library.

Why SSDL?

...but although SDL is *relatively* simple, it's not simple enough to start with on day 1 of programming with C++. SSDL (*Simple* SDL) saves you from needing to know things we don't get to until Chapter 14³ before doing basic things like displaying images (Chapter 2) or even printing a greeting (Chapter 1). It also hides the initialization and cleanup code that's pretty much the same every time you write a program, and makes error handling less cumbersome.

You may want to keep using SSDL as is after you're done with this book, but if you decide to go on with SDL, you'll find you know a lot of it already, with almost nothing to unlearn: most SSDL function names are names from SDL with another "S" stuck on the front. We'll go into greater depth on moving forward with SDL in Chapter 29.

³Pointers.

Software You Will Need

Your compiler, plus various free SDL libraries (SDL2, SDL2_Image, SDL2_TTF, and SDL2_Mixer), my free SSDL library, and (for Chapter 2, and whenever you need it) a deluxe graphics editing package. I use GIMP, which is free, and at time of writing is available from www.gimp.org.

SSDL is available at www.apress.com/9781484251867, as is my sample code.

In Unix, you may choose to install the GNU Free Fonts library, or msttcorefonts, Microsoft Core Fonts for the Web. Look for `ttf-mscore-fonts` and `fonts-freefont-ttf` (Debian and Ubuntu systems) and `gnu-free-fonts-common` and `msttcore-fonts-<something or other>` (Red Hat and Fedora), remembering that systems differ, standards change, and Unix is hard. But if you're using Unix, you knew that. I use Microsoft Core Fonts for the Web in the example programs.

Programming with sound may not be practical over remote connections, because of the difficulty of streaming sound. If using Unix emulation, you might check the emulator's sound capabilities – say, by playing a video.

If this is for a course...

C++ for Lazy Programmers covers through pointers, operator overloading, virtual functions, templates, exceptions, STL – everything you might reasonably expect in two semesters of C++.

The SSDL library does take a small amount of time, but the focus is firmly on writing good C++ programs, with SSDL there just to make the programs more enjoyable. How many labs or projects do you have in which it's hard to stop working because it's so much fun? It may not happen with *all* these problems, but I do see it happen.

SDL also gives a gentle introduction to event-driven programming.

In the first 12 chapters, there is emphasis on algorithm development and programming style, including early introduction of constants.

After Chapter 12, the examples are in standard I/O, though SDL is still an option for a few exercises and is used in Chapter 21 and (briefly) Chapter 26.

A normal two-semester sequence should cover approximately

- Semester 1: The first 12 chapters, using SDL; Chapter 13, introducing standard I/O. With some exceptions (& parameters, stream I/O), this looks a lot like C, and includes variables, expressions, functions, control structures, arrays, and stream I/O.
- Semester 2: Chapters 14–22, using standard I/O, covering pointers, character arrays, classes, operator overloading, templates, exceptions, virtual functions, multiple inheritance (briefly), and a taste of the Standard Template Library using vectors and linked lists.

Subsequent chapters cover material that wouldn't easily fit in two semesters, including more of the Standard Template Library, history of C++, C programming, and a few more esoteric topics.

Online Help

Here are some sites to go to for more information, with URLs correct at time of writing.

SDL: www.libsdl.org; click “Wiki.” You’ll find a reference for SDL functions.

SDL’s helper libraries SDL_Image, SDL_Mixer, and SDL_TTF: www.libsdl.org/projects/SDL_image/, www.libsdl.org/projects/SDL_mixer/, and www.libsdl.org/projects/SDL_ttf/. In each case, click Documentation. You’ll find references for their functions. If the web sites have changed, doing a web search for the name of the library (SDL_image, for example) should get you there.

Legal Stuff

Visual Basic, Visual Studio, Windows, Windows Vista, Excel, and Microsoft are trademarks of the Microsoft Corporation. All other trademarks referenced herein are property of their respective owners.

This book and its author are neither affiliated with nor authorized, sponsored, or approved by Microsoft Corporation.

Screenshots of Microsoft products are used with permission from Microsoft.

CHAPTER 1

Getting Started

Most programs in the first half of this book use the SDL and SSDL libraries,¹ on the theory that watching colorful shapes move across the screen and shoot each other is more interesting than printing text. Don't worry; when you're done, you'll be able to write programs both with and without this library – and if I have anything to say about it, you'll have had fun doing it. Let's see how it goes.

A simple program

It's wise to start small. Fewer things can go wrong.

So we'll start small here with a simple program that writes “Hello, world!” on the screen. We'll take it line by line to see what's in it. (In the next section, we'll compile and run it.)

Example 1-1. “Hello, world!” is a classic program to start off a new language with. (I think it's a law somewhere.)

```
//Hello, world! program, for _C++ for Lazy Programmers_  
// Your name goes here  
// Then the date2
```

¹SDL provides graphics, sound, and friendly interaction including mouse input. SSDL, standing for *Simple* SDL, is a “wrapper” library that wraps SDL's functions in easier-to-use versions. Both libraries are described in more detail in the Introduction. Don't worry; I don't read introductions either.

²From here on, I'll be putting the title of the text, rather than name and date, because that's more useful for textbook examples. Ordinarily name of programmer and date are better for keeping track of what was done and who to track down if it doesn't work.

CHAPTER 1 GETTING STARTED

```
//It prints "Hello, world!" on the screen.
//    Quite an accomplishment, huh?

#include "SSDL.h"

int main (int argc, char** argv)
{
    sout << "Hello, world!  (Press any key to quit.)\n";

    SSDL_WaitKey ();      //Wait for user to hit any key

    return 0;
}
```

The first set of lines are comments. **Comments** look like this – `//Something` on a line after two slashes – and are there just for you or for someone who later tries to understand your program. It’s best to be kind to yourself and your maintainers – help them easily know what the program’s doing, without having to search and figure it out.

Next we have an include file. Some language features are built into the C++ compiler itself, like the comment markers `//` and `#include`. Other things are in libraries; they’ll only be loaded if needed. In this case, we need to know how to print things on the screen using the SSDL library, so we load the include file `SSDL.h`.

Next we have the main program. `main ()` is special: it’s what that tells the compiler, “This is what we’re doing in the program; start here.” The `int` at the start and the weird sequence `int argc, char** argv` we’ll get in Chapter 26 under “Command-line arguments.” Same for the `return 0;` at the end. For now, we always put these things in. If not, the C++ gods will punish us with incomprehensible error messages.

In this case, `main ()` only does two things.

First, it prints “Hello, world” using the `sout` object, pronounced “S-out.”

Second, it calls `SSDL_WaitKey ()`, which waits for you to hit a key before ending the program. Otherwise the program closes before you have a chance to see its message.

We return 0 because `main ()` has to return something, largely for historical reasons. In practice we almost never care what `main` returns.

The curly braces `{}` tell `main ()` where to start taking action and where to end: whatever you want the computer to do when it runs the program goes between the curly braces.

The compiler is very picky about what you type. Leave off a `;` and the program won't compile. Change capitalization on something and C++ won't recognize it.

If you're curious what this program would have looked like without SSDL, see Chapter 29. It's not for the fainthearted beginner, but later it should make perfect sense.

Extra “Hello, world!” is often the first program a beginner writes in a new language. Although it was originally a simple example in C – the language C++ is descended from; more on that in Chapter 25 – the practice of writing this as the first program has spread. Here's “Hello, world!” in BASIC:

```
10 PRINT "Hello, world!"
```

Not bad, huh?

This is what it looks like in APL. APL (A Programming Language) has been described as a “write-only” language because it's said you can't read the programs you wrote yourself. APL requires symbols such as \square , ∇ , and ρ .

```
 $\square \leftarrow$  'Hello, world!'
```

Although those look easier than C++'s version, C++'s is neither the longest nor the toughest. I'll spare you the long ones to save trees (an example for the language Redcode took 158 lines, which may be why you've never heard of Redcode), but here's one of the tough ones, from a purposefully difficult language sometimes called BF.

```
+++++[>++++>++++>++++>++++>+<<<<-]>
+++++.>++++.+++++. .++>>----.>.<<+++++.<.>
-----.<---.-----.>>>+.
```

More “Hello, world!” examples, at time of writing, can be found at <http://helloworldcollection.de/>.

Spacing

One thing the compiler *doesn't* care about is spacing. As long as you don't put a space inside a word, you can put it wherever you like. You can break lines or not as you choose; it won't care, as long as you don't break a `//comment` or a "quotation".

Example 1-2. A blatant instance of evil and rude³ in programming

```
//Hello, world! program, for _C++ for Lazy Programmers_
//It prints "Hello, world!" on the screen.
//Quite an accomplishment, huh?
//          -- from _C++ for Lazy Programmers_

#include "SSDL.h"

        int main (int argc, char** argv) {
    sout <<
    "Hello, world! (Press any key to quit.)\n";

        SSDL_WaitKey ();      //Wait for user to hit any key

return 0;
    }
```

The compiler won't care about spacing – but the poor soul that has to understand your 500-page program will! Example 1-2's spacing would be a cruel thing to do to the people who later maintain your code.

Readability is a Good Thing.⁴ The programmer struggling to figure what you meant may very well be you a few days after writing it. Most of the expense in software development is programmer time; you won't want to waste yours trying to decipher your own code. *Make it clear.*

³"Evil and rude" is a technical term meaning, essentially, "maliciously awful." See *The New Hacker's Dictionary*, currently online at www.catb.org/jargon, for other terms in programmers' slang.

⁴Good Thing: hacker slang for something that's completely wonderful and everybody knows it (or should).

Tip Make your code clear *as you write it*, not later. Readable code helps with development, not just future maintenance.

To help with clarity, I have things in Example 1-1, like initial comments, `#include`, and `main ()`, separated by **blank lines**. It's sort of like writing paragraphs in an English paper: each section is its own "paragraph." Blank lines increase readability.

I also break lines in sensible places and indent in a way that makes the program easy to read. The default **indentation** is the left margin. But if something is contained in something else – as the `sout` statement is contained in the main program – it gets indented one tab, or a few spaces.

This is like outline format for a paper, or like the layout of a table of contents (Figure 1-1). What's contained in something else is indented slightly. What's in Example 1-2 breaks the rule because `#include "SSDL.h"` isn't part of the comment above it, so it shouldn't be indented relative to it. `int main (int argc, char** argv)` isn't part of `#include "SSDL.h"`, so it shouldn't be indented either.

<pre>int main (int argc, char** argv) { sout << "Hello, world!\n"; SSDL_WaitKey (); return 0; }</pre>	<pre>My wonderful paper Part One Part Two Reasons Part One is wrong a. Why those reasons fail b. Reassurance it's right Reasons Part Two is way better Conclusion</pre>
---	--

Figure 1-1. Like an English paper outline, a C++ program is indented, with subparts indented relative to what they're parts of

By contrast, the `sout` statement in Example 1-1 is contained in `main`, so it gets indented a little.

You'll have plenty of examples of clear indenting as you read on.