



C++20 Recipes

A Problem-Solution Approach

—

Second Edition

—

J. Burton Browning
Bruce Sutherland

apress®

C++20 Recipes

A Problem-Solution Approach

Second Edition

**J. Burton Browning
Bruce Sutherland**

Apress®

C++20 Recipes: A Problem-Solution Approach

J. Burton Browning
Bolivia, NC, USA

Bruce Sutherland
Carnegie, VIC, Australia

ISBN-13 (pbk): 978-1-4842-5712-8
<https://doi.org/10.1007/978-1-4842-5713-5>

ISBN-13 (electronic): 978-1-4842-5713-5

Copyright © 2020 by J. Burton Browning and Bruce Sutherland

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 designed by Raw Pixel (www.rawpixel.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 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/9781484257128. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

This book is dedicated to Zada Browning, my love and light.

Table of Contents

About the Authors.....xxi

About the Technical Reviewerxxiii

Acknowledgmentsxxv

Introductionxxvii

Chapter 1: Beginning C++ 1

 Recipe 1-1. Finding a Text Editor 2

 Problem 2

 Solution 2

 Recipe 1-2. Installing Clang on Ubuntu..... 4

 Problem 4

 Solution 4

 How It Works 4

 Recipe 1-3. Installing Clang on Windows..... 6

 Problem 6

 Solution 6

 How It Works 6

 Recipe 1-4. Installing Clang on macOS..... 8

 Problem 8

 Solution 8

 How It Works 8

 Recipe 1-5. Building Your First C++ Program..... 9

 Problem 9

 Solution 9

TABLE OF CONTENTS

Recipe 1-6. Debugging C++ Programs Using GDB in Cygwin or Linux.....	11
Problem	11
Solution	11
How It Works	12
Recipe 1-7. Debugging Your C++ Programs on macOS.....	16
Problem	16
Solution	16
How It Works	17
Recipe 1-8. Switching C++ Compilation Modes.....	19
Problem	19
Solution	19
How It Works	19
Recipe 1-9. Building with the Boost Library	20
Problem	20
Solution	20
How It Works	20
Recipe 1-10. Install Microsoft Visual Studio	22
Problem	22
Solution	22
How It Works	22
Chapter 2: Modern C++	25
Recipe 2-1. Initializing Variables.....	26
Problem	26
Solution	26
How It Works	26
Recipe 2-2. Initializing Objects with Initializer Lists	29
Problem	29
Solution	29
How It Works	29

Recipe 2-3. Using Type Deduction	32
Problem	32
Solution	33
How It Works	33
Recipe 2-4. Using auto with Functions	36
Problem	36
Solution	37
How It Works	37
Recipe 2-5. Working with Compile Time Constants	40
Problem	40
Solution	40
How It Works	40
Recipe 2-6. Working with Lambdas	44
Problem	44
Solution	44
How It Works	44
Recipe 2-7. Working with Time	55
Problem	55
Solution	55
How It Works	55
Recipe 2-8. Understanding lvalue and rvalue References	60
Problem	60
Solution	61
How It Works	61
Recipe 2-9. Using Managed Pointers.....	71
Problem	71
Solution	71
How It Works	71

TABLE OF CONTENTS

Chapter 3: Working with Text 81

 Recipe 3-1. Representing Strings in Code Using Literals 81

 Problem 81

 Solution 81

 How It Works 82

 Recipe 3-2. Localizing User-Facing Text 88

 Problem 88

 Solution 88

 How It Works 89

 Recipe 3-3. Reading Strings from a File 98

 Problem 98

 Solution 98

 How It Works 98

 Recipe 3-4. Reading the Data from an XML File 102

 Problem 102

 Solution 103

 How It Works 103

 Recipe 3-5. Inserting Runtime Data into Strings..... 110

 Problem 110

 Solution 110

 How It Works 110

Chapter 4: Working with Numbers 115

 Recipe 4-1. Using the Integer Types in C++ 115

 Problem 115

 Solution 115

 How It Works 116

 Recipe 4-2. Making Decisions with Relational Operators 121

 Problem 121

 Solution 121

 How It Works 122

Recipe 4-3. Chaining Decisions with Logical Operators	127
Problem	127
Solution	127
How It Works	127
Recipe 4-4. Using Hexadecimal Values.....	130
Problem	130
Solution	130
How It Works	130
Recipe 4-5. Bit Twiddling with Binary Operators	133
Problem	133
Solution	133
How It Works	133
Recipe 4-6. C++20 “Spaceship” or Three-Way Comparison Operator	143
Problem	143
Solution	143
How It Works	143
Chapter 5: Classes	147
Recipe 5-1. Defining a Class.....	147
Problem	147
Solution	147
How It Works	147
Recipe 5-2. Adding Data to a Class.....	149
Problem	149
Solution	149
How It Works	149
Recipe 5-3. Adding Methods.....	151
Problem	151
Solution	151
How It Works	152

TABLE OF CONTENTS

Recipe 5-4. Using Access Modifiers 154

 Problem 154

 Solution 154

 How It Works 154

Recipe 5-5. Initializing Class Member Variables 158

 Problem 158

 Solution 158

 How It Works 158

Recipe 5-6. Cleaning Up Classes 164

 Problem 164

 Solution 164

 How It Works 164

Recipe 5-7. Copying Classes..... 168

 Problem 168

 Solution 168

 How It Works 168

Recipe 5-8. Optimizing Code with *Move* Semantics 180

 Problem 180

 Solution 180

 How It Works 180

Recipe 5-9. Implementing Virtual Functions 186

 Problem 186

 Solution 186

 How It Works 186

Chapter 6: Inheritance 189

 Recipe 6-1. Inheriting from a Class 189

 Problem 189

 Solution 189

 How It Works 189

Recipe 6-2. Controlling Access to Member Variables and Methods in Derived Classes	192
Problem	192
Solution	192
How It Works	192
Recipe 6-3. Hiding Methods in Derived Classes	198
Problem	198
Solution	198
How It Works	198
Recipe 6-4. Using Polymorphic Base Classes	201
Problem	201
Solution	201
How It Works	201
Recipe 6-5. Preventing Method Overrides	205
Problem	205
Solution	205
How It Works	205
Recipe 6-6. Creating Interfaces	208
Problem	208
Solution	208
How It Works	208
Recipe 6-7. Multiple Inheritance	210
Problem	210
Solution	210
How It Works	211
Chapter 7: The STL Containers	215
Recipe 7-1. Storing a Fixed Number of Objects	215
Problem	215
Solution	215
How It Works	215

TABLE OF CONTENTS

Recipe 7-2. Storing a Growing Number of Objects 218

 Problem 218

 Solution 218

 How It Works 218

Recipe 7-3. Storing a Set of Elements That Is Constantly Altered 228

 Problem 228

 Solution 228

 How It Works 228

Recipe 7-4. Storing Sorted Objects in a Container That Enables Fast Lookups..... 231

 Problem 231

 Solution 231

 How It Works 232

Recipe 7-5. Storing Unsorted Elements in a Container for Very Fast Lookups 242

 Problem 242

 Solution 242

 How It Works 243

Recipe 7-6. Using C++20 Designated Initialization 248

 Problem 248

 Solution 248

 How It Works 248

Chapter 8: The STL Algorithms 251

 Recipe 8-1. Using an Iterator to Define a Sequence Within a Container..... 251

 Problem 251

 Solution 251

 How It Works 251

 Recipe 8-2. Calling a Function on Every Element in a Container..... 259

 Problem 259

 Solution 259

 How It Works 259

Recipe 8-3. Finding the Maximum and Minimum Values in a Container	261
Problem	261
Solution	261
How It Works	261
Recipe 8-4. Counting Instances of a Value in a Sequence	267
Problem	267
Solution	267
How It Works	268
Recipe 8-5. Finding Values in a Sequence	271
Problem	271
Solution	271
How It Works	271
Recipe 8-6. Sorting Elements in a Sequence	273
Problem	273
Solution	273
How It Works	273
Recipe 8-7. Looking Up Values in a <i>Set</i>	275
Problem	275
Solution	275
How It Works	275
Chapter 9: Templates	277
Recipe 9-1. Creating a Template Function	277
Problem	277
Solution	277
How It Works	277
Recipe 9-2. Partially Specializing a Template	282
Problem	282
Solution	282
How It Works	282

TABLE OF CONTENTS

Recipe 9-3. Creating Class Templates 289

 Problem 289

 Solution 289

 How It Works 289

Recipe 9-4. Creating Singletons 292

 Problem 292

 Solution 292

 How It Works 292

Recipe 9-5. Calculating Values at Compile Time..... 295

 Problem 295

 Solution 295

 How It Works 295

Recipe 9-6. Calculating Values at Compile Time..... 299

 Problem 299

 Solution 299

 How It Works 299

Chapter 10: Memory 301

 Recipe 10-1. Using Static Memory 301

 Problem 301

 Solution 301

 How It Works 301

 Recipe 10-2. Using Stack Memory..... 304

 Problem 304

 Solution 304

 How It Works 304

 Recipe 10-3. Using Heap Memory 309

 Problem 309

 Solution 309

 How It Works 309

Recipe 10-4. Using Automated Shared Memory	313
Problem	313
Solution	313
How It Works	313
Recipe 10-5. Creating Single-Instance Dynamic Objects	317
Problem	317
Solution	317
How It Works	317
Recipe 10-6. Creating Smart Pointers	322
Problem	322
Solution	322
How It Works	322
Recipe 10-7. Debugging Memory Problems by Overloading new and delete	332
Problem	332
Solution	332
How It Works	332
Recipe 10-8. Calculating Performance Impacts of Code Changes.....	341
Problem	341
Solution	341
How It Works	341
Recipe 10-9. Understanding the Performance Impacts of Memory Choices	343
Problem	343
Solution	344
How It Works	344
Recipe 10-10. Reducing Memory Fragmentation	348
Problem	348
Solution	348
How It Works	348

Chapter 11: Concurrency 365

 Recipe 11-1. Using Threads to Execute Concurrent Tasks..... 365

 Problem 365

 Solution 366

 How It Works 366

 Recipe 11-2. Creating thread Scope Variables..... 373

 Problem 373

 Solution 374

 How It Works 374

 Recipe 11-3. Accessing Shared Objects Using Mutual Exclusion 387

 Problem 387

 Solution 388

 How It Works 388

 Recipe 11-4. Creating Threads That Wait for Events..... 400

 Problem 400

 Solution 400

 How It Works 400

 Recipe 11-5. Retrieving Results from a Thread 408

 Problem 408

 Solution 408

 How It Works 408

 Recipe 11-6. Synchronizing Queued Messages Between Threads..... 413

 Problem 413

 Solution 413

 How It Works 414

Chapter 12: Networking..... 429

 Recipe 12-1. Setting Up a Berkeley Sockets Application on macOS..... 429

 Problem 429

 Solution 430

 How It Works 430

Recipe 12-2. Setting Up a Berkeley Sockets Application in Eclipse on Ubuntu	434
Problem	434
Solution	434
How It Works	434
Recipe 12-3. Setting Up a Winsock 2 Application in Visual Studio on Windows	439
Problem	439
Solution	439
How It Works	439
Recipe 12-4. Creating a Socket Connection Between Two Programs	445
Problem	445
Solution	446
How It Works	446
Recipe 12-5. Creating a Networking Protocol Between Two Programs	473
Problem	473
Solution	473
How It Works	474
Chapter 13: Scripting.....	497
Recipe 13-1. Running Lua Commands in Visual Studio C++	497
Problem	497
Solution	497
How It Works	498
Create and Open a Lua Script File in C++	500
Recipe 13-2. Creating a Lua Library Project in Eclipse.....	501
Problem	501
Solution	502
How It Works	502
Recipe 13-3. Creating a Lua Project in Xcode.....	504
Problem	504
Solution	504
How It Works	505

TABLE OF CONTENTS

Recipe 13-4. Using the Lua Programming Language 507

 Problem 507

 Solution 507

 How It Works 507

Recipe 13-5. Calling Lua Functions from C++ 519

 Problem 519

 Solution 519

 How It Works 519

Recipe 13-6. Calling C Functions from Lua..... 532

 Problem 532

 Solution 532

 How It Works 532

Recipe 13-7. Creating Asynchronous Lua Functions 539

 Problem 539

 Solution 539

 How It Works 539

Chapter 14: 3D Graphics Programming 547

Recipe 14-1. An Introduction to GLFW 547

 Problem 547

 Solution 547

 How It Works 548

Recipe 14-2. Rendering a Triangle..... 551

 Problem 551

 Solution 551

 How It Works 551

Recipe 14-3. Creating a Textured Quad 565

 Problem 565

 Solution 565

 How It Works 565

Recipe 14-4. Loading Geometry from a File	592
Problem	592
Solution	592
How It Works	592
Recipe 14-5. Working with C++20 Modules	615
Problem	615
Solution	615
How It Works	616
Index.....	621

About the Authors



Dr. J. Burton Browning earned his doctorate from North Carolina State University in 1999 under the advisement of Dr. Richard Peterson. He has conducted research in areas including distance learning, programming, and instructional technology. As a life-long learner and someone who has interests in topics such as programming, photography, robotics, car restoration, woodworking, hunting, reading, fishing, and archery, he is never at a loss for something to do. Dr. Browning's previous publications include works on cross-functional learning teams (CFLT), The Utopian School (teacher-led school model), computer programming (several

languages), open source software, healthcare statistics and data mining, CNC plasma cutter operation, educational technology, biography, mobile learning, online teaching, and more. Since retiring as a college professor in 2018, Burton is traveling and working on many automotive and other projects.

Bruce Sutherland is a video game programmer hailing from Dundee, Scotland. He graduated with a BSc (Hons) Computer Games Technology from the University of Abertay, Dundee, in Scotland, in 2005. After graduating, he began his first job in the game industry at 4J Studios where he worked on *Star Trek: Encounters* (PS2), *The Elder Scrolls IV: Oblivion* (PS3), *Star Trek: Conquest* (PS2, Wii), *Ducati Moto* (NDS), and *AMF Bowling Pinbusters!* (NDS). In July 2008, he moved from Dundee to Melbourne, Australia, where he joined Visceral Studios and was a software engineer on *Dead Space* (Xbox 360, PS3, PC), *The Godfather II* (Xbox 360, PS3, PC), and *Dead Space 3* (Xbox 360, PS3, PC). He developed an interest in developing for Android in his spare time and writes tutorials on his blog.

About the Technical Reviewer



Michael Thomas has worked in software development for more than 20 years as an individual contributor, team lead, program manager, and vice president of engineering. Michael has more than 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 healthcare providers.

Acknowledgments

The authors would like to acknowledge Steve Anglin, Matthew Moodie, and Mark Powers of Apress and the production team for their help and support. You are all fantastic to work with!

Introduction

The C++ programming language is undergoing continuous development and improvement. This effort to keep C++ on the cutting edge of language features is driven by the fact that C++ still finds an important role to play in high-performance, portable applications. Few other languages can be used on as many platforms as C++ without having a runtime environment dependency. This is partly thanks to the nature of C++ as a compiled programming language. C++ programs are built into application binaries through a combination of processes that include compiling and linking.

Compiler choice is particularly important in today's C++ landscape, thanks to the rate at which the language is changing. Development of the C++ programming language was started by Bjarne Stroustrup in 1979, when it was called C with Classes. The language didn't see formal standardization until 1998; an updated standard was published in 2003. There was another gap of eight years until the standard was updated again with the introduction of C++11 in 2011. This version brought a considerable number of updates to the C++ programming language and is distinguished from "older" C++ with the modern C++ moniker. C++ 17 and C++ 20 deprecated old features and brought many significant changes to the language.

This book introduces you to code written for the C++14 through C++20 ISO standard using both the Clang compiler, Microsoft Visual Studio (VS) 2019, and Xcode. Clang is an open source compiler that started life as a closed source Apple project. Apple released the code to the open source community in 2007, and the compiler has been adding strengths ever since. This book explains how to install and use Clang on a computer running OS X, Windows, or Linux (Ubuntu). The examples that accompany each chapter have been compiled and tested using Clang 3.5 and/or Visual Studio 2019. All of the listed applications are free, so choose which works best for your needs, or use them all to learn more!

The book's accompanying source code can be accessed via the **Download Source Code** link located at www.apress.com/9781484257128. You can find source code for all of the executable code listings contained in this book along with makefiles that can be used to build running programs.

CHAPTER 1

Beginning C++

The C++ programming language is a powerful low-level language that allows you to write programs that are compiled into machine instructions to be executed on a computer's processor. This makes C++ different from newer languages such as C# and Java. These languages are interpreted languages. This means they are not executed directly on the processor but instead are sent to another program that is responsible for operating the computer. Java programs are executed using the Java virtual machine (JVM), and C# programs are executed by the Common Language Runtime (CLR).

Thanks to C++ being a language that is compiled ahead of time, it still finds wide use in fields where absolute performance is paramount. The most obvious area where C++ is still the most predominantly used programming language is the video game industry. C++ allows programmers to write applications that take full advantage of the underlying system architecture. You might become familiar with phrases such as *cache coherency* while pursuing a career as a C++ programmer. There aren't many other languages that allow you to optimize your applications to suit the individual processors that your program is being designed to run on. This book introduces you to some of the pitfalls that can affect the performance of your applications at different times and shows you some techniques to tackle those issues.

Modern C++ is in a period where the language is seeing continual updates to its features. This has not always been the case. Despite being around since the early 1980s, the C++ programming language was only standardized in 1998. A minor update and clarification of this standard was released in 2003 and is known as C++03. The 2003 update did not add any new features to the language; however, it did clarify some of the existing features that had gone overlooked. One of these was an update to the standard for the Standard Template Library (STL) vector template to specify that the members of a vector should be stored contiguously in memory. The C++11 standard was released in 2011 and saw a massive update to the C++ programming language. C++ gained features for generalized type deduction system outside of templates, lambda, and closure support, a built-in concurrency library, and many more features. C++14 brought

a smaller update to the language and generally built upon the features already supplied by C++11. Features such as auto return type deduction from functions had been cleaned up, lambdas had been updated with new features, and there were some new ways to define properly typed literal values. C++ 17 introduced features such as *folds* and *static if* statements. C++20 now offers some powerful new features such as *Modules* and *Concepts* to name but a few of the enhancements to make the language even more powerful.

This book strives to write portable, standards-compliant C++20 code. At the time of writing, it's possible to use many of the new features of C++20 code on Windows, Linux, and macOS machines so long as you use a compiler that provides all of the language features. Since as of the time of writing all of the version 20 standards have not been formally agreed to nor implemented in many compilers, you may need to use more than one development tool to implement various features in your projects. To this end, this book will use three different tools: Clang as the compiler on Windows and Ubuntu and Xcode on macOS, with Microsoft Visual Studio 19 or higher on Windows and Mac platforms. The rest of this chapter focuses on the software you need to write programs in C++ before showing you how to acquire some of the more common options available for Windows, macOS, and Linux operating systems.

Recipe 1-1. Finding a Text Editor

Problem

C++ programs are constructed from lots of different source files that must be created and edited by one or more programmers. Source files are simply text files, which usually come in two different types: header files and source files. Header files are used to share information about your types and classes between different files, and source files are generally used to contain the methods and the actual executable code that makes up your program. With C++ version 20, programmers now have the opportunity to use Modules instead of traditional header files for faster build times and cleaner implementation design.

Solution

A text editor then becomes the first major piece of software you require to begin writing C++ programs. There are many excellent choices of text editors available on different platforms. My best two picks at the moment are the free Notepad++ for Windows and

Sublime Text 2, which despite not being free is available on all major operating systems. Figure 1-1 shows a screenshot from Sublime Text 2. Vim and gvim are also very good options that are available for all three operating systems. These editors provide many powerful features and are excellent choices for someone willing to learn.

Note Don't feel the urge to grab a text editor straight away. Some of the recipes later in this chapter cover integrated development environments (IDEs) that include all the software you need to write, build, and debug C++ applications. It is really a matter of user preference and platform to determine which is best for you. Try both using text editors and a comprehensive IDE (or two!) and see which fits your workflow the best. You will probably end up using several ultimately.

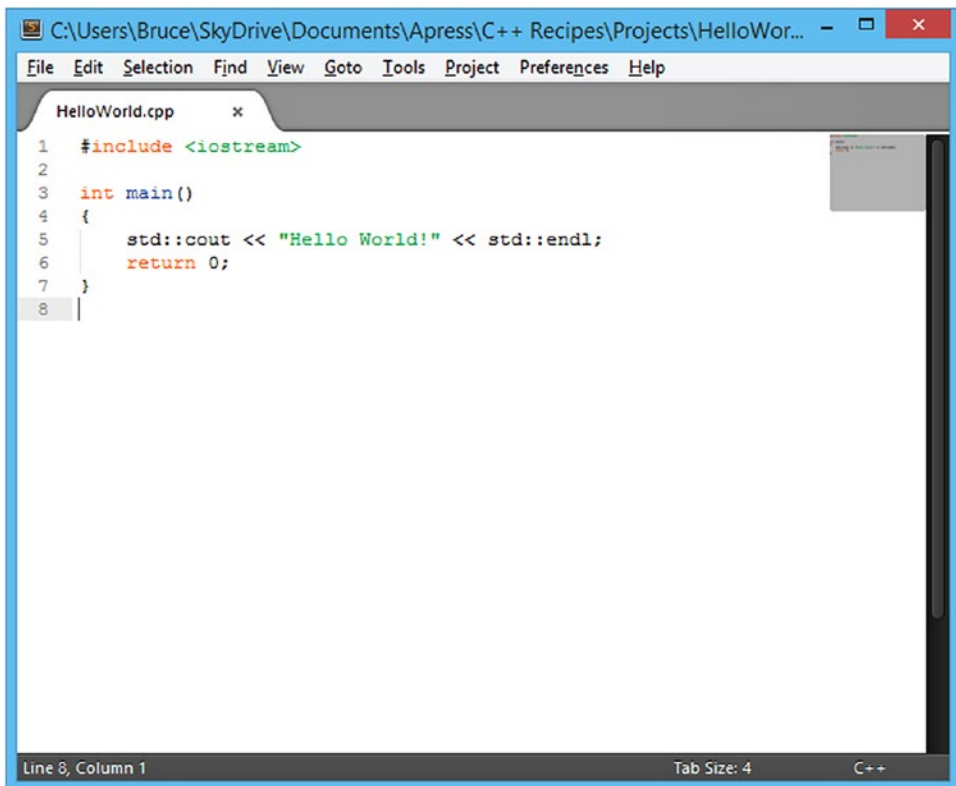


Figure 1-1. A screenshot from the Sublime text editor

Figure 1-1 shows one of the most important features of a good text editor: it should be able to highlight the different types of keywords in your source code. You can see in the simple Hello World program in Figure 1-1 that Sublime Text is capable of highlighting the C++ keywords `include`, `int`, and `return`. It has also added different-colored highlights to the main function name and the strings `<iostream>` and `"Hello World!"`. Once you have some experience writing code with your text editor of choice, you will become adept at scanning your source files to zero in on the area of code you are interested in, and syntax highlighting will be a major factor in this process.

Another feature some editors have is *autocomplete* or, as Microsoft calls it, *IntelliSense*. This feature is language-specific and speeds development time by autocompleting or auto-expanding options to functions and methods in the language. For a text editor such as Sublime with C++, you could install the Clang plugin for *ClangAutoComplete* for Sublime 3. Although not free, Sublime allows for continuous free evaluation time if you do not purchase a license.

Recipe 1-2. Installing Clang on Ubuntu

Problem

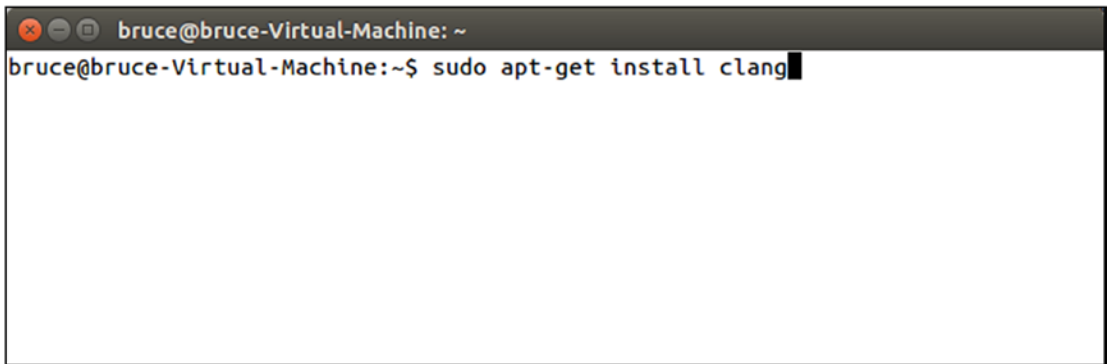
You would like to build C++ programs that support the latest C++20 language features on a computer system running Ubuntu.

Solution

The Clang compiler supports all of the latest C++20 language features, and the `libstdc++` library supports all of the C++20 STL features.

How It Works

The Ubuntu operating system comes configured with package repositories that allow you to install Clang without much difficulty. You can achieve this using the `apt-get` command in a terminal window. Figure 1-2 shows the command that you should enter to install Clang.



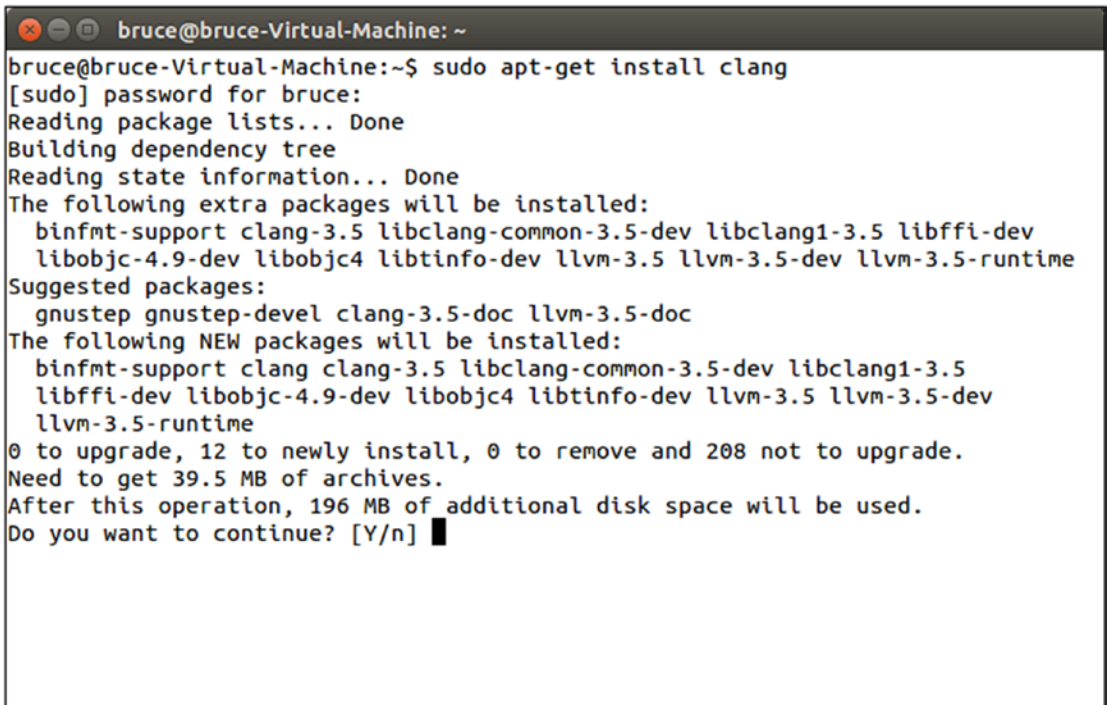
```

bruce@bruce-Virtual-Machine: ~
bruce@bruce-Virtual-Machine:~$ sudo apt-get install clang

```

Figure 1-2. An Ubuntu terminal window showing the command needed to install Clang

To install Clang, you can enter the following command on the command line: `sudo apt-get install clang`. Running this command will cause Ubuntu to query its repositories and work out all of the dependencies needed to install Clang. You will be prompted once this process has been completed to confirm that you wish to install Clang and its dependencies. You can see this prompt in Figure 1-3.



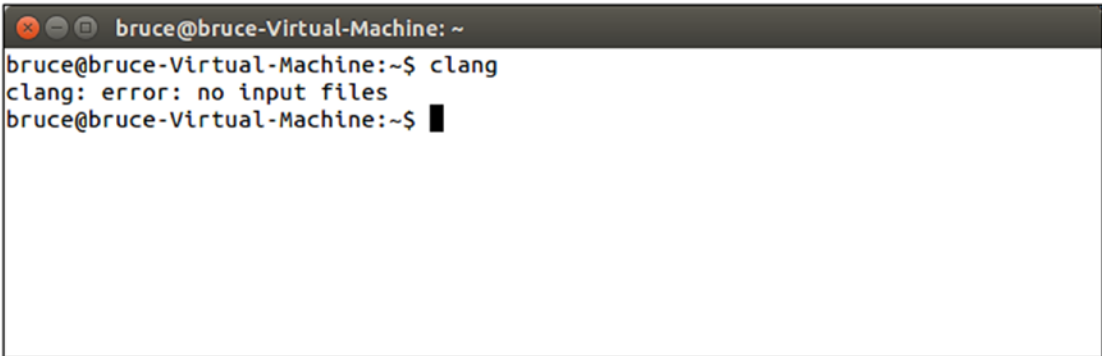
```

bruce@bruce-Virtual-Machine: ~
bruce@bruce-Virtual-Machine:~$ sudo apt-get install clang
[sudo] password for bruce:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  binfmt-support clang-3.5 libclang-common-3.5-dev libclang1-3.5 libffi-dev
  libobjc-4.9-dev libobjc4 libtinfo-dev llvm-3.5 llvm-3.5-dev llvm-3.5-runtime
Suggested packages:
  gnustep gnustep-devel clang-3.5-doc llvm-3.5-doc
The following NEW packages will be installed:
  binfmt-support clang clang-3.5 libclang-common-3.5-dev libclang1-3.5
  libffi-dev libobjc-4.9-dev libobjc4 libtinfo-dev llvm-3.5 llvm-3.5-dev
  llvm-3.5-runtime
0 to upgrade, 12 to newly install, 0 to remove and 208 not to upgrade.
Need to get 39.5 MB of archives.
After this operation, 196 MB of additional disk space will be used.
Do you want to continue? [Y/n]

```

Figure 1-3. The apt-get dependency confirmation prompt

At this point, you can press Enter to continue as yes is the default option. Ubuntu will then download and install all of the software needed for you to be able to install Clang on your computer. You can confirm that this has been successful by running the `clang` command. Figure 1-4 shows what this should look like if everything was successful.

A terminal window titled 'bruce@bruce-Virtual-Machine: ~' showing the command 'clang' being executed. The output is 'clang: error: no input files', which is a standard message when the compiler is run without any source files. The prompt returns to 'bruce@bruce-Virtual-Machine:~\$' with a cursor.

```
bruce@bruce-Virtual-Machine: ~  
bruce@bruce-Virtual-Machine:~$ clang  
clang: error: no input files  
bruce@bruce-Virtual-Machine:~$
```

Figure 1-4. A successful Clang installation in Ubuntu

Recipe 1-3. Installing Clang on Windows

Problem

You would like to build supported C++20-based programs on the Windows operating system.

Solution

You can use Cygwin for Windows to install Clang and build applications.

How It Works

Cygwin provides a Unix-like command-line environment for Windows computers. This is ideal for building programs using Clang as the Cygwin installed comes preconfigured with package repositories that include everything you need to install and use Clang on Windows computers.

You can get a Cygwin installer executable from the Cygwin web site at www.cygwin.com. Be sure to download the 32-bit version of the Cygwin installer as the default packages supplied by Cygwin currently only work with the 32-bit environment.

Once you have downloaded the installer, you should run it and click through until you are presented with the list of packages to install. At this point, you want to select the Clang, make, and libstdc++ packages. Figure 1-5 shows the Cygwin installer with the Clang package selected.

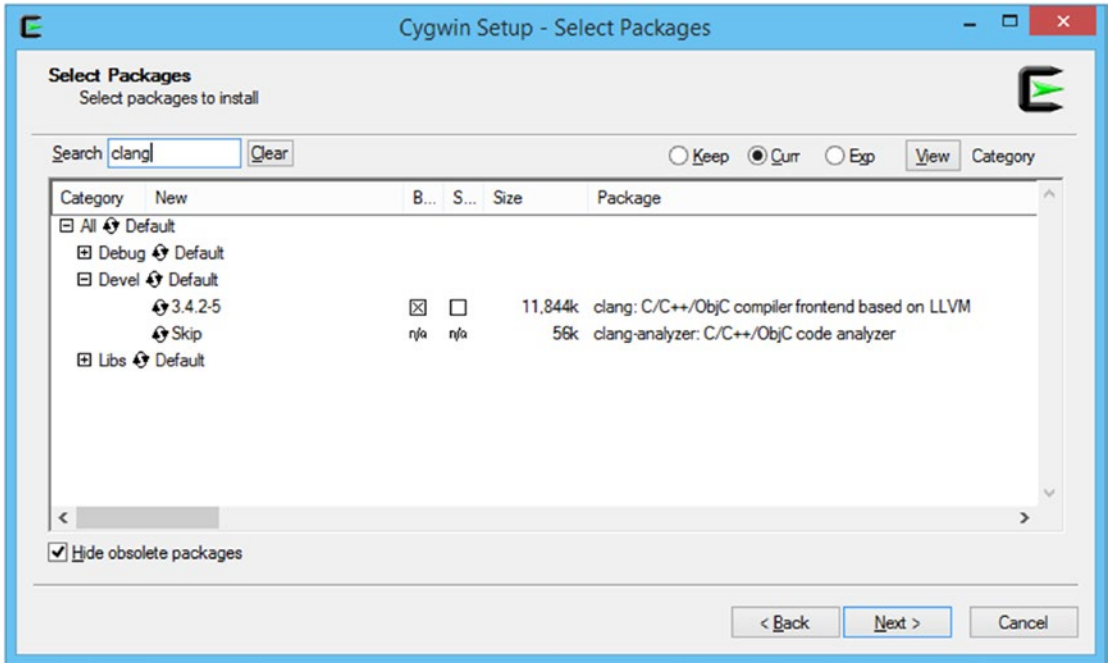


Figure 1-5. Filtering the Clang package in the Cygwin installer

Packages can be marked for installation in the installer by clicking the Skip area on the line for the package. Clicking Skip once moves the package version to the latest. You should select the latest packages for Clang, make, and libstdc++. Once you have selected all three, you can click Next to be taken to a window asking to confirm the installation of the dependencies needed by these three packages.

Once you have successfully downloaded and installed all of the packages that you needed to be able to run Clang, you can check that it was successful by opening a Cygwin terminal and typing the clang command. You can see the result of this output in Figure 1-6.