



C++20 for Lazy Programmers

Quick, Easy, and Fun C++ for Beginners

—

Second Edition

—

Will Briggs

Apress®

C++20 for Lazy Programmers

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

Second Edition

Will Briggs

Apress®

C++20 for Lazy Programmers: Quick, Easy, and Fun C++ for Beginners

Will Briggs
Lynchburg, VA, USA

ISBN-13 (pbk): 978-1-4842-6305-1
<https://doi.org/10.1007/978-1-4842-6306-8>

ISBN-13 (electronic): 978-1-4842-6306-8

Copyright © 2021 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 Susan Wilkinson on Unsplash (www.unsplash.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. 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 booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail 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/9781484263051. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*To the learners and readers. They tell me
what's wrong, and what's right.*

Table of Contents

About the Author	xvii
About the Technical Reviewer	xix
Acknowledgments	xxi
Introduction	xxiii
 Chapter 1: Getting Started	 1
Initial setup	1
... in Unix	2
... in MinGW	3
... in Microsoft Visual Studio.....	5
A simple program.....	6
Spacing.....	8
Creating an SSDL project	10
... with g++ (Unix or MinGW).....	10
... in Microsoft Visual Studio.....	13
How not to be miserable (whatever your platform).....	23
Shapes and the functions that draw them.....	25
consts and colors.....	36
Text	39
sout, escape sequences, and fonts	39
SSDL_RenderText, SSDL_RenderTextCentered	43

TABLE OF CONTENTS

Chapter 2: Images and Sound	47
Images and window characteristics	47
Multiple images together	55
Adding transparency with GIMP	57
Sound.....	62
Chapter 3: Numbers.....	67
Variables	67
Constants	70
When to use constants, not literal values.....	70
Math operators.....	72
Integer division	73
Assignment (=) operators.....	73
A diving board example	74
The no-worries list for math operators.....	77
Built-in functions and casting	78
Chapter 4: Mouse, and if.....	85
Mouse functions	85
if.....	89
Coercion and if conditions (if's dirty little secret).....	92
Combining conditions with &&, , and !.....	92
Boolean values and variables	96
A hidden object game	98
Chapter 5: Loops, Input, and char.....	107
Keyboard input.....	107
while and do-while	111
Loops with SSDL.....	112
break and continue.....	114
for loops	118
Increment operators	118
An example: Averaging numbers	119

chars and ctype.....	123
switch	128
Chapter 6: Algorithms and the Development Process	133
Adventures in robotic cooking	133
Writing a program, from start to finish	137
Requirements: What do we want to do?	137
Algorithm: How do we do it?	138
Trace the algorithm: Will it work?	140
Coding: Putting it all into C++ (plus: commenting the lazy way).....	140
Chapter 7: Functions	147
Functions that return values	147
Functions that return nothing	154
Global variables.....	158
How to write a function in four easy steps (and call it in one).....	162
Why have functions, anyway?	168
Recap	176
Chapter 8: Functions, Continued.....	179
Random numbers.....	179
Making a random number generator.....	179
Using the built-in random number generator	182
Boolean functions	187
& parameters	189
Identifier scope	195
A final note on algorithms.....	197
Chapter 9: Using the Debugger	199
Breakpoints and watched variables.....	203
ddd	204
gdb	205
Visual Studio.....	205

TABLE OF CONTENTS

Fixing the stripes	207
Going into functions	207
ddd	207
gdb	208
Visual Studio.....	209
Fixing the stars	210
Wrap-up	211
Bottom-up testing	212
More on antibugging.....	213
Chapter 10: Arrays and enum.....	217
Arrays.....	217
Arrays' dirty little secret: using memory addresses.....	220
Arrays as function parameters.....	222
Array parameters that change, or don't.....	222
Array parameters and reusability	223
Enumeration types	225
Multidimensional arrays	229
Displaying the board.....	230
Arrays of more than two dimensions.....	234
Chapter 11: Animation with structs and Sprites	237
structs.....	237
Making a movie with struct and while.....	242
Sprites.....	249
Chapter 12: Making an Arcade Game: Input, Collisions, and Putting It All Together.....	257
Determining input states	257
Mouse.....	257
Keyboard	258
Events	260
Cooldowns and lifetimes.....	262

Collisions.....	266
The big game	267
Chapter 13: Standard I/O and File Operations	285
Standard I/O programs.....	285
Compiling standard I/O programs.....	286
Building a project from scratch (optional)	287
File I/O (optional).....	291
cin and cout as files	292
Using filenames.....	298
Chapter 14: Character Arrays and Dynamic Memory	305
Character arrays	305
Dynamic allocation of arrays	312
Using the * notation	318
Chapter 15: Classes.....	325
Constructors.....	328
const objects, const member functions	334
...and const parameters.....	335
Multiple constructors	336
Copy constructors.....	336
Default constructors	337
Conversion constructors.....	338
Summary	338
Default parameters for code reuse	340
Date program (so far).....	341
Chapter 16: Classes, Continued	345
inline functions for efficiency.....	345
Access functions.....	347
Separate compilation and include files.....	348

TABLE OF CONTENTS

What happens in separate compilation	349
Writing your .h file	350
Backing up a multi-file project	353
Multiple-file projects in Microsoft Visual Studio	355
Multiple-file projects in g++	357
Command line: more typing, less thinking	357
Makefiles: more thinking, less typing (optional)	357
Final Date program	362
Chapter 17: Operators	369
The basic string class	369
Destructors	371
Binary and unary operators: ==, !=, and !	372
All other comparison operators at once	373
Assignment operators and *this	374
Arithmetic operators	377
[] and ()	380
++ and --	382
>> and <<: operators that aren't class members	383
static members	386
Explicit call to constructor	387
The final String program	388
#include <string>	394
Chapter 18: Exceptions, Move Constructors and =, Recursion, and O notation	395
Exceptions	395
Move constructors and move = (optional)	400
Recursion (optional; referenced in the next section)	402
Efficiency and O notation (optional)	407

Chapter 19: Inheritance	411
The basics of inheritance	411
Constructors and destructors for inheritance and member variables	416
Inheritance as a concept	417
Classes for card games	419
An inheritance hierarchy	422
private inheritance	425
Hiding an inherited member function	427
A game of Montana	428
Chapter 20: Templates	439
Function templates	439
Concepts for function templates (optional)	442
The Vector class	445
Efficiency and O notation (optional)	449
Making Vector a template	451
Concepts for class templates (optional)	455
pair	457
Non-type template arguments	458
#include <vector>	460
Chapter 21: Virtual Functions and Multiple Inheritance	461
Virtual functions, plus: move functions with movable parents and class members	461
Behind the scenes	466
Pure virtual functions and abstract base classes	467
Why virtual functions often mean using pointers	467
Virtual destructors	472
Move functions with movable parents and class members (optional)	474
Multiple inheritance (optional)	478

TABLE OF CONTENTS

Chapter 22: Linked Lists 483

What lists are and why have them..... 483

Efficiency and O notation (optional) 485

Starting the linked list template..... 486

List<T>::List () 488

void List<T>::push_front (const T& newElement); 488

void List<T>::pop_front () 490

List<T>::~List () 492

->: a bit of syntactic sugar..... 493

More friendly syntax: pointers as conditions 493

The linked list template 494

#include <list>..... 499

Chapter 23: The Standard Template Library 501

Iterators..... 501

 ... with vector too 504

 const and reverse iterators 505

Getting really lazy: range-based for and auto 508

Spans 509

initializer_lists (optional)..... 512

<algorithm> (optional)..... 514

Chapter 24: Building Bigger Projects 517

Namespaces 517

Conditional compilation 518

Libraries 519

 g++ 520

 Microsoft Visual Studio..... 522

Chapter 25: Esoterica (Recommended) 533

sstream: using strings like cin/cout..... 533

Formatted output with format strings..... 537

Command-line arguments 543

Debugging with command-line arguments in Unix	546
Debugging with command-line arguments in Visual Studio	547
Bit manipulation: &, , ~, and <</>>	549
Chapter 26: Esoterica (Recommended), Continued	557
Defaulted constructors and =	557
constexpr and static_assert: moving work to compile time	558
Structured bindings and tuples: returning multiple values at once	562
Smart pointers	566
unique_ptr	566
shared_ptr	572
static_cast et al.	573
User-defined literals: automatic conversion between systems of measurement	575
Lambda functions for one-time use	578
Lambda captures	580
An example with lambda functions	581
Chapter 27: Esoterica (Not So Recommended)	585
protected sections, protected inheritance	585
friends and why you shouldn't have any	590
User-defined conversions (cast operators)	594
Modules	596
Coroutines	596
Chapter 28: C	601
Compiling C	602
I/O	603
printf	603
scanf and the address-of (&) operator	604
fprintf and fscanf, fopen and fclose	606
sprintf and sscanf; fgets, fputs, and puts	608
Summary of commands	611

TABLE OF CONTENTS

Parameter passing with * 613

Dynamic memory 616

Chapter 29: Moving on with SDL 619

Writing code..... 622

Compiling 625

Further resources..... 626

Appendix A: Help with Setup 627

...for Unix users..... 627

 Debian/Ubuntu..... 627

 RedHat/Fedora..... 628

 SSDL..... 628

...for MinGW users 629

...for Microsoft Visual Studio users 630

...for other platforms..... 631

Sound..... 631

Making your own projects 632

 ...in g++..... 632

 ...in Microsoft Visual Studio..... 632

Appendix B: Operators..... 635

Associativity..... 635

Precedence 635

Overloading..... 637

Appendix C: ASCII Codes 639

Appendix D: Fundamental Types..... 641

Appendix E: Escape Sequences 643

Appendix F: Basic C Standard Library 645

cmath..... 645

cctype 646

cstdlib 646

Appendix G: Common Debugger Commands	647
Microsoft Visual Studio	647
gdb/ddd.....	648
Appendix H: SSDL Reference	649
Updating the screen.....	649
Added types	649
Clearing the screen.....	650
Colors.....	650
Drawing.....	651
Images	652
Mouse, keyboard, and events	653
Music	653
Quit messages	655
Sounds.....	655
Sprites.....	657
Text	659
Time and synchronization	660
Window	660
References.....	661
Index.....	663

About the Author

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

About the Technical Reviewer



Dr. Charles A. Bell conducts research in emerging technologies. He is a principal software developer of the Oracle MySQL Development team. He lives in a small town in rural Virginia with his loving wife. He received his Doctor of Philosophy in Engineering from Virginia Commonwealth University in 2005.

Dr. Bell is an expert in the database field and has extensive knowledge and experience in software development and systems engineering. His research interests include microcontrollers, three-dimensional printing, database systems, software engineering, and sensor networks. He spends his limited free time as a practicing maker focusing on microcontroller projects and refinement of three-dimensional printers.

Acknowledgments

Special thanks to

- Dr. Kim McCabe, for advice on publishing.
- Dr. Zakaria Kurdi, for the same.
- Apress, especially Steve Anglin.
- Microsoft.
- The makers of GIMP (the GNU Image Manipulation Program).
- Pixabay.com and contributors, especially 3D Animation Production Company/QuinceCreative (Chapter 1, bullseye), 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 (Chapter 12, 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).
- FreeSound.org and contributors, especially Razor5 (Chapters 1 and 2, techno music), robbo799 (Chapter 2, church bells), alqutis (Chapter 12, hover car), 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).
- Chad Savage of Sinister Fonts for Werewolf Moon (Chapter 12).

ACKNOWLEDGMENTS

- Lazy Foo' Productions.
- StackOverflow.com.
- Einar Egilsson of cardgames.io for images of card games and Nicu Buculei (<http://nicubunu.ro/cards>) for card images.
- The alumni and colleagues who gave me reviews. You're the best!

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. 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 well enough 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.

Here's what you can expect in this book:

- A pleasant reading experience.
- Adequate coverage.
- Games, that is, use of the SDL (Simple DirectMedia Layer) graphics library, which makes it easy to get graphics programs working quickly. It isn't fair that Python and Visual Basic should get all the eye candy.¹ The SDL library is used through Chapter 12. After that, we'll mostly use standard I/O, so we can get practice with the more common console programs.

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

INTRODUCTION

- An easy introduction to SDL's graphical magic, using the SSDL (*simple* SDL) library (see below).
- Sufficient examples—and they won't all be about actuarial tables or how to organize an address book. (See “A pleasant reading experience” above.)
- Antibugging sections throughout the text to point out common or difficult-to-trace errors and how to prevent them.
- For g++ programmers, instructions on using g++, the ddd/gdb debugger system, and Makefiles; for Visual Studio, use of the debugger and project files.
- Compliance with C++20, the latest standard, and the goodies it provides.
- Hands-on experience with advanced data types like strings, stacks, vectors, and lists – not by reading about them, but by building them yourself.
- 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 are 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.

²WIMP: window, icon, mouse, 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* easy, it's not simple enough to start 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.

(Free) software you will need

At the time of writing, Microsoft Visual Studio (Community Edition) for Windows is absolutely free, and g++ always is. So are the SSDL and SDL2 libraries; Microsoft Core fonts for the Web, which you'll need on Unix systems; and the GIMP deluxe graphics editing package. See Chapter 1 and Appendix A for help installing these essentials.

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++20 for Lazy Programmers covers through pointers, operator overloading, virtual functions, templates, exceptions, STL (Standard Template Library), and everything you might reasonably expect in two semesters of C++ – plus extras at the end.

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 SSDL is still an option for a few exercises and is used in Chapter 21 and (briefly) Chapters 25 and 26.

A normal two-semester sequence should cover approximately the following:

- Semester 1: The first 12 chapters, using SSDL; Chapter 13, introducing standard I/O. With some exceptions (& parameters, stream I/O, constexpr), 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, dynamic memory, 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, C programming, and advanced topics including the use of command-line arguments, bit manipulation, format strings, lambda functions, and smart pointers.

Online help

Here are some sites to go to for more information, with URLs correct at the 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 websites have changed, doing a web search for the name of the library (e.g., SDL_Image) 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 the 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 graphics-and-games 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.

If you've already chosen your platform, great. If not, here's my recommendation:

- If you just want to learn C++ on an easy and easily managed platform, Microsoft Visual Studio is great.
- If you are a Unix system administrator or have good access to one, and want to use that popular and powerful platform, go for it.
- To learn g++ and make – powerful tools from the Unix world – in Windows, with a relatively easy setup, use MinGW.

The programming won't differ much between platforms. But system setup can be an issue.

Initial setup

First, you'll need the source code for the textbook. You can access the code via the Download Source Code button located at www.apress.com/9781484263051.

Then unzip it. In Unix, the `unzip` command should work; in Windows, you can usually double-click it or right-click and choose Extract or Extract All.

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.

...in Unix

Getting around in Unix isn't in the scope of this book, but no worries. The basics of copying files, moving files, and so on are easy to pick up.²

Unix system administration is *way* beyond the scope of this book.³ But installing SSDL is easy. In that folder you just unzipped

- Go into `external/SSDL/unix`, and type `make`. This builds SSDL in a place where programs in the source code will know where to find it.
- Go into `ch1/test-setup`.
- `cp Makefile.unix Makefile`
- `make`
- `./runx`

You should see (and hear) the program illustrated in Figure 1-1. (If not, something must be missing – please see Appendix A.) You might take a moment to try another program from `ch1`, like `1-hello`. Run it the same way you did `test-setup`.

²I recommend UNIX Tutorial for Beginners, at www.ee.surrey.ac.uk/Teaching/Unix/. Up through Tutorial 4 should be fine for now. Or search for your own.

³OK, I can't just let it go at that. Appendix A has suggestions on how to install the other tools you need (g++, SDL, etc.). But distributions of Unix vary, so it'll help to know what you're doing.

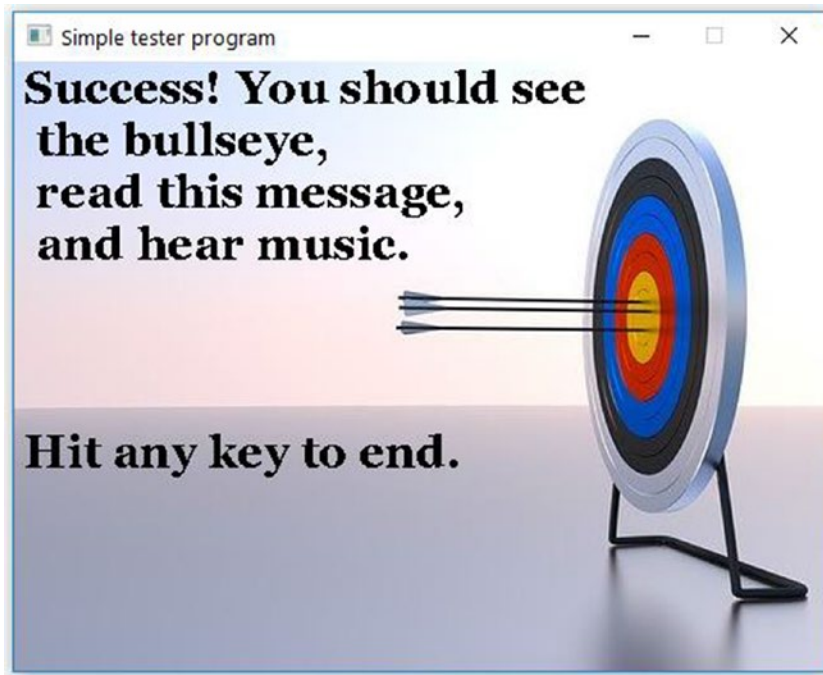


Figure 1-1. Output of test-setup

...in MinGW

You can find MinGW at sourceforge.net and other places. Try a web search on “MinGW download.”

Once that’s installed, have it add the basics for C++; start the MinGW Installation Manager (`mingw-get.exe`) and have it install, at least, `mingw32-gcc-g++-bin`, `mingw32-gdb-bin`, and `msys-make-bin`.

You *won’t* need to install SDL or SSDL; they’re in the source code you unzipped.

So let’s try ‘em out. Open the Windows command prompt (click the Start Menu and type `cmd`) and go to the source code’s `ch1/test-setup` folder. Here’s an easy way: in the window for that folder, click the folder icon left of the address bar, the part that shows something like `... > ch1 > test-setup`. It’ll be replaced by a highlighted path like the one in Figure 1-2. Press Ctrl-C to copy it.

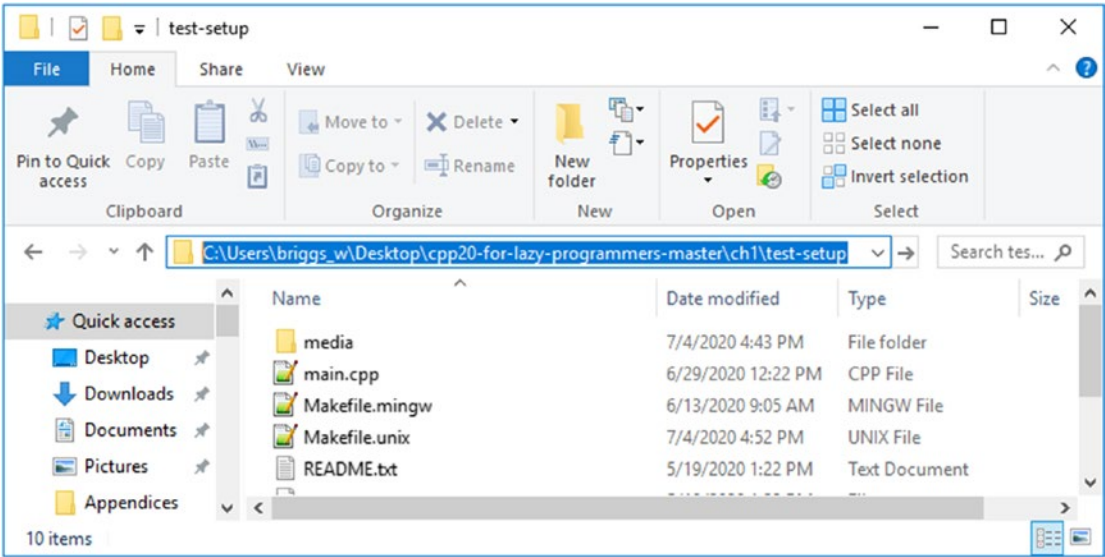


Figure 1-2. Getting a path to use with the command prompt, in Windows

In the command window, enter the first two characters of that path you copied (in my case, `C:`); then type `cd`, paste in the path (`Ctrl-V`), and press Enter again (see Figure 1-3).

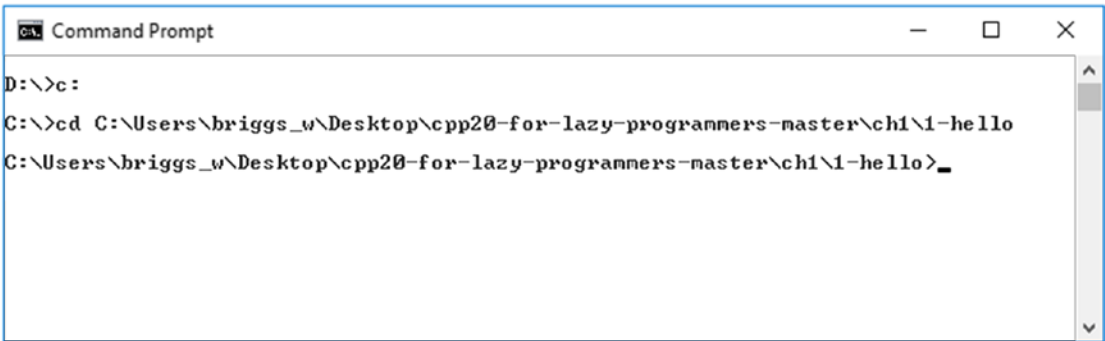


Figure 1-3. Getting to the right folder in the command prompt

Then

```
copy Makefile.mingW Makefile
make
bash runw
```

You should see (and hear) the program illustrated in Figure 1-1. (If not, see Appendix A.) You might take a moment to try another program from `ch1`, like `1-hello`. Run it the same way you did `test-setup`.

...in Microsoft Visual Studio

At the moment, Visual Studio is absolutely free. Go to Microsoft's download page (currently visualstudio.microsoft.com/downloads/) and download the Community Edition.

It'll take a long time to install. Be sure to put a check by Desktop development with C++ (Figure 1-4, upper right) – else, you'll have Visual Studio, all right, but it won't know C++.

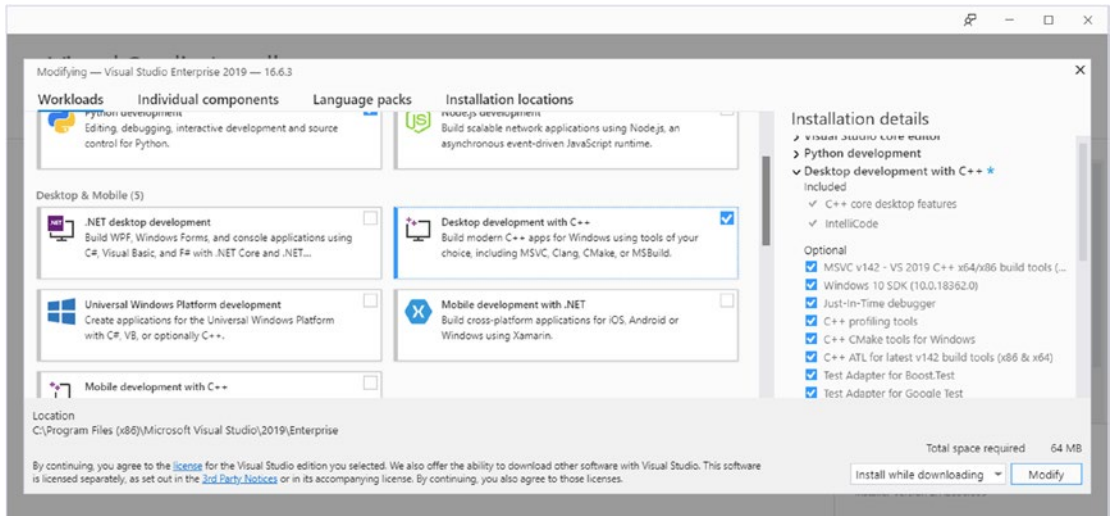
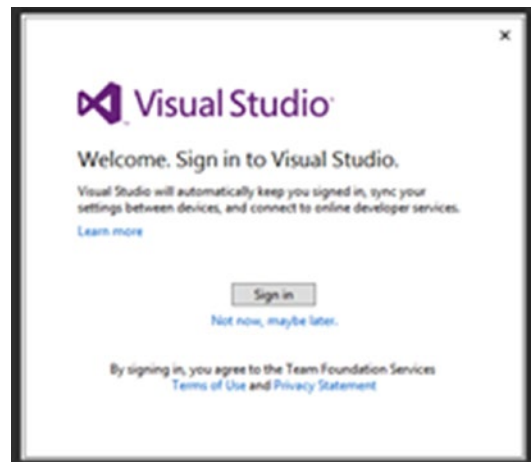


Figure 1-4. Installing the C++ part of Visual Studio

When it's installed, go to the book's source code folder, into the ch1 subfolder; double-click the solution file, ch1.sln or ch1. (If it asks you to sign in and you're not ready to do that now, notice the line "Not now, maybe later.")

Now, in the Solution Explorer window (see Figure 1-5), you should see at the bottom a project named **test-setup**. Right-click it, and select Debug ► Start New Instance.



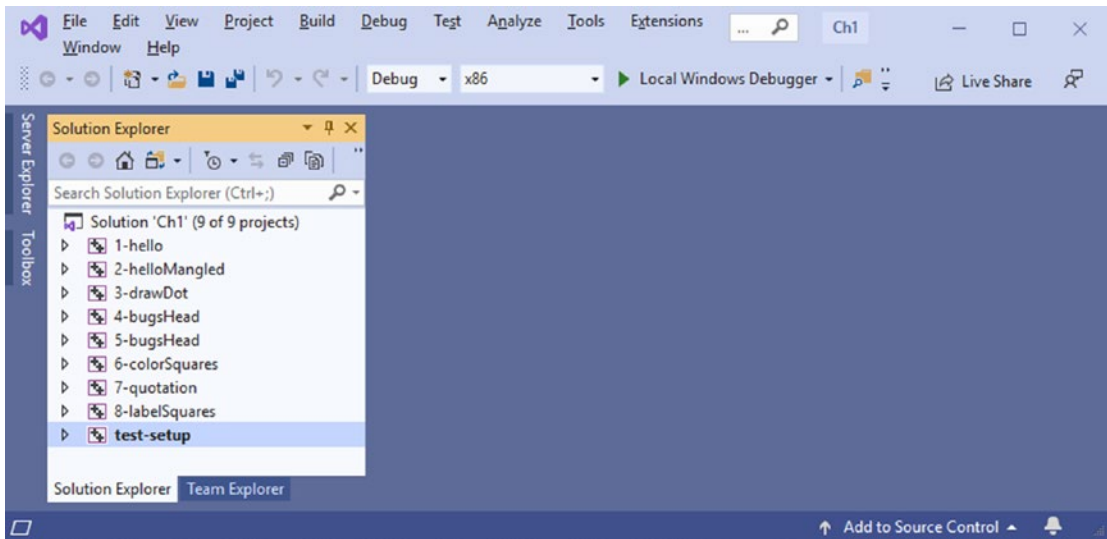


Figure 1-5. The *ch1* solution in Visual Studio, with the *test-setup* project highlighted

You should see and hear the program in Figure 1-1. (If not, see Appendix A.) You might take a moment to try another program from *ch1*, like *1-hello*. Run it the same way you did *test-setup*.

A simple program

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

So we'll begin 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. For now, sit tight.)

Example 1-1. "Hello, world!" is a classic program to start a new language with. (I think it's a law somewhere.) This program is in source code, in the *ch1* folder, as *1-hello*

```
// Hello, world! program, for _C++ for Lazy Programmers_
//   Your name goes here
//   Then the date4
```

⁴From here on, I'll put 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.