Exercises for Programming in C++

(Alpha Release, Version 2021-04-01)

Michael D. Adams

To obtain the most recent version of this book (with functional hyperlinks) or for additional information and resources related to this book (such as lecture slides, video lectures, and errata), please visit:

http://www.ece.uvic.ca/~mdadams/cppbook

If you like this book, **please consider posting a review** of it at:

https://play.google.com/store/search?q=ISBN:9780987919755 or
http://books.google.com/books?vid=ISBN9780987919755

youtube.com/iamcanadian1867 github.com/mdadams @mdadams16

Exercises for Programming in C++

(Alpha Release, Version 2021-04-01)

Michael D. Adams

Department of Electrical and Computer Engineering
University of Victoria
Victoria, British Columbia, Canada

This book was typeset with LaTeX.

ISBN 978-0-9879197-5-5 (PDF)

To my students, past, present, and future

Contents

List of Listings

License

Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. "Distribute" means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

d. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

e. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

f. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to

exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,
b. to Distribute and Publicly Perform the Work including as incorporated in Collections.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.
b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author,

Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt:

   i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
   ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,
   iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).

e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author’s honor or reputation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
e. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.

Preface

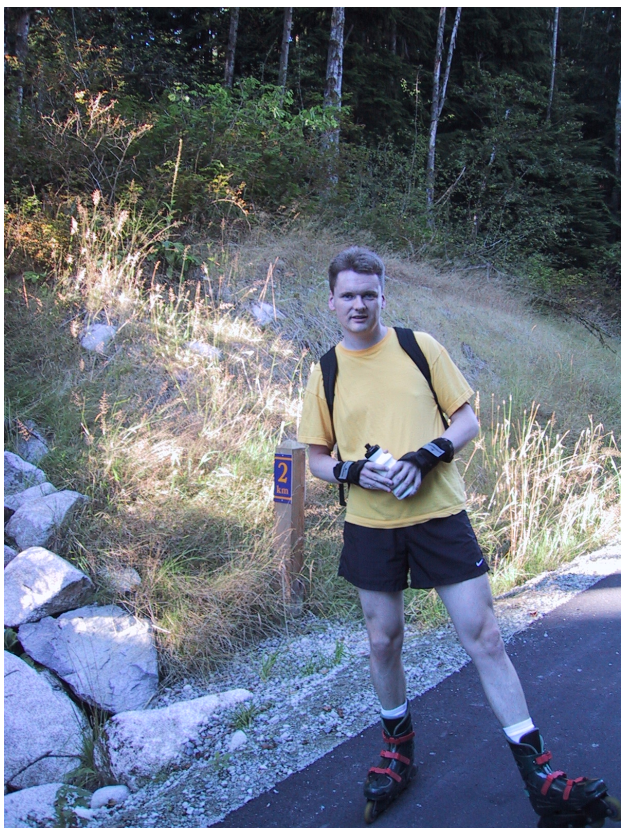
This book presents a study plan for learning C++ that is based on video lectures developed by the author. This book also provides a large set of programming and other exercises that relate to C++. This book evolved, in part, out of the need for a tool to assist in teaching C++ to the graduate students in the author's research group at the University of Victoria.

Acknowledgments

I would like to thank my past and present students for serving as test subjects for the use of this book. This has helped me to make numerous improvements to this book that would have otherwise not been possible (e.g., clarifying ambiguous exercise descriptions and correcting typographical errors).

Michael Adams
Victoria, BC
2021-04-01

About the Author



Michael Adams received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada in 1993, the M.A.Sc. degree in electrical engineering from the University of Victoria, Victoria, BC, Canada in 1998, and the Ph.D. degree in electrical engineering from the University of British Columbia, Vancouver, BC, Canada in 2002. From 1993 to 1995, Michael was a member of technical staff at Bell-Northern Research in Ottawa, ON, Canada, where he developed real-time software for fiber-optic telecommunication systems. Since 2003, Michael has been on the faculty of the Department of Electrical and Computer Engineering at the University of Victoria, Victoria, BC, Canada, first as an Assistant Professor and currently as an Associate Professor.

Michael is the recipient of a Natural Sciences and Engineering Research Council (of Canada) Postgraduate Scholarship. He has served as a voting member of the Canadian Delegation to ISO/IEC JTC 1/SC 29 (i.e., Coding of Audio, Picture, Multimedia and Hypermedia Information), and been an active participant in the JPEG-2000 standardization effort, serving as co-editor of the JPEG-2000 Part-5 standard and principal author of one of the first JPEG-2000 implementations (i.e., JasPer). His research interests include signal processing, image/video/audio processing and coding, multiresolution signal processing (e.g., filter banks and wavelets), geometry processing, and data compression.

Other Works by the Author

Some other open-access textbooks and slide decks by the author of this book include:

1. M. D. Adams, *Lecture Slides for Programming in C++ (Version 2021-04-01)*, University of Victoria, Victoria, BC, Canada, Apr. 2021, xxiii + 2901 slides, ISBN 978-0-9879197-4-8 (PDF). Available from Google Books, Google Play Books, and author's web site http://www.ece.uvic.ca/~mdadams/cppbook.

2. M. D. Adams, *Signals and Systems*, Edition 3.0, University of Victoria, Victoria, BC, Canada, Dec. 2020, xliv + 680 pages, ISBN 978-1-55058-673-2 (print), ISBN 978-1-55058-674-9 (PDF). Available from Google Books, Google Play Books, University of Victoria Bookstore, and author's web site http://www.ece.uvic.ca/~mdadams/sigsysbook.

3. M. D. Adams, *Lecture Slides for Signals and Systems*, Edition 3.0, University of Victoria, Victoria, BC, Canada, Dec. 2020, xvi + 625 slides, ISBN 978-1-55058-677-0 (print), ISBN 978-1-55058-678-7 (PDF). Available from Google Books, Google Play Books, University of Victoria Bookstore, and author's web site http://www.ece.uvic.ca/~mdadams/sigsysbook.

4. M. D. Adams, *Multiresolution Signal and Geometry Processing: Filter Banks, Wavelets, and Subdivision (Version 2013-09-26)*, University of Victoria, Victoria, BC, Canada, Sept. 2013, xxxviii + 538 pages, ISBN 978-1-55058-507-0 (print), ISBN 978-1-55058-508-7 (PDF). Available from Google Books, Google Play Books, University of Victoria Bookstore, and author's web site http://www.ece.uvic.ca/~mdadams/waveletbook.

5. M. D. Adams, *Lecture Slides for Multiresolution Signal and Geometry Processing (Version 2015-02-03)*, University of Victoria, Victoria, BC, Canada, Feb. 2015, xi + 587 slides, ISBN 978-1-55058-535-3 (print), ISBN 978-1-55058-536-0 (PDF). Available from Google Books, Google Play Books, University of Victoria Bookstore, and author's web site http://www.ece.uvic.ca/~mdadams/waveletbook.

Chapter 1

Introduction

1.1 Disclaimer

The book that you are currently reading represents a work in progress and should be considered an alpha release. It is not intended to be well polished. Some parts of this book clearly need improvement. Moreover, some important material is either not covered at all or has only very minimal coverage. For example, coverage of material related to the recently ratified C++20 standard is very minimal (although more coverage of such material will certainly be added in subsequent editions of this book). One of the few C++20 topics covered is text formatting via the video lecture mentioned in Section D.3.1.42. In spite of the above, it is believed that this book will be of benefit to some people. So, it is being made available in its current form. If you have any suggestions for improvements or other comments, your feedback would be most welcome. Please send any comments directly to the author. The author's contact information can be found on the web site for this book. For details about the web site, see Section 1.5.

1.2 Important Comment on Hyperlinks

If you obtained a PDF version of this book from Google Play Books or Google Books instead of the book's web site, please be aware that all of the hyperlinks in the document will have been removed by Google. Since these hyperlinks are critically important for navigating the content associated with this book, the author would very strongly recommend that all users of this book download the PDF version from the book's web site. (Again, for details about the book's web site, see Section 1.5.)

1.3 About This Book

To begin, this book presents a brief study plan for learning C++ based on video lectures prepared by the author and some supplemental reading. Then, the remainder of the book consists of a collection of exercises that relate to programming in C++. Most of the exercises focus on the C++ programming language, the C++ standard library, and several other commonly-used libraries. Some of the exercises, however, focus on the use of software tools, such as build tools (e.g., CMake) and version control tools (e.g., Git). In its current form, this book relies on video lectures (and some supplemental reading) for the delivery of instructional content. In the long term, however, the author plans to add instructional content (in written form) to this book, to yield a complete book on C++ programming.

1.4 Lecture Slides

This book is intended to be used in conjunction with the following (very large) set of lecture slides:

- M. D. Adams, *Lecture Slides for Programming in C++ (Version 2021-04-01)*, University of Victoria, Victoria, BC, Canada, Apr. 2021, xxiii + 2901 slides, ISBN 978-0-9879197-4-8 (PDF). Available from Google Books, Google Play Books, and author's web site http://www.ece.uvic.ca/~mdadams/cppbook.

1.5 Book Web Site

This book has an associated web site whose URL is:

- http://www.ece.uvic.ca/~mdadams/cppbook

To obtain the most recent version of this book (with functional hyperlinks) or for additional information and resources related to this book (such as lecture slides, video lectures, and errata), please visit this site.

1.6 Git Repository

This book has an associated Git repository containing some source code and other supplemental files. The URL for this repository is https://github.com/mdadams/cppbook_companion.git.

1.7 Virtual Machine (VM) Disk Images

The author has prepared a number of virtual machine (VM) disk images that each contain a basic software development environment that can be used for learning C++. This development environment includes recent versions of software such as: the GCC and Clang compiler tool chains, Git, CMake, GDB, Lcov, Boost, Catch2, GSL, YouCompleteMe (YCM), and Vim LSP. These VM disk images can be obtained from https://www.ece.uvic.ca/~mdadams/cppbook/#vm_disk_images.

1.8 Study Plan

The instructional content for this book is delivered exclusively via video lectures and various supplemental reading material. To facilitate easier learning, a study plan for learning C++ is provided. This study plan consists of viewing video lectures and completing exercises related to those lectures. The video lectures in this study plan are described in detail in Appendix D. The study plan consists of covering the following items (in order):

1. Software Development Environment (SDE)
   - Watch the following video lectures:
     - Accessing the SDE Using VM Software [2020-04-26] (Section D.4.1.2)
2. Version Control Systems and Git
   - Watch the following video lectures:
     - Version Control — Introduction [2017-04-06] (Section D.3.1.2)
     - Git — Introduction [2017-04-08] (Section D.3.1.3)
     - Git — Demonstration [2017-04-05] (Section D.3.1.4)
   - Complete all of the Git exercises from Appendix C.
3. Compiling and Linking
   - Watch the following video lectures:
     - Getting Started — Compiling and Linking [2017-04-13] (Section D.3.1.1)
4. Build Systems
   - Watch the following video lectures:
     - Build Systems — Introduction [2017-04-12] (Section D.3.1.5)
     - CMake — Introduction [2017-04-16] (Section D.3.1.7)
     - CMake — Examples [2017-04-18] (Section D.3.1.8)

    - Assertions and CMake Build Types Demonstration [2020-04-30] (Section D.4.1.3)
  - Complete all of the CMake exercises from Appendix B.
  - Optionally, watch the following video lectures:
    - Make — Introduction [2017-04-12] (Section D.3.1.6)

    Although the author strongly recommends the use of CMake over Make, it is still likely to be helpful to have an understanding of Make (since CMake often uses Make as the underlying native build system).
5. Overview of C++
   - Read the following chapters from [27]:
     - Chapter 1: Notes to Reader
     - Chapter 2: A Tour of C++: The Basics
     - Chapter 3: A Tour of C++: Abstraction Mechanisms
     - Chapter 4: A Tour of C++: Containers and Algorithms
     - Chapter 5: A Tour of C++: Concurrency and Utilities

     At the time of this writing, the above chapters are freely available at `https://isocpp.org/tour`.
6. Software Testing Tools
   - Watch the following video lectures:
     - Address Sanitizer (ASan) Demonstration [2020-04-26] (Section D.4.1.4)
     - Undefined-Behavior Sanitizer (UBSan) Demonstration [2020-04-26] (Section D.4.1.5)
     - Lcov Demonstration [2020-04-30] (Section D.4.1.6)
7. Basics
   - Watch the following video lectures:
     - Basics — Introduction [2015-04-06] (Section D.3.1.9)
     - Basics — Objects, Types, and Values [2015-04-08] (Section D.3.1.10)
     - Basics — Operators and Expressions [2016-03-20] (Section D.3.1.11)
     - Basics — Control-Flow Constructs [2015-04-09] (Section D.3.1.12)
     - Basics — Functions [2016-03-20] (Section D.3.1.13)
     - Basics — Input/Output [2016-03-21] (Section D.3.1.14)
     - Basics — Miscellany [2016-03-21] (Section D.3.1.15)
   - Read the article [23]. This article provides a detailed tutorial on type declarations. The syntax used in type declarations can often be a source of confusion to those learning C++. This article will help to greatly reduce the potential for confusion in this regard.
   - Complete all of the exercises from Chapter 2.
   - All exercises (here and subsequently) should be **built using CMake**.
   - All exercises (here and subsequently) should normally be **built with ASan and UBSan enabled**.
   - If additional reference material is needed, refer to the following chapters from [27]:
     - Chapter 6: Types and Declarations
     - Chapter 7: Pointers, Arrays, and References
     - Chapter 8: Structures, Unions, and Enumerations
     - Chapter 9: Statements
     - Chapter 10: Expressions
     - Chapter 11: Select Operations
     - Chapter 12: Functions
     - Chapter 14: Namespaces
     - Chapter 15: Sources Files and Programs
8. Classes
   - Watch the following video lectures:
     - Classes — Introduction [2016-03-05] (Section D.3.1.16)

  - Classes — Members and Access Specifiers [2016-03-05] (Section D.3.1.17)
  - Classes — Constructors and Destructors [2016-03-06] (Section D.3.1.18)
  - Classes — Operator Overloading [2016-03-09] (Section D.3.1.19)
  - Classes — More on Classes [2016-03-22] (Section D.3.1.20)
  - Classes — Temporary Objects [2016-03-24] (Section D.3.1.21)
  - Classes — Functors [2016-03-24] (Section D.3.1.22)
- Complete all of the exercises from Chapter 3.
- If additional reference material is needed, refer to the following chapters from [27]:
  - Chapter 16: Classes
  - Chapter 17: Construction, Cleanup, Copy, and Move
  - Chapter 18: Overloading
  - Chapter 19: Special Operators

9. Templates
   - Watch the following video lectures:
     - Templates — Introduction [2016-03-14] (Section D.3.1.23)
     - Templates — Function Templates [2016-03-17] (Section D.3.1.24)
     - Templates — Class Templates [2016-03-17] (Section D.3.1.25)
     - Templates — Variable Templates [2016-03-14] (Section D.3.1.26)
     - Templates — Alias Templates [2016-03-14] (Section D.3.1.27)
   - Complete all of the exercises from Chapter 4.
   - If additional reference material is needed, refer to the following chapters from [27]:
     - Chapter 23: Templates
     - Chapter 24: Generic Programming
10. Standard Library
    - Watch the following video lectures:
      - Standard Library — Introduction [2016-03-30] (Section D.3.1.28)
      - Standard Library — Containers, Iterators, and Algorithms [2016-04-05] (Section D.3.1.29)
      - Standard Library — The vector Class Template [2016-03-30] (Section D.3.1.30)
      - Standard Library — The basic_string Class Template [2016-04-01] (Section D.3.1.31)
      - Standard Library — Time Measurement [2016-04-02] (Section D.3.1.32)
      - Text Formatting in C++20 [2021-02-03] (Section D.3.1.42)
    - Complete all of the exercises from Chapter 5.
    - If additional reference material is needed, refer to the following chapters from [27]:
      - Chapter 30: Standard Library Summary
      - Chapter 31: STL Containers
      - Chapter 32: STL Algorithms
      - Chapter 33: STL Iterators
      - Chapter 34: Memory and Resources
      - Chapter 35: Utilities
      - Chapter 36: Strings
      - Chapter 37: Regular Expressions
      - Chapter 38: I/O Streams
      - Chapter 39: Locales
      - Chapter 40: Numerics
11. Exceptions
    - Watch the parts of the video lectures from Lectures 13–15 for SENG 475 (in Section D.2).

- Complete all of the exercises from Chapter 6.
- If additional reference material is needed, refer to the following chapters from [27]:
  - Chapter 13: Exception Handling

12. Miscellaneous Topics
    - Complete all of the exercises from Chapter 8.
13. Classes Revisited: Inheritance [Can Be Deferred Until Later]
    - Read the following chapters from [27]:
      - Chapter 20: Derived Classes
      - Chapter 21: Class Hierarchies
      - Chapter 22: Run-Time Type Information
    - This material is not currently covered by the video lectures.
14. Templates Revisited [Can Be Deferred Until Later]
    - Read the following chapters from [27]:
      - Chapter 25: Specialization
      - Chapter 26: Instantiation
      - Chapter 27: Templates and Hierarchies
      - Chapter 28: Metaprogramming
      - Chapter 29: A Matrix Design
    - This material is not currently covered by the video lectures.
15. Concurrency
    - Watch the following video lectures:
      - Concurrency — Preliminaries [2015-02-12] (Section D.3.1.33)
      - Concurrency — Threads [2015-02-17] (Section D.3.1.34)
      - Concurrency — Mutexes [2015-02-23] (Section D.3.1.35)
      - Concurrency — Condition Variables [2015-02-27] (Section D.3.1.36)
      - Concurrency — Promises and Futures [2015-04-02] (Section D.3.1.37)
    - Note that much of this material is also covered in Lectures 25–31 for SENG 475 (in Section D.2).
    - Complete all of the exercises from Chapter 7.
    - If additional reference material is needed, refer to the following chapters from [27]:
      - Chapter 41: Concurrency
      - Chapter 42: Threads and Tasks

      For much more detailed coverage of concurrency in C++, the book by Williams [33] is very highly recommended.
16. CGAL
    - Read the reference material on CGAL identified in Appendix A.
    - Watch the following video lectures:
      - CGAL — Introduction [2015-06-29] (Section D.3.1.38)
      - CGAL — Polygon Meshes [2015-07-02] (Section D.3.1.39)
      - CGAL — Subdivision Surface Methods [2015-06-29] (Section D.3.1.40)
      - CGAL — Example Programs [2015-07-01] (Section D.3.1.41)
      - Meshlab/Geomview Demo [2019-06-16] (Section D.4.1.1)
    - Complete all of the exercises from Appendix A.

Chapter 2

Basics

2.1 Exercises

Objects, Types, and Values

2.1 For each of the declarations below, do the following. State whether the declaration is also a definition. If the declaration is not a definition, write a definition for the entity being declared. If the declaration is a definition, write a declaration for the entity being declared that is not also a definition.

(a) `int i = 5;`

(b) `int abs(int x);`

(c) `float sqr(float x) {return x * x;}`

(d) `extern const double pi;`

(e) `char s[] = "Hello";`

(f) `double x;`

(g) `int (*func)(int, int);`

(h) `template <typename T> T max(T x, T y);`

(i) `template <typename T> bool is_negative(T x) {return x < 0;}`

(j) `auto meaning_of_life = 42;`

2.2 For each of the declarations below, do the following. State whether the declaration is also an initialization. If it is not an initialization, modify it so that it is one.

(a) `int x;`

(b) `void (*f)();`

(c) `const double pi = 3.14;`

2.3 State whether each line of code below corresponds to a type declaration or definition.

(a) `struct Point {double x; double y;};`

(b) `struct Thing;`

(c) `enum Fruit : int {apple, orange, banana};`

(d) `enum Color : int;`

2.4 Write a declaration for each of the entities listed below. Initialize each one. Do not use the null pointer in any initializations.

(a) a pointer to a **char**

(b) a pointer to a constant **char**

(c) a constant pointer to a **char**

(d) a constant pointer to a constant **char**

(e) a pointer to a function taking a **double** parameter and returning an **int**

(f) a pointer to a pointer to an **int**

(g) an lvalue reference to an array of 16 **int**s

(h) a pointer to an array of 10 elements of type `std::string`

(i) an lvalue reference to an array of 8 **int**s

2.5 Explain what is potentially wrong with the line of code below.

```
char c = -1;
```

2.6 For each of the conditions below, state whether the condition must be true, must be false, or could be either true or false.

(a) `sizeof(char) == 1`

(b) `sizeof(int) == 2 || sizeof(int) == 4`

(c) `sizeof(short) < sizeof(int) && sizeof(int) < sizeof(long) && sizeof(long) < sizeof(long long)`

(d) `sizeof(short) <= sizeof(int) && sizeof(int) <= sizeof(long) && sizeof(long) <= sizeof(long long)`

2.7 Using **typedef**, create a type alias for each of the types listed below.

(a) a pointer to a **char**

(b) a pointer to a **const char**

(c) a **const** pointer to a **char**

(d) a pointer to a function taking a **float** parameter and returning an **int**

(e) a pointer to an array of 16 elements of an array of 8 elements of type **long**

(f) an lvalue reference to an array of 8 **int**s

(g) an rvalue reference to a pointer to an **int**

2.8 Repeat the previous problem with a **using** statement instead of a **typedef** statement.

2.9 In the code given below, state the type of each of following objects: `a`, `b`, `c`, `d`, `e`, and `f`.

```
1  int main()
2  {
3      const int i = 42;
4      int j = 0;
5      auto a = i;
6      auto b = j;
7      decltype(i) c = 0;
8      decltype((i)) d = 0;
9      decltype(j) e;
10     decltype(&i) f;
11 }
```


2.10 For each of the literals given below, state its type.

(a) `123`

(b) `3.14`

(c) `1.0f`

(d) `"Hello, World!\n"`

(e) `0UL`

2.11 Each of the code fragments below contains an error. In each case, explain what the error is, why it is an error, and how the error can be fixed.

(a)
```
const char a = 'a';
char *c = &a;
```

(b)
```
const int i = 42;
auto& j = i;
++j;
```

2.12 What is the size of the character array `s` below and what is the length of the character string in this array as returned by the `strlen` function?

```
char s[] = "Hello";
```

2.13 In the code given below, state the type of each of following objects: `a`, `b`, `c`, `d`, `e`, `f`, `g`, and `h`.

```
1  int main()
2  {
3      const int ci = 42;
4      int i = ci;
5      auto& a = i;
6      auto& b = ci;
7      auto&& c = 0;
8      const auto&& d = 0;
9      auto e = &ci;
10     auto f = &i;
11     auto const g = &i;
12     const auto h = &i;
13 }
```

Operators and Expressions

2.14 Fully parenthesize each of the expressions given below.

(a) `a = b + c * d >> 1 & 2`

(b) `a == 0 && b != 0 || c < 0`

(c) `a & 15 != 15`

(d) `a++, b = a`

(e) `0 <= i < 8`

(f) `a = b = c = 0`

(g) `a[2][1] *= f(1, 2) + 1`

(h) `a << b << c << d`

(i) `c = a < 0`

(j) `a+++b + a + ++b`

(k) `a *= * b < 0 ? - * b : * b`

2.15 For each expression identified by a comment in the code below, state whether the expression is an lvalue or rvalue.

```
int abs(int);

void func()
{
    int x = 0;
    int y = 0;
    int z;

    ++x;
      // x
      // ++x
    y++;
      // y++
    z = x + y;
      // x + y
      // z = x + y
    z = abs(x + 1);
      // abs(x + 1)
    x = y;
      // x = y
}
```

Control-Flow Constructs and Functions

2.16 Write a program that prints to standard output the size and alignment of each of following types: **`int`**, **`long`**, **`float`**, and **`double`**. The program should also print similar information for the type corresponding to a pointer to each of these types.

2.17 Consider the execution of the program whose source listing is given below. For each line of the source code marked by a comment `/* ??? */` indicate the value of the objects `x`, `y`, and `z`, after the line of code completes execution.

```
int main()
{
    int x = 0;
    int y = 1;
    int& z = x; /* ??? */
    x = y;      /* ??? */
    y = z + 1; /* ??? */
    ++z;       /* ??? */
}
```


2.18 Write a function called `strlen` that takes a pointer to the first character of a C-style string (i.e., a null-terminated string) and returns an **`int`** indicating the number of characters in the string (not counting the terminating null character).

2.19 Write a function called `swap` that exchanges the values of two **`int`** objects, such that the two objects to be exchanged are passed to the function using:

(a) two pointers to **`int`** objects

(b) two references to **`int`** objects

Which of these two approaches is preferable? Explain your answer.

2.20 State the output that will be produced by the execution of the program listed below. Explain your answer.

```
1  #include <iostream>
2
3  void increment(int x)
4  {
5      ++x;
6  }
7
8  int main()
9  {
10     int i = 0;
11     increment(i);
12     std::cout << i << "\n";
13 }
```

2.21 Write a program that outputs the lowercase letters (i.e., a to z) and decimal digits (i.e., 0 to 9) and their corresponding integer values. (Note: The C++ language standard does not require that lower case letters be numbered consecutively. The decimal digits, however, must be numbered consecutively.)

2.22 Write a program to output the smallest and largest values of the following types: **`char`**, **`short`**, **`int`**, **`long`**, **`long long`**, **`float`**, **`double`**, **`long double`**, and **`unsigned int`**. (Hint: Use `std::numeric_limits`.)

2.23 Consider a dataset that consists of a sequence of records, where each record consists of the following two fields: 1) a name, which is a string (containing no whitespace characters) and 2) a value, which is a real number. Records and fields within records are delimited by whitespace. The same name can appear in multiple records. The number of records in the dataset may be extremely large (e.g., quadrillions of records). Develop a program that reads the above type of dataset from standard input and writes the following information to standard output: 1) the number of (distinct) names; 2) for each name, the minimum, maximum, and average of the values for all of the records with that particular name; and 3) the minimum, maximum, and average of the values for all of the records. (Hint: Use `std::map`.)

2.24 (a) Write a function `str_to_int` that converts a C-style string representing a signed integer (i.e., a string consisting of a possible plus or minus sign followed by one or more digits) to its corresponding numeric value. The function should take a single parameter that is a pointer to the C-style string of digits and return an **`int`**. If the string is not formatted correctly, the value zero should be returned.

(b) Write a program to test the `str_to_int` function. Also, test the `str_to_int` function with the program below.


```
#include <iostream>

// Place the code for the str_to_int function here.

int main()
{
    const char s[] = "123";
    std::cout << str_to_int(s) << "\n";
}
```

2.25 (a) Write a function `str_concat` that takes two C-style strings as parameters and returns the concatenation of these two strings as a C-style string. Use **`new`** to obtain the memory to hold the concatenated string.

(b) The interface provided by the `str_concat` function has an important shortcoming that is likely to lead bugs in practice. Identify this shortcoming and the type of bug that is likely to arise.

2.26 (a) Write a function `copy_ints` that copies a specified number of **`int`**s from one area of memory to another. The function should have a **`void`** return type and take the following parameters: 1) a pointer specifying the start of the source area for the copy operation; 2) an integral type specifiying how many **`int`**s to copy; and 3) a pointer specifying the start of the destination area for the copy operation.

(b) Write a program to test the `copy_ints` function. Also, test the function with the program below.

```
1  #include <iostream>
2  #include <cassert>
3
4  // Place the code for the copy_ints function here.
5
6  int main()
7  {
8      const int src[4] = {1, 2, 3, 4};
9      int dst[4] = {0, 0, 0, 0};
10     copy_ints(src, 4, dst);
11     assert(!memcmp(src, dst, sizeof(src)));
12 }
```

2.27 Consider the following types: i) a function taking arguments of type pointer to **`char`** and lvalue reference to **`int`** and returning no value; ii) a pointer to a function of the type in (i); iii) a function taking a pointer of the type in (ii) as an argument and returning no value; and iv) a function taking no arguments and returning a pointer to a function of the type in (iii). Write a declaration for each of the preceding types. Write the definition of a function `func` that takes as an argument a pointer to a function of the type in (iv) and returns its argument as the return value (without any type conversion).

2.28 Develop a program that writes the contents of one or more files in succession to standard output. (That is, the program writes the concatenation of a number of files to standard output.) The sequence of files to be processed should be specified as command-line arguments (i.e., via the arguments passed to the `main` function).

2.29 Consider the code for the function given below. Identify the one very serious bug in this code. Explain how this bug could be fixed.


```
#include <cmath>

double& get_value(double x)
{
    double result = 0.0;
    for (int i = 0; i < 100; ++i) {
        result += exp(x + i / 100.0);
    }
    return result;
}
```

Namespaces

2.30 (a) Consider the program whose source listing is given below. Modify this program so that the scope resolution operator `::` (e.g., as appears in `std::cout`) is no longer needed. Resist the temptation to commit the cardinal sin of writing "**`using namespace`** `std;`".

```
#include <iostream>

int main()
{
    std::cout << "Hello, World!\n";
}
```

(b) Explain why a using a construct like "**`using namespace`** `std;`" is truly evil.

2.31 Write your own implementation of the `strlen` function from the standard library. To avoid naming conflicts, place your `strlen` function in the namespace `mine`. Write a program that reads a line of text from standard input and then prints the length of the line of text (in characters) using your `strlen` function and also the one from the standard library. (Of course, both functions should yield the same answer.)

Preprocessor

2.32 What is either erroneous or potentially problematic with each of the following preprocessor macro definitions?

(a) **`#define`** `multiply(x, y) x * y`

(b) **`#define`** `maximum(x, y) x > y ? x : y`

(c) **`#define`** `factorial(x) ((x) * factorial((x) - 1))`

2.33 For the particular compiler that you are using, find the directory in which standard library header files (such as `iostream`) are kept.

2.34 An include guard is a construct used to avoid the potential problems caused by multiple inclusion of header files resulting from the use of the **`#include`** directive. There are two types of include guards: internal and external. An external include guard performs a test outside the header file it is guarding and includes it only once per compilation. An internal include guard performs a test inside the header file it is guarding. Write a simple program using a single header file with an internal include guard. Modify the code to use an external include guard. Discuss the advantages and disadvantages of each of these two approaches.

2.35 Preprocessor macros have many shortcomings. Explain how macros do not interact well with namespaces.

Chapter 3

Classes

3.1 Exercises

Classes; Construction, Destruction, and Operator Overloading

3.1 Develop a class `Counter` that represents a simple integer counter. The class should satisfy the following requirements:

(a) A constructor should be provided that takes a single **`int`** argument that is used to initialize the counter value. The argument should default to zero.

(b) The prefix increment and postfix increment operators should be overloaded in order to provide a means by which to increment the counter value.

(c) A member function `getValue` should be provided that returns the current counter value.

In addition, the class must track how many `Counter` objects are currently in existence. A means for querying this count should be provided. The code must not use any global variables. (Hint: Use static members.)

3.2 (a) Define a class `Exponent` to hold a real exponent for an exponentiation operation. Through clever use of operator overloading, find a way to have the expression `x ** y` call `std::pow(x, y)`, where the types of `x` and `y` are **`double`** and `Exponent`, respectively. In other words, the program listed below when executed would output the value 16 (i.e., $2^4 = 16$).

```
#include <iostream>
#include <cmath>

// Insert your code here.

int main()
{
    const double x = 2.0;
    const Exponent y = 4.0;

    std::cout << x ** y << "\n";
}
```

(b) Is the preceding use of operator overloading wise? Explain.

3.3 (a) Develop a class `Histogram` for performing histogram calculations (i.e., counting how many values fall in each of a number of intervals). The class should satisfy the following requirements:

i. A constructor should be provided that takes a single argument specifying a `std::vector` of **`double`**s containing the bounds of the histogram bins. The elements of the `std::vector` must be strictly monotonically increasing. For example, invoking the constructor with the argument

```
std::vector<double>{0.0, 3.14, 20.0, 42.42}
```

would create a histogram with three bins, corresponding to the intervals $[0, 3.14)$, $[3.14, 20)$, and $[20, 42.42)$.

ii. A member function `clear` should be provided that clears the histogram statistics.

iii. A member function `update` should be provided that takes a new data value to be added to the histogram statistics. This function should be able to handle out-of-range data in some appropriate manner.

iv. A member function `display` should be provided that outputs the contents of the histogram to a given output stream (i.e., `std::ostream`) in some human-readable format.

v. The class should not be default constructable (i.e., no default constructor should be provided).

vi. The class should provide move and copy constructors, move and copy assignment operators, and a destructor.

vii. All data members should be private.

(b) Write a program to thoroughly test your `Histogram` class.

3.4 (a) Develop a class `Integer` that behaves similar to the built-in integer type **`int`**, except that: 1) the meaning of addition and subtraction are reversed; and 2) the meaning of multiplication and division are reversed. The `Integer` class should satisfy the following requirements:

i. A constructor should be provided that takes a single **`int`** argument that is used to initialize the `Integer`. The argument should default to zero.

ii. The class should provide move and copy constructors, move and copy assignment operators, and a destructor.

iii. The class should overload all of the following operators: addition, subtraction, multiplication, division, `+=`, `-=`, `*=`, and `/=`.

iv. A stream inserter should be provided to allow an `Integer` to be written to an output stream (i.e., `std::ostream`).

v. A stream extractor should be provided to allow an `Integer` to be read from an input stream (i.e., `std::istream`).

vi. All data members should be private.

(b) Write a program to test your `Integer` class.

(c) Also, test your class with the program below. The program should output a value of 3.

```
#include <iostream>

// Insert the code for the Integer class here.

int main()
{
   const Integer x = 1;
   const Integer y = 2;
   const Integer z = 3;

   // Since the meaning of + and - are reversed and
   // the meaning of * and / are reversed,
   // the line of code following this comment
   // effectively computes:
   //    (x - y) / (x - y) * z
   std::cout << (x + y) * (x + y) / z << "\n";
}
```


(d) Although the class that you have just developed is perfectly valid C++ code, it is pure evil. Explain why.

3.5 (a) Develop a class `Rational` that represents a rational number (i.e., a number of the form x/y where x and y are integers and $y \neq 0$). The class should satisfy the following requirements:

i. A type member `Integer` should be provided that corresponds to the integer type used to represent the numerator and denominator of the rational number.

ii. A constructor should be provided that takes two arguments corresponding to the numerator and denominator values of the rational number, respectively. The first argument should default to zero. The second argument should default to one.

iii. The class should provide move and copy constructors, move and copy assignment operators, and a destructor.

iv. The addition, subtraction, multiplication, and division operators should be provided.

v. The `+=`, `-=`, `*=`, and `/=` operators should be provided.

vi. A member function `toDouble` should be provided that returns the best approximation of the rational number as a **`double`**.

vii. A stream inserter should be provided to write a rational number to an output stream (`std::ostream`) using a format like "-15/23".

viii. All data members should be private.

In order to simplify the implementation, you do not need to maintain a rational number in reduced form (so that the numerator and denominator are coprime).

(b) Write a program to thoroughly test your `Rational` class.

(c) Explain why maintaining a rational number in reduced form (i.e., such that the numerator and denominator are coprime) would be desirable in many applications (although this is not done in this exercise).

3.6 (a) In this problem, you will develop a class `RealPoint2` that represents a point in two dimensions with **`double`** (i.e., real) coordinates. This class should provide an interface like that shown below.

```
1  class RealPoint2
2  {
3  public:
4     // The coordinate type, in case we want to change it later.
5     typedef double Real;
6
7     // Create a new point with coordinates (0, 0).
8     RealPoint2();
9
10    // Create a new point with coordinates (x, y).
11    RealPoint2(const Real& x, const Real& y);
12
13    // Obtain a reference to the x-coordinate of the point.
14    Real& x();
15    const Real& x() const;
16
17    // Obtain a reference to the y-coordinate of the point.
18    Real& y();
19    const Real& y() const;
20
21    // Add/subtract displacement to/from point (i.e., translate point).
22    RealPoint2& operator+=(const RealPoint2& p);
23    RealPoint2& operator-=(const RealPoint2& p);
24
```



```
25 };
26
27 // Write a point to an output stream.
28 std::ostream& operator<<(std::ostream& out, const RealPoint2& p);
29
30 // Read a point from an input stream.
31 std::istream& operator>>(std::istream& in, RealPoint2& p);
```

(b) Write a program that reads points from standard input, translates them by $(1.5, -1.5)$, and writes them to standard output. The program should terminate when EOF is reached. Use `+=` or `-=` to perform the translation.

(c) The `x` and `y` member functions of the `RealPoint2` class return references. Explain why such a practice might be undesirable.

3.7 Develop a class `IntStack` that represents a stack containing **`int`** elements. Provide the basic functionality that would normally be expected from a stack (e.g., push an element on the stack, pop an element off the stack, get the top element on the stack, query the number of elements on the stack). The `std::vector` class template may be used to hold the elements on the stack.

3.8 (a) Develop a class `IntArray2` that represents a two-dimensional array of **`int`**s. The class should satisfy the following requirements:

i. A default constructor should be provided that creates an empty (i.e., 0×0) array.

ii. A constructor should be provided that takes two arguments corresponding to the width and height of the array to be created.

iii. The `std::vector` class template may be used to store the elements of the array.

iv. Objects of the class should be movable and copyable.

v. Member functions should be provided for querying the width, height, and size (i.e., number of elements) of the array.

vi. The function call operator should provide access to the (x, y)th element of the array.

vii. A stream inserter (i.e., `<<`) should be provided so that an array can be written to an output stream (`std::ostream`).

viii. A stream extractor (i.e., `>>`) should be provided so that an array can be read from an input stream (`std::istream`).

ix. All data members should be private.

(b) Write a program that thoroughly tests the `IntArray2` class.

3.9 (a) Develop a class `String` that represents a sequence of zero or more characters (i.e., a character string). Any character may appear in the string, including the null character (i.e., `'\0'`). The class should satisfy the following requirements:

i. A default constructor should be provided that creates an empty string.

ii. A constructor should be provided that takes a pointer to a null-terminated string as an argument to which to initialize the object to be created.

iii. The `std::vector` class template may be used to store the underlying string data.

iv. Objects of the class must be movable and copyable.

v. The operators `+=` and `+` should perform string concatenation.

vi. The subscripting operator (i.e., `[]`) should allow access to an indivdual character in the string.

vii. A stream inserter (i.e., `<<`) should be provided so that strings can be written to an output stream (`std::ostream`).

viii. A stream extractor (i.e., `>>`) should be provided so that strings can be read from an input stream (`std::istream`).

ix. The member function `c_str` should return a pointer to a null-terminated string corresponding the current value of the `String` object. More specifically, the null-terminated string should be equivalent to the contents of the `String` object with a null character appended.

x. The member function `size` should return the number of characters in the string.

(b) Write a program that thoroughly tests the `String` class.

(c) Also, test your `String` class with the program below.

```
#include <iostream>

// Replace the following line with your code.
#include "String.cpp"

int main()
{
    const String hello("Hello");
    const String world("World");

    String s = hello + String(" ");
    s += world;
    std::cout << s << "\n";
}
```

3.10 (a) Augment the functionality of the `String` class from Exercise 3.9 to provide iterator types that can be used for iteration over characters in a `String` object. The mutating and non-mutating iterator types should be called `Iterator` and `ConstIterator`, respectively. Provide `begin` and `end` member functions to obtain iterators.

(b) Write a program to thoroughly test the `String` class (including its `Iterator` and `ConstIterator` classes).

(c) Test your `String` class with the program below.

```
#include <iostream>

// Replace the following line with your code.
#include "String.cpp"

int main()
{
    const String hello = "Hello";
    const String world = "World";

    String s = hello;
    s += String(" ");
    s += world;
    std::cout << s << "\n";

    String::Iterator si = s.begin();
    String::ConstIterator sci = si;
    for (String::Iterator i = s.begin(); i != s.end(); ++i) {
        ++(*i);
    }
    for (String::ConstIterator i = s.begin(); i != s.end(); ++i) {
        std::cout << *i;
    }
```



```
24      std::cout << "\n";
25  }
```

Functor Classes and Lambda Expressions

3.11 (a) Develop a class `Quadratic` that represents a function of the form $f(x) = ax^2 + bx + c$, where x is a real variable and a, b, c are real constants. The class must satisfy the following requirements:

i. A constructor should be provided that takes the values for a, b, and c (from above) as arguments. All three of these arguments should default to zero.

ii. The class should provide move and copy constructors, move and copy assignment operators, and a destructor.

iii. The function call operator (i.e., **`operator()`**) should be provided. It should take a single argument x and return the value $f(x)$.

iv. All data members should be private.

(b) Write a program to thoroughly test your `Quadratic` class.

(c) Test your `Quadratic` class with the code below.

```
1  #include <iostream>
2
3  int main()
4  {
5      const Quadratic f(1.0, 2.0, 3.0);
6      const Quadratic g = f;
7      std::cout << f(1.0) << " " << g(1.0) << "\n";
8  }
```

3.12 State the output of the program whose source code is given below. Explain why this output is obtained.

```
1  #include <iostream>
2
3  int x = 1;
4
5  auto f1 = []() {return x + 1;};
6  auto f2 = [x = x]() {return x + 1;};
7
8  int main()
9  {
10     x = 10;
11     std::cout << f1() << " " << f2() << "\n";
12 }
```

3.13 State the output of the program whose source code is given below. Explain why this output is obtained.

```
1  #include <iostream>
2
3  auto make_functor(int x)
4  {
5      static int a = 0;
6      return [=](int y) {
7          static int b = 0;
8          return y + x + (a++) + (b++);
```



```
    };
}

int main()
{
    auto f1 = make_functor(1);
    auto f2 = make_functor(2);
    std::cout << f1(0) << "\n";
    std::cout << f2(0) << "\n";
    std::cout << f1(1) << "\n";
    std::cout << f2(1) << "\n";
}
```

3.14 State the output of the program whose source code is given below. Explain why this output is obtained.

```
#include <iostream>

auto f = [](auto x)
{
    static int c = 0;
    return (++c) + x;
};

int main()
{
    std::cout << f(1) << '\n';
    std::cout << f(0.5) << '\n';
    std::cout << f(1) << '\n';
    std::cout << f(0.5) << '\n';
}
```

Inheritance

3.15 (a) Consider the class `Base` shown in the listing below. Derive two classes `Derived1` and `Derived2` from `Base`, and in each case, define the member function `display` to output the name of the class.

```
class Base {
public:
    virtual void display()
    {
        std::cout << "Base\n";
    }
};
```

(b) Create an object of each of the types `Base`, `Derived1`, and `Derived2`, and invoke its `display` member function. Create pointers of type `Base*` that are initialized to point to the preceding `Base`, `Derived1`, and `Derived2` objects, and call `display` through each of these pointers.

3.16 Suppose that we want a class that can be used to represent people at a university (e.g., faculty, staff, undergraduate students, and graduate students). For all types of people, we want to record their name and university ID. For specific types of people, we want to record some additional information. In the case of faculty, we want to record their rank (i.e., Assistant Professor, Associate Professor, or Full Professor). In the case of undergraduate students, we want to record their year of study (i.e., 1, 2, 3, or 4). In the case of graduate students, we want to record their supervisor's university ID. Suggest a class hierarchy that might be used to represent the above collection of people. You do not need to fully implement the class.

Run-Time Type Information

3.17 (a) Use the **`typeid`** operator and the `name` member function of the `std::`**`type_info`** class in order to print the name of each of the types listed below.

i. **`int`**

ii. **`double`**

iii. **`void`** `(*)(`**`int`**`,` **`int`**`)`

iv. **`struct`** `Widget {};`

(b) Are the names that are printed what you expect? (It is worth noting that `name` member function typically returns a mangled (i.e., encoded) version of the type name.)

Chapter 4

Templates

4.1 Exercises

Function Templates

4.1 Consider a program that consists of the single source-code file whose listing is given below. The code shown in this listing below will fail to compile, due to a type-deduction failure.

(a) Identify which type cannot be deduced, and explain why this type cannot be deduced.

(b) Explain how the code can be modified to resolve this problem, without changing the definition of `func`.

```
template <class T, class U>
void func(const U& u)
{
    T x;
    // ...
}

int main()
{
    int x;
    func(x); // ERROR: type-deduction failure
}
```

4.2 Develop a template function `min3` that takes three arguments of the same type and returns the least of these arguments. For example, `min3(1, 0, 2)` would return `0` and `min3(1.5, 0.5, 3.0)` would return `0.5`.

4.3 (a) Develop a template function `sum` that computes the sum of zero or more elements (of the same type) that are stored contiguously in memory. The template should have a single parameter that is the type of the elements to be processed by the function. The function has two parameters: 1) a pointer to the first element in the range to be summed; and 2) a pointer to one-past-the-last element in the range to be summed.

(b) Write a program to test the `sum` function.

(c) Also, test the `sum` function with the code below.

```
#include <iostream>

int main()
{
```



```
5       const int i[3] = {1, 2, 3};
6       const double d[3] = {1.0, 2.0, 3.0};
7       std::cout << sum(&i[0], &i[3]) << "\n";
8       std::cout << sum(&d[0], &d[3]) << "\n";
9   }
```

Class Templates

4.4 Convert the `RealPoint2` class from Exercise 3.6 to a class template `Point2`. The template should take as a parameter the type to be used for point coordinates (which might be **`double`**, **`float`**, **`int`**, or some other numeric type). Test your code for at least one integral type (e.g., **`short`**, **`int`**, **`long`**) and one real type (e.g., **`float`**, **`double`**).

4.5 Develop a template class `Complex` that represents a complex number and is parameterized on the type `T` used to represent the real and imaginary parts of the complex number. So, for example, `Complex<`**`float`**`>` would be a complex number with the real and imaginary parts represented with **`float`**s.

4.6 (a) Convert the `IntArray2` class developed in Exercise 3.8 to a template class `Array2`. This template should have a single parameter, which is the type of the elements to be stored in the array.

(b) Write a program to thoroughly test your `Array2` class. Be sure to test it for both integral (e.g., **`int`**) and floating-point (e.g., **`double`**) types.

(c) Write a program that does the following:

i. Read a real array (including its dimensions) from standard input into an `Array2<`**`double`**`>`.
ii. Copy the elements from the `Array2<`**`double`**`>` object into an `Array2<`**`int`**`>` object which will result in the elements being rounded.
iii. Write the rounded array to standard output.

4.7 (a) Develop a template class `Array` that represents a fixed-size array. The template should have two parameters. The first specifies the type of the elements in the array and the second is an integer specifying the number of elements in the array. The class should satisfy the following requirements:

i. The array element data should be stored in a C++ array (which has fixed size). (In other words, do not use **`new`** and **`delete`** for memory allocation.)
ii. A constructor should be provided that has a single `std::initializer_list` parameter so that the array can be constructed from a list of element values. The size of the `initializer_list` must be less than or equal to the array size. In the case that it is less, the remaining elements (for which values are not provided) should be initialized to zero.
iii. Move and copy constructors, move and copy assignment operators, and a destructor should be provided.
iv. The subscripting operator should provide access to an array element.
v. The member function `size` should return the number of elements in the array. Since the value returned by this function is a compile-time constant, the function should be `constexpr`.
vi. The member function `fill` should fill the array with a specified value.
vii. All data members must be private.
viii. A stream inserter (i.e., `<<`) must be provided that writes an array to an output stream (i.e., `std::ostream`).
ix. The member type `Value` should be an alias for the type of the array elements.

(b) Write a program to thoroughly test the `Array` class.

(c) Also, test your code with the program given in the listing below.


```
#include <iostream>

// Include your code here.
#include "Array.hpp"

int main()
{
    const Array<int, 4> a{1, 2, 3, 4};
    const Array<int, 4> b{1, 2};
    auto c = a;
    auto d = b;
    d = std::move(c);
    std::cout << c << "\n";
}
```

4.8 (a) Develop a class template called `array_iterator` that is parameterized on a single type `T`. The type provides all of the functionality of a contiguous iterator that refers to an object of type `T`. If `T` is a const type, then `array_iterator<T>` is a nonmutating iterator. Otherwise, `array_iterator<T>` is a mutating iterator. The type `array_iterator<T>` is constructible from a pointer having type `T*`.

(b) Test the `array_iterator` class template type thoroughly.

(c) Ensure that the class works correctly with the code shown below.

```
#include <cassert>
#include <cstddef>
#include <iterator>
#include "array_iterator.hpp"

int main()
{
    using iterator = array_iterator<int>;
    using const_iterator = array_iterator<const int>;
    static_assert(std::random_access_iterator<iterator>);
    static_assert(std::random_access_iterator<const_iterator>);
    int a[]{1, 2, 3, 4};
    const std::size_t size = std::size(a);
    array_iterator<int> begin(std::begin(a));
    array_iterator<const int> cbegin(std::begin(a));
    array_iterator<int> end(std::end(a));
    array_iterator<const int> cend(std::end(a));
    assert(end - begin == size);
    assert(begin + size == end);
    assert(end - size == begin);
    assert(size + begin == end);
    auto i = begin;
    assert((i += size) == end);
    i = end;
    assert((i -= size) == begin);
    assert(&*begin == &a[0]);
    i = begin;
    auto ci = cbegin;
    assert(ci == i);
    assert(i == ci);
    ++ci;
    assert(i != ci);
    assert(ci != i);
}
```


Variadic Templates

4.9 Develop a (variadic) template function `minimum` that takes an one or more arguments of the same type and returns the argument with the least value. For example:

- `minimum(6, 5, 4, 3, 2)` would return `2`; and
- `minimum(1.0, 2.0, 3.0, 4.0, 5.0, 6.0)` would return `1.0`.

[Hint: Use recursion. Two function templates are needed, with one being variadic.]

4.10 Develop a (variadic) template function `sum` that takes two or more arguments of the same type and returns their sum. For example:

- `sum(1, 2, 4)` would return `7`; and
- `sum(2.0, 1.0, 1.0, 0.5)` would return `4.5`.

Template Specialization

4.11 Develop a class template that can be used to compute the factorial of a `std::size_t` value at compile time. The class template should have a declaration like the following:

```
template<std::size_t N> struct factorial;
```

The class template should provide a single (public) member `value` that holds the factorial of `N`. Template specialization should be used to handle the case of `N` equal to `0`. [Hint: Use an enumeration.]

4.12 (a) Develop a class template `IsVoid` that can be used to test if a type matches the type **`void`**, ignoring any **`const`** or **`volatile`** qualifiers. The template has a single parameter `T`, which is a type that is to be tested for being **`void`**. The class should have only the following two members, both of which are public:

i. A type member `ValueType` that is an alias for **`bool`**.

ii. A constant data member `value` of type `ValueType` that indicates if `T` matches with **`bool`**.

For example, `IsVoid<void>::value` should be **`true`**, while `IsVoid<int>::value` should be **`false`**. [Hint: Use template specialization.]

(b) Test your `IsVoid` template class with the program whose listing is given below.

```
#include <iostream>

// Include your IsVoid class template here.
#include "IsVoid.hpp"

int main()
{
    std::cout << std::boolalpha
      << IsVoid<void>::value << "\n"
      << IsVoid<const void>::value << "\n"
      << IsVoid<volatile void>::value << "\n"
      << IsVoid<const volatile void>::value << "\n"
      << IsVoid<int>::value << "\n"
      << IsVoid<const double>::value << "\n"
      << IsVoid<volatile char>::value << "\n"
      << IsVoid<const volatile long long>::value << "\n"
      << IsVoid<void (*)(int, int)>::value << "\n"
      ;
}
```

This program would be expected to produce the following output when executed:


```
true
true
true
true
false
false
false
false
false
```

Generic Lambdas

4.13 (a) Use a lambda expression to define a functor called `print_container`. The function-call operator should take two parameters, having the types `std::ostream&` and `T`, where `T` is intended to be a sequence container type such as `std::vector`, `std::list`, or `std::deque`. This operator should output each of the elements in the container on the same line separated by space characters to the output stream specified by the first parameter.

(b) Use a lambda expression to define a functor called `prefix_increment`. This functor should have a function-call operator that takes a single parameter of any type. The behavior of this operator should be such that `prefix_increment(x)` is semantically equivalent to `++x`.

(c) In the code below, replace the indicated preprocessor directive (i.e., **`#include`** `"generic_lambda_2_b.cpp"`) with the functor definitions written in the preceding parts of this exercise and confirm that the resulting code works correctly.

```
#include <iostream>
#include <vector>
#include <list>
#include <deque>
#include <algorithm>
#include <iterator>

template <class T>
void do_work()
{
// Replace the following preprocessor include directive with
// the definitions of print_container and prefix_increment functors.
#include "generic_lambda_2_b.cpp"

    T v{0, 1, 2, 3, 4, 5, 6, 7};
    // Print the elements in the container.
    print_container(std::cout, v);
    // Increment each of the elements in the container.
    std::for_each(v.begin(), v.end(), prefix_increment);
    // Print the elements in the container.
    print_container(std::cout, v);
}

int main()
{
    do_work<std::vector<int>>();
    do_work<std::deque<int>>();
    do_work<std::list<int>>();
}
```


4.14 In the code below, replace the indicated preprocessor directive (i.e., `#include "generic_lambda_1_b.cpp"`) with the definition of a functor called `multiply` that is obtained using a lambda expression. This functor should have a function-call operator that takes two parameters `x` and `y` of any type and returns the value `x * y` (having the correct type). Confirm that the resulting code works correctly.

```
#include <iostream>
#include <complex>

int main()
{

// Replace the following preprocessor include directive with
// the defintion of the multiply functor.
#include "generic_lambda_1_b.cpp"

    using namespace std::literals;
    std::cout
      << multiply(2, 3) << " "
      << multiply(0.5, 2.0) << " "
      << multiply(2, 0.25) << " "
      << multiply(1.0 + 1.0i, 1.0 - 1.0i) << "\n";

    return 0;
}
```

Miscellany

4.15 Write a template raw literal operator that uses the suffix `_4` to denote an integer literal in radix 4. Test your code with the program in the listing below.

```
#include <iostream>

// Include your code here.
#include "raw_user_defined_literal_1.hpp"

int main()
{
    // Output 1 4 16
    std::cout << 1_4 << " " << 10_4 << " " << 100_4 << "\n";
    // Output -1 -4 -16
    std::cout << -1_4 << " " << -10_4 << " " << -100_4 << "\n";
    // Output 228 27
    std::cout << 3210_4 << " " << 0123_4 << "\n";
    // Output -228 -27
    std::cout << -3210_4 << " " << -0123_4 << "\n";
}
```


Chapter 5

Library

5.1 Exercises

Containers, Iterators Algorithms

5.1 (a) Develop a template function `merge` that merges two containers. The template function should have three parameters. The first two parameters specify the containers to be merged. The last parameter specifies the container to hold the result of the merge operation. The types of all three containers may be different. The template should work for at least the following container types: `std::vector`, `std::list`, and `std::set`.

(b) Develop a template function `output` that writes the contents of any container to a particular output stream (i.e., `std::ostream`). The template function should have two parameters. The first parameter specifies the output stream to which the container elements should be written. The second parameter specifies the container whose elements are to be output. You may assume that the container element type has a stream inserter. In other words, the `<<` operator may be used to write a container element to an output stream. The `output` function should return a reference to the output stream.

(c) Test your `merge` and `output` functions with the program whose listing is given below.

```
#include <iostream>
#include <vector>
#include <list>
#include <set>
#include <string>

// Include your code here.
#include "merge.hpp"

int main()
{
    const std::vector<std::string> c1{"delta", "beta", "alpha"};
    const std::list<std::string> c2{"delta", "beta", "alpha"};
    const std::set<std::string> c3{"one", "two", "four"};
    std::vector<std::string> d1;
    std::list<std::string> d2;
    std::set<std::string> d3;
    merge(c1, c2, d1);
    output(std::cout, d1) << "\n";
    merge(c2, c3, d2);
    output(std::cout, d2) << "\n";
    merge(c3, c1, d3);
    output(std::cout, d3) << "\n";
```



```
24  }
```

5.2 Use a functor (i.e., function object) and the `std::sort` algorithm in order to write a program that does the following:

(a) Read real numbers from standard input (separated by whitespace) and store them into a `std::vector<double>`. The program should keep reading numbers until end-of-file is reached.

(b) Output the numbers to standard output from the `std::vector<double>` in the same order that they were input.

(c) Sort the elements in the `std::vector<double>` in order of decreasing magnitude.

(d) Output the sorted list to standard output.

5.3 Suppose that we want to compute the sum of all of the elements in a container of type `std::vector<int>`. This can be accomplished by each of the functions `calcSum1` and `calcSum2` in the listing below.

```
1  #include <vector>
2
3  int calcSum1(const std::vector<int>& v)
4  {
5     int sum = 0;
6     for (auto i = v.begin(); i != v.end(); ++i) {
7        sum += *i;
8     }
9     return sum;
10 }
11
12 int calcSum2(const std::vector<int>& v)
13 {
14    int sum = 0;
15    for (int i = 0; i < v.size(); ++i) {
16       sum += v[i];
17    }
18    return sum;
19 }
```

(a) Of the two functions `calcSum1` and `calcSum2`, which would you expect to require less time to execute? Explain the reason for your answer.

(b) Write a program that uses each of the `calcSum1` and `calcSum2` functions to compute the sum of a `std::vector<int>` containing $2^{24} = 16777216$ elements. Use `std::chrono::high_resolution_timer` to measure the execution time required for each of these functions. Enable full optimization when compiling your program. After having computed the sum in each case, be sure to use the result (e.g., by printing its value), since a good optimizer is likely to eliminate the sum calculation if the result of the calculation is never used. Of the functions `calcSum1` and `calcSum2`, which requires less execution time? By what factor do the execution times of these two functions differ? Is this result consistent with your answer in the previous part of this exercise? (It should be.)

5.4 Consider the two functions `getVector1` and `getVector2` in the listing below. Each of these functions returns a `std::vector<int>` of the requested size n with the elements initialized to $0, 1, 2, \ldots, (n-1)$.


```
#include <vector>

std::vector<int> getVector1(int n)
{
    std::vector<int> result;
    for (int i = 0; i < n; ++i) {
        result.push_back(i);
    }
    return result;
}

std::vector<int> getVector2(int n)
{
    std::vector<int> result;
    result.reserve(n);
    for (int i = 0; i < n; ++i) {
        result.push_back(i);
    }
    return result;
}
```

(a) Of the two functions `getVector1` and `getVector2`, which would you expect to be require less time to execute? Explain the reason for your answer.

(b) Write a program that uses each of the `getVector1` and `getVector2` functions to generate a `std::vector<int>` containing $2^{24} = 16777216$ elements. Use `std::chrono::high_resolution_timer` to measure the execution time required for each of these functions. Enable full optimization when compiling your program. Of the functions `getVector1` and `getVector2`, which requires less execution time? By what factor do the execution times of these two functions differ? Is this result consistent with your answer in the previous part of this exercise? (It should be.)

5.5 For each of the cases below, write a program to perform the requested task.

(a) Print 100 random numbers. Each random number must be obtained by applying the function-call operator to a closure. Only one closure may be constructed. The random number should be generated by `drand48` which has the declaration (found in `cstdlib`): `double drand48();`

(b) Use `std::sort` to sort (in ascending order) a `std::vector` of `int`s containing the elements $\{1,3,8,6,4,5,7,2,0,9\}$. After the sorting is completed, print the number of comparisons that were needed during sorting. Your solution should make use of a lambda expression.

(c) Given the vector and set:

```
std::vector<int> vec{1, 3, 8, 6, 4, 5, 7, 2, 0, 9};
std::set<int> infinities{3, 2};
```

Use `std::stable_sort` along with a lambda expression to sort the elements of `vec` in ascending order with elements matching those in `infinities` being treated as if they were (positive) infinity (i.e., coming last in sort order). Your solution should utilize a lambda expression.

5.6 A program needs to store elements of type `T` in a container. For each set of assumptions below, identify which container in the standard library would be the most appropriate to employ and fully justify your choice.

(a) First case.

i. elements in the container will need to be located very frequently (based on their value);

ii. `T` is a builtin type;

iii. memory usage should be minimized;
iv. the container will hold a very large number of elements;
v. the contents of the container will not be changed after initialization.

(b) Second case.

i. elements in the container will need to be located very frequently (based on their value);
ii. the contents of the container will be changed very frequently, with elements being added and removed at arbitrary positions in the container;
iii. the container will hold a relatively large number of elements;

(c) Third case.

i. elements will frequently be added to and removed from arbitrary positions in the container;
ii. only the first and last elements need to located frequently;

5.7 Given an object `v` of type `std::vector<int>`, write code to determine whether all of the elements of the vector are odd integers. Use a lambda expression and `std::all_of`.

5.8 Given an object `v` of type `std::vector<std::pair<double,double>`, write code to sort the elements of the vector by the product of their `first` and `second` members. Use a lambda expression and `std::sort`.

5.9 Write a function called `count_even` that takes a single parameter of type `const std::vector<int>&` and returns a value of type `int` corresponding to a count of the number of even elements in the vector. Use `std::for_each` with a lambda expression to count the number of even elements.

5.10 Write a function called `find_first_negative` that takes a single parameter of type `const std::vector<int>&` and returns a value of type `int` corresponding to the index of the first negative element in the vector. If the vector contains no negative elements, the value `-1` should be returned. Use `std::find_if` with a lambda expression to find the first negative element.

5.11 Write a function called `transform_quadratic` that matches the following declaration:

```
void transform_quadratic(std::vector<int>& v, double a, double b, double c);
```

The function should replace each element `x` in the vector `v` with the value `a * x * x + b * x + c`. Use `std::transform` with a lambda expression in order to perform this task.

5.12 Write a function called `is_sorted` that matches the following declaration:

```
bool is_sorted(const std::vector<double>& v, int& num_compares)
```

The function should return `true` if the elements of the vector `v` are sorted in ascending order and `false` otherwise. Use `std::is_sorted` with a lambda expression in order to determine if the elements are sorted. Upon return, the parameter `num_compares` should be set to the number of comparisons required by `std::is_sorted` (i.e., the number of times that the function-call operator is invoked for the comparison functor).

Input/Output

5.13 Consider the function `copyStream` given in the listing below. This function is intended to copy the contents of one stream to another so that the output stream will be associated with exactly the same sequence of characters as the input stream. The function returns **`true`** upon success and **`false`** otherwise. Unfortunately, this code has a serious flaw. Identify this flaw and explain how it can be fixed.

```
#include <iostream>

bool copyStream(std::istream& in, std::ostream& out)
{
    char c;
    while (in >> c) {
        if (!(out << c)) {
            // Output failed.
            return false;
        }
    }
    if (!in.eof()) {
        // Input failed.
        return false;
    }
    // Success.
    return true;
}
```

5.14 Develop a template function `output` that writes data to a `std::ostream`. The parameters of the function are as follows: 1) an output stream `std::ostream`; and 2) one or more parameters that correspond to objects that can be inserted into a stream with a stream inserter (i.e., `<<`). For example, the `output` function should work with the program listed below.

```
#include <iostream>
#include <iomanip>

// Include your code here.
#include "variadic_template_output_1.hpp"

int main()
{
    // Output "Hello, World!\n".
    output(std::cout, "Hello, World!\n");

    // Output "1\n".
    output(std::cout, 1, "\n");

    // Output "Hello World!\n".
    output(std::cout, "Hello ", "World!", "\n");

    // Output "Testing***1***2***3\n"
    output(std::cout,
      std::setfill('*'),
      "Testing",
      std::setw(4), 1,
      std::setw(4), 2,
      std::setw(4), 3,
      "\n");
}
```


Smart Pointers

5.15 What is the difference between the smart pointer types `std::unique_ptr` and `std::shared_ptr`? When should one of these types be preferred over the other?

5.16 Recall the `str_concatenate` function developed in Exercise 2.25. The interface provided by this function is quite error prone, as it requires that the caller remember to free memory at some potentially much later point in the code. Modify the interface of this function in such a way as to eliminate this problem, using an approach based on smart pointers. [Hint: Use one of the smart pointer types from the standard library.]

5.17 For each part of this exercise, consider the program whose listing is given. The given code has a number of problems that relate to memory management (e.g., memory ownership). Fix these problems by introducing smart pointers into the code. Only change interfaces to the extent that it is necessary in order to fix the problems in the original code.

(a)

```
#include <iostream>

std::size_t string_length(const char* s)
{
    std::size_t n = 0;
    while (*s != '\0') {
        ++s;
        ++n;
    }
    return n;
}

char* string_copy(char* d, const char* s)
{
    char* p = d;
    while ((*p = *s) != '\0') {
        ++p;
        ++s;
    }
    *p = '\0';
    return d;
}

char *string_duplicate(const char* s)
{
    std::size_t n = string_length(s);
    char *result = new char[n + 1];
    string_copy(result, s);
    return result;
}

char* string_concatenate(const char* first, const char* second)
{
    std::size_t first_len = string_length(first);
    std::size_t second_len = string_length(second);
    char* result = new char[first_len + second_len + 1];
    string_copy(result, first);
    string_copy(result + first_len, second);
    return result;
```



```
}

int main() {
    char *s = string_concatenate("Hello, ", "World!");
    char *t = string_duplicate(s);
    std::cout << t << '\n';
    delete[] s;
    delete[] t;
}
```

(b)

```
#include <iostream>
#include <cstring>

// Duplicate a C-style string.
char *string_duplicate(const char* s)
{
    std::size_t n = std::strlen(s);
    char *result = new char[n + 1];
    std::strcpy(result, s);
    return result;
}

// Read character data from an input stream and return it.
char* source_data()
{
    char buffer[1024];
    if (std::cin.getline(buffer, sizeof(buffer))) {
        return string_duplicate(buffer);
    } else {
        return nullptr;
    }
}

// Transform the character data and return the transformed data.
// (Map lowercase to uppercase.)
char* transform_data(char* s)
{
    char* p = s;
    while (*p != '\0') {
        if (islower(*p)) {
            *p = toupper(*p);
        }
        ++p;
    }
    return s;
}

// Write the character data to an output stream and then discard the data.
void sink_data(char *s)
{
    std::cout << s << '\n';
    delete[] s;
}

int main()
{
```



```
    char *s;
    while (s = source_data()) {
        s = transform_data(s);
        sink_data(s);
    }
}
```

(c)

```
#include <iostream>
#include <cassert>

class Counter
{
public:
    Counter(int i = 0) : p_(new int(i)) {}
    ~Counter() {
        if (p_) {
            delete p_;
        }
    }
    Counter(Counter&& c) : p_(c.p_) {
        c.p_ = nullptr;
    }
    Counter(const Counter& c) {
        if (p_) {
            p_ = new int(*c.p_);
        } else {
            p_ = nullptr;
        }
    }
    Counter& operator=(Counter&& c) {
        if (this != &c) {
            if (p_) {
                delete p_;
            }
            p_ = c.p_;
            c.p_ = nullptr;
        }
        return *this;
    }
    Counter& operator=(const Counter& c) {
        if (this != &c) {
            if (p_) {
                delete p_;
            }
            p_ = new int(*c.p_);
        }
        return *this;
    }
    int getCount() const {
        assert(p_);
        return *p_;
    }
    Counter& operator++() {
        assert(p_);
        ++*p_;
        return *this;
```



```
50      }
51  private:
52      int* p_; // pointer to counter value
53  };
54
55  int main() {
56      Counter a(0);
57      Counter b(std::move(a));
58      Counter c(b);;
59      c = std::move(b);
60      b = ++c;
61      a = ++c;
62      std::cout << a.getCount() << '\n';
63      std::cout << b.getCount() << '\n';
64      std::cout << c.getCount() << '\n';
65  }
```

Miscellany

5.18 Write a program that reads real numbers from standard input into a `std::vector` (until end-of-file is reached), and then prints (to standard output) the median of the numbers read. Use the standard library as much as possible. Do not fully sort the vector, as this would be inefficient.

5.19 Develop a program that writes to standard output a table for converting from pounds to kilograms. The table should have two columns, the first for pounds and the second for kilograms. The entries in the first column (i.e., pounds) should range from 0 to 200 in increments of 10. The program output should be formatted so that it exactly matches that shown below. Use I/O manipulators to assist with the formatting. To convert from pounds to kilograms, use the formula $k = 0.453592p$, where p and k are the mass in pounds and kilograms, respectively.

```
Pounds      Kilograms
  0.00           0.00
 10.00           4.54
 20.00           9.07
 30.00          13.61
   ...            ...
200.00          90.72
```

5.20 Write a program that generates one million random real numbers chosen from a normal distribution with mean μ and standard deviation σ. The quantity μ should be chosen as a random number taken from a uniform distribution on the interval $[-10, 10]$, and σ should be chosen as 1. Use the random-number-generation engine `std::mt19937` (i.e., 32-bit Mersenne Twister) seeded with a random number from `std::random_device`. Output a (text-based) histogram with 20 bins to show the approximate distribution. The histogram output should follow the formatting example shown below. The output has one line per histogram bin, consisting of two fields: 1) a real number specifying the center of the histogram bin; and 2) zero or more asterisks, with number of asterisks chosen in proportion to the histogram bin count. Avoid having a single line of output exceed 80 characters so that the histogram will display without line wrapping on an 80-column terminal.

```
-11.07
-10.60
-10.13
 -9.66
 -9.19 *
 -8.73 ****
```



```
-8.26 **********
-7.79 ******************
-7.32 ****************************
-6.85 **********************************
-6.38 ********************************
-5.91 ***************************
-5.44 ******************
-4.97 *********
-4.50 ***
-4.04 *
-3.57
-3.10
-2.63
-2.16
```


Chapter 6

Exceptions

6.1 Exercises

Exceptions

6.1 Explain what is fundamentally wrong with structuring the code for a function as shown below.

```
void func() {
   initialize(); // perform initialization
   do_work(); // do some work
   cleanup(); // perform any necessary cleanup
}
```

6.2 Consider the code given below, in which `Thing` is some class type. Can the function `analyze` throw an exception? Justify your answer. Can the function `doWork` throw an exception? Justify your answer.

```
// The type Thing is defined in the following header file.
#include <Thing.hpp>

Thing globalThing;

void analyze(Thing x) noexcept
{
   // ...
}

void doWork()
{
   analyze(globalThing);
}
```

6.3 (a) Write a function called `safe_divide` that takes two **`double`** arguments and returns a **`double`** corresponding to the result of dividing the first argument by the second argument. Before performing the division, the function should check to determine if division by zero would occur. If division by zero would occur, instead of performing the division, the function should throw an exception of type `std::invalid_argument`.

(b) Write a program that loops performing the following steps: 1) read two real numbers from standard input; 2) compute the quotients of these two numbers using the `safe_divide` function; 3) write the division result to standard output. If the `safe_divide` function throws an `invalid_argument` exception, this should be caught by the caller and an appropriate error message written to standard error. Check to ensure that the program correctly handles both the cases of division by zero and nonzero values.

6.4 Develop a function template called `quadratic` that can be used to evaluate a quadratic function of the form $f(x) = ax^2 + bx + c$ (where x, a, b, and c are real) and has the following properties:

- The template is parameterized on a real number type `T`, which may be assumed to have all of the usual arithmetic operators (such as addition and multiplication).
- The function has four parameters of type `T` corresponding to the values of x, a, b, and c, respectively.
- The function returns a value of type `T` corresponding to $f(x)$.
- The `noexcept` specifier for the function must be correct regardless of the type `T`.

6.5 (a) Write a function `process` having a single **`int`** parameter `x` that performs the following:

- if `x` is 1, an exception of type **`int`** is thrown;
- if `x` is 2, an exception `"yikes"` is thrown;
- otherwise, nothing is done.

(b) Write a program whose `main` function performs the following for each element `x` of `std::vector<int>{0, 1, 2, 3}`: Call `process(x)` and then print `"okay\n"` to standard output. While doing this, exceptions of type **`int`** and **`char *`** should be caught. In each case, the exception handler should print the type of exception (e.g., **`int`**) and the value of the exception (and allow execution of the program to continue normally).

Stack Unwinding

6.6 When each of the programs listed below is executed, the **`throw`** statement marked by a comment of the form `/* throw site */` will be reached, resulting in an exception being thrown. Identify the objects that are destroyed during the stack unwinding process initiated by this **`throw`** statement and specify the sequence in which these objects are destroyed.

(a)

```
#include <iostream>
#include <stdexcept>
#include <vector>
#include <complex>
#include <string>

using namespace std::literals;

void func1() {
    std::string hello("hello"s);
    std::vector<int> countdown{3, 2, 1, 0};
    {
        std::string die("die"s);
        std::vector<std::string> die3{die, die, die};
        throw std::runtime_error("yikes!"); /* throw site */
    }
    std::string goodbye("goodbye"s);
}

void func2(bool flag) {
    std::vector<double> dv{1.0, 0.5, 0.25};
    std::string herb("Herb"s);
    if (flag) {
        std::string bjarne("Bjarne"s);
        std::complex<double> i(1.0i);
        func1();
```



```
    }
}

int main() {
    std::string scott("Scott"s);
    std::vector<int> count{1, 2, 3};
    try {
        std::complex<double> z(1.0i);
        std::complex<double> u(z * z);
        func2(true);
    }
    catch (std::runtime_error& e) {
        std::cout << e.what() << "\n";
    }
}
```

(b)

```
#include <iostream>
#include <stdexcept>
#include <string>
#include <cmath>
#include <vector>

using namespace std::literals;

[[noreturn]] void panic(std::string s)
{
    throw std::runtime_error(s); /* throw site */
}

double squareRoot(double x)
{
    if (x < 0.0) {
        std::string s("square root of negative number"s);
        panic(s);
    }
    return std::sqrt(x);
}

int main()
{
    std::vector<double> v{1.0, 4.0, -1.0};
    for (auto x : v) {
        try {
            std::cout << squareRoot(x) << "\n";
        }
        catch (std::runtime_error& e) {
            std::cout << "exception: " << e.what() << "\n";
        }
    }
}
```

6.7 Identify (in order) each point in the execution of the code given below where the destructor for `std::string` is called. For each invocation of this destructor, identify the object being destroyed and its value (at destruction time). Assume that the program does not run out of memory (e.g., no `std::bad_alloc` exceptions are thrown).


```
#include <iostream>
#include <string>

using std::string;

class omg {};
class yikes {};

void func_1(string s)
{
    string alpha("alpha");
    if (s == "omg") {
        string beta("beta");
        throw omg();
        string gamma("gamma");
    } else if (s == "yikes") {
        string delta("delta");
        throw yikes();
        string epsilon("epsilon");
    }
    string zeta("zeta");
}

void func_2(string s)
{
    string eta("eta");
    try {
        string theta("theta");
        func_1(s);
        string iota("iota");
    } catch (const omg& e) {
        string kappa("kappa");
        func_1("yikes");
        string lambda("lambda");
    }
    string mu("mu");
}

int main()
try {
    string nu("nu");
    func_2("omg");
    string xi("xi");
} catch (...) {
    std::cerr << "exception\n";
}
```

Exception Safety

6.8 For each of the code listings given below, identify any exception-safety problems and suggest how they can be eliminated.

(a)

```
#include <cstddef>

```



```
3  void useBuffers(std::size_t, char* p1, char* p2) noexcept;
4
5  void func(std::size_t size)
6  {
7     // allocate memory for first buffer
8     char* buf1 = new char[size];
9     // allocate memory for second buffer
10    char* buf2 = new char[size];
11    // use the buffers
12    useBuffers(size, buf1, buf2);
13    // deallocate memory for first buffer
14    delete[] buf1;
15    // deallocate memory for second buffer
16    delete[] buf2;
17 }
```

(b)

```
1  #include <iostream>
2  #include <ios>
3  #include <iomanip>
4
5  // print int to stream in hexadecimal
6  bool print(std::ostream& out, int x)
7  {
8     // save formatting flags
9     auto oldFlags = out.flags();
10    // set formatting flags to use hexadecimal and output integer value
11    out << std::showbase << std::hex << x << "\n";
12    // restore formatting flags
13    out.flags(oldFlags);
14    return out;
15 }
```

(c)

```
1  #include <deque>
2
3  // Note:
4  // std::deque::front: guaranteed not to throw
5  // std::deque::pop_front: guaranteed not to throw
6  // std::deque::push_back: may throw, provides strong guarantee
7
8  // FIFO queue class
9  template <class T>
10 class Queue
11 {
12 public:
13    // get number of elements on queue
14    int size() const {
15       return q_.size();
16    }
17    // remove and return element from front of queue
18    T get() {
19       T result(q_.front());
20       q_.pop_front();
21       return result;
22    }
```



```
23      // add element to back of queue
24      void put(const T& value) {
25          q_.push_back(value);
26      }
27  private:
28      // underlying queue
29      std::deque<T> q_;
30  };
```

6.9 Write a template function `strongSort` that sorts a `std::vector` (using `std::sort`) and provides the strong exception-safety guarantee. The function should have a declaration matching the following:

```
template <class T, class F> void strongSort(std::vector<T>&, F less);
```

The first and second function parameters correspond to the `vector` to be sorted and a comparison function/functor.

6.10 Write a function `getBuffer` that has a single **`int`** parameter `size` that specifies the size (in characters) of a memory buffer to allocate. The function allocates the buffer and returns it. Use an appropriate smart pointer type in order to ensure that the code is exception safe. Assume that the buffer is to be owned by only one part of the program.

6.11 (a) Write a function `getWord` that reads a (whitespace-delimited) word of input as a `std::string` from a `std::istream` and has the following characteristics:

- The function has a single `std:istream&` parameter specifying the stream from which to read the `std::string`.
- The function should return an appropriate pointer type that refers to the `std::string` read. A null pointer should be returned if the input operation fails. (The returned `std::string` must be allocated on the heap.)
- The function must be exception safe.

(b) Write a program whose `main` function loops performing the following: 1) read a word (i.e., `std::string`) using the `getWord` function; 2) store the pointer to the `std::string` in `std::vector` and `std::set` containers. The loop terminates when the `getWord` function returns a null pointer. The string data should be stored in the two containers in such a way that the data is not duplicated between the containers. All of the code must be exception safe.

Chapter 7

Concurrency

7.1 Exercises

Sequential Consistency

7.1 Consider a multithreaded program written in a language with a syntax similar to C++ that provides (full) sequential consistency (not SC-DRF) and an assertion mechanism (i.e., `assert`) similar to the one in C++. The program has two threads, and these threads share the variables `x`, `y`, and `z` of type **`int`**, all of which are initially zero. Below, several different scenarios are given for the code of the two threads. In each case, the code contains a number of assertions. For each scenario, indicate whether each assertion will be true: always, sometimes, or never. Justify your answer in each case.

(a) First scenario.

- Thread 1 Code.

```
[A1]  x = 1;
[A2]  y = 1;
```

- Thread 2 Code.

```
[B1]  if (x == 1)
[B2]      assert(y == 1);
```

(b) Second scenario.

- Thread 1 Code

```
[A1]  x = 1;
[A2]  y = 1;
```

- Thread 2 Code

```
[B1]  while (!y) {}
[B2]  assert(x == 1);
```

(c) Third scenario.

- Thread 1 Code

```
[A1]  x = 1;
[A2]  y = 1;
[A3]  z = 1;
```

- Thread 2 Code

```
[B1]  while (!y) {}
[B2]  assert(x == 1);
[B3]  assert(z == 1);
```

(d) Fourth scenario.

- Thread 1 Code.

```
[A1]  x = 1;
[A2]  y = 1;
```

- Thread 2 Code.

```
[B1]  if (y == 1)
[B2]      assert(x == 1);
```

7.2 Consider a multithreaded program written in a language with a syntax similar to C++ that provides (full) sequential consistency (not SC-DRF). The program has three threads, and these threads share the variables `a` and `b` of type **`int`**, both of which are initially zero. Identify *n* possible sets of values for `a` and `b` that can be obtained upon completion of program execution, where *n* is specified in each part of this question. For each set, specify at least one sequentially-consistent execution of the program that produces that set. When specifying an execution sequence, the line labels provided in the source listing (i.e., A1, A2, B1, B2, C1, and C2) can be used to identify the statements being executed.

(a) First scenario. Find at least 2 different sets of values for `a` and `b`.

Thread 1 Code

```
[A1]  a = 1;
[A2]  b = 1;
```

Thread 2 Code

```
[B1]  while (b == 0) {}
[B2]  a = a + 1;
```

Thread 3 Code

```
[C1]  while (a == 0) {}
[C2]  b = b + 1;
```

(b) Second scenario. Find at least 5 different sets of values for `a` and `b`.

Thread 1 Code

```
[A1]  a = a + 1;
[A2]  b = b + 1;
```

Thread 2 Code

```
[B1]  if (a == 0)
[B2]    {a = 2 * b;}
```

Thread 3 Code

```
[C1]  if (b == 0)
[C2]    {b = 2 * a;}
```

7.3 Consider the execution of the two-threaded program listed below. The program has four integer variables `a`, `b`, `x`, and `y`, all of which are initially zero. Enumerate all possible sequentially-consistent executions of this program. For each case, state the value of `a` and `b` upon completion of the program. Upon program completion, is there any combination of the values 0 and 1 that cannot be obtained for `a` and `b`?

- Thread A Code:

```
[A1]  x = 1;
[A2]  a = y;
```

- Thread B Code:

```
[B1]  y = 1;
[B2]  b = x;
```

Data Races

7.4 For each of the programs listed below: 1) state the behavior of the program (i.e., what it does) when executed (being sure to include all possibilities); and 2) if the program contains any data races, identify them and suggest how they might be fixed.

(a)

```
1  #include <iostream>
2  #include <thread>
3
4  int x = 0;
5  int y = 0;
```



```
6
7  int main()
8  {
9      std::thread t1([]() {
10         x = 1;
11         y = 2;
12     });
13     std::thread t2([]() {
14         std::cout << y << " ";
15         std::cout << x << std::endl;
16     });
17     t1.join();
18     t2.join();
19 }
```

(b)

```
1  #include <iostream>
2  #include <thread>
3  #include <mutex>
4
5  std::mutex m;
6  int x = 0;
7  int y = 0;
8
9  int main()
10 {
11     std::thread t1([]() {
12         std::scoped_lock<std::mutex> lock(m);
13         x = 1;
14         y = 2;
15     });
16     std::thread t2([]() {
17         std::scoped_lock<std::mutex> lock(m);
18         std::cout << y << " ";
19         std::cout << x << std::endl;
20     });
21     t1.join();
22     t2.join();
23 }
```

(c)

```
1  #include <iostream>
2  #include <thread>
3  #include <atomic>
4
5  std::atomic<int> x(0);
6  std::atomic<int> y(0);
7
8  int main()
9  {
10     std::thread t1([]() {
11         x.store(1);
12         y.store(2);
13     });
14     std::thread t2([]() {
15         std::cout << y.load() << " ";
```



```
        std::cout << x.load() << std::endl;
    });
    t1.join();
    t2.join();
}
```

(d)

```
#include <iostream>
#include <thread>
#include <atomic>

std::atomic<int> x(0);
std::atomic<int> y(0);

int main()
{
    std::thread t1([]() {
        x.store(1, std::memory_order_relaxed);
        y.store(2, std::memory_order_relaxed);
    });
    std::thread t2([]() {
        std::cout << y.load(std::memory_order_relaxed) << " ";
        std::cout << x.load(std::memory_order_relaxed) << std::endl;
    });
    t1.join();
    t2.join();
}
```

(e)

```
#include <iostream>
#include <string>
#include <atomic>
#include <thread>

std::atomic<std::string*> s(nullptr);
int i = 0;

void producer()
{
    std::string* p = new std::string("Hello");
    i = 1;
    s.store(p, std::memory_order_release);
}

void consumer()
{
    std::string *p;
    while (!(p = s.load(std::memory_order_acquire))) {
        ;
    }
    std::cout << *p << " ";
    std::cout << i << "\n";
}

int main()
{
```



```
    std::thread t1(producer);
    std::thread t2(consumer);
    t1.join();
    t2.join();
}
```

(f)

```
#include <iostream>
#include <string>
#include <atomic>
#include <thread>

std::atomic<std::string*> s(nullptr);
int i = 0;

void producer()
{
    std::string* p = new std::string("Hello");
    i = 1;
    s.store(p, std::memory_order_release);
}

void consumer()
{
    std::string *p;
    while (!(p = s.load(std::memory_order_consume))) {
        ;
    }
    std::cout << *p << " ";
    std::cout << i << "\n";
}

int main()
{
    std::thread t1(producer);
    std::thread t2(consumer);
    t1.join();
    t2.join();
}
```

(g)

```
#include <thread>
#include <cassert>

int x = 0;
bool done = false;

int main()
{
    std::thread t1([](){
        x = 42;
        done = true;
    });
    std::thread t2([](){
        while (!done) {}
        assert(x == 42);
```



```
    });
    t1.join();
    t2.join();
}
```

(h)

```
#include <thread>
#include <mutex>
#include <cassert>

std::mutex m;
bool initialized = false;
int x;

void do_work()
{
    if (!initialized) {
        std::scoped_lock<std::mutex> g(m);
        x = 42;
        initialized = true;
    }
    assert(x == 42);
}

int main()
{
    std::thread t1(do_work);
    std::thread t2(do_work);
    t1.join();
    t2.join();
}
```

(i)

```
#include <thread>

int x = 0;
int y = 0;

int main()
{
    std::thread t1([](){
        if (x) {
            y = 1;
        }
    });
    std::thread t2([](){
        if (y) {
            x = 1;
        }
    });
    t1.join();
    t2.join();
}
```

(j)


```
#include <iostream>
#include <list>
#include <thread>
#include <mutex>

std::list<int> x;
std::mutex m;
std::thread t;

int main()
{
    t = std::thread([](){
        for (;;) {
            std::scoped_lock<std::mutex> g(m);
            if (!x.empty()) {
                std::cout << x.front() << "\n";
                x.pop_front();
            }
        }
    });
    t.detach();
    for (int i = 0; i < 1000; ++i) {
        std::scoped_lock<std::mutex> g(m);
        x.push_back(i);
    }
}
```

(k)

```
#include <thread>
#include <atomic>
#include <cassert>

std::atomic<int> x(0);
std::atomic<bool> done(false);

int main()
{
    std::thread t1([](){
        x = 42;
        done = true;
    });
    std::thread t2([](){
        while (!done) {}
        assert(x == 42);
    });
    t1.join();
    t2.join();
}
```

(l)

```
#include <thread>
#include <iostream>

unsigned long long counter(0);

int main()
```



```
{
    std::thread t1([](){
        for (int i = 0; i < 100000; ++i) {
            ++counter;
        }
    });
    std::thread t2([](){
        for (int i = 0; i < 100000; ++i) {
            ++counter;
        }
    });
    t1.join();
    t2.join();
    std::cout << counter << "\n";
}
```

(m)

```
#include <thread>
#include <mutex>

struct Widget {
    char x;
    char y;
    std::mutex xMutex;
    std::mutex yMutex;
};

Widget w;

int main()
{
    std::thread t1([](){
        {
            std::scoped_lock<std::mutex> lock(w.xMutex);
            w.x = 1;
        }
    });
    std::thread t2([](){
        {
            std::scoped_lock<std::mutex> lock(w.yMutex);
            w.y = 1;
        }
    });
    t1.join();
    t2.join();
}
```

(n)

```
#include <thread>
#include <mutex>

struct Widget {
    int x:9;
    int y:7;
};

```



```
Widget w;
std::mutex xMutex;
std::mutex yMutex;

int main()
{
    std::thread t1([](){
        {
            std::scoped_lock<std::mutex> lock(xMutex);
            w.x = 1;
        }
    });
    std::thread t2([](){
        {
            std::scoped_lock<std::mutex> lock(yMutex);
            w.y = 1;
        }
    });
    t1.join();
    t2.join();
}
```

Threads

7.5 (a) In the code below, the function `runThread` creates a thread, performs some other work by calling the function `doStuff`, and then waits for the created thread to complete execution. What is problematic about the manner in which the `runThread` function is written? (Hint: Note the declaration of the `doStuff` function.)

```
#include <thread>

void doWork();
void doStuff();

void runThread()
{
    std::thread t(doWork);
    doStuff();
    t.join();
}
```

(b) Suggest two possible solutions to the problem identified above.

7.6 The `std::thread` class requires that a user of the class must never leave a `thread` object in a joinable state at the time of destruction. Of course, this leads to the common programming mistake of forgetting to perform a join operation on a thread object prior to destruction. Develop a RAII class for `std::thread` called `scoped_thread`. The `scoped_thread` class should serve as a simple wrapper for a `std::thread` object that automatically performs a join operation (if needed) at the time of destruction. For example, the `scoped_thread` class should allow a program like that shown below to operate correctly.

```
#include <iostream>
#include <thread>

// Replace the following line with the appropriate
// header file for your scoped_thread class.
#include "scoped_thread.hpp"
```



```
7
8  void bonjour()
9  {
10     std::cout << "Bonjour, Le Monde!\n";
11 }
12
13 int main()
14 {
15     // Include extra set of parentheses on the following
16     // line in order to avoid the most-vexing parse problem.
17     scoped_thread t1((std::thread(bonjour)));
18
19     scoped_thread t2(std::thread([](){
20         std::cout << "Hello, World!\n";
21     }));
22 }
```

7.7 Write a program whose `main` function starts two threads that behave as follows. The first thread prints `"Greetings!\n"` to standard output. The second thread performs the following:

(a) Print `"Sleep.\n"` to standard output.
(b) Sleep for 3 seconds.
(c) Print `"Wakeup.\n"` to standard output.

Use `std::osyncstream` to avoid potential data races on `std::cout`.

7.8 Write a program whose `main` function starts three threads that behave as described below. The first thread loops 10 times, performing the following in each iteration:

(a) Print `"tick\n"` to standard output.
(b) Sleep for 1 second.

The second thread loops 5 times, performing the following in each iteration:

(a) Print `"tock\n"` to standard output.
(b) Sleep for 2 seconds.

The third thread simply does the following:

(a) Sleep for 10 seconds.
(b) Print `"chime\n"` to standard output.

Use `std::osyncstream` to avoid potential data races on `std::cout`.

Mutexes

7.9 Writing data to an arbitrary `std::ostream` is not guaranteed to be thread safe in all cases (e.g., unsynchronized streams), and even in the cases where thread safety is guaranteed, the characters output by different threads may interleave arbitrarily. With this in mind, modify the `output` function from Exercise 5.14 so that: 1) thread safety is guaranteed; and 2) the output produced by a single call to the `output` function will not interleave with output produced by calls made in other threads.

7.10 (a) Write a program whose `main` function starts two threads that perform the work described below. For each integer i from -100 to 100 (inclusive), the first thread computes the square of i and prints the result to standard output using a message of the form "The square of i is x." (where $x = i^2$) followed by a newline character. For each integer i from -100 to 100 (inclusive), the second thread computes the cube of i and prints the result to standard output using a message of the form "The cube of i is x." (where $x = i^3$) followed by a newline character. [The lines of output from the two threads are permitted to interleave.]

(b) Modify the program developed in part (a) of this exercise so that each line of output produced by the threads is written to the stream atomically (i.e., the characters in a single line of output do not interleave with any characters written by other threads). [Hint: Use a mutex.]

7.11 Write a program that creates 16 threads, each of which loop 128 times performing the following operations:

(a) Write to standard output the message `"Greetings from thread "` followed by the thread ID (obtained via `std::thread::get_id`) and a newline character. Use a mutex to ensure that accesses to `std::cout` are guaranteed to be properly synchronized and to ensure that each message is output atomically (i.e., without being interleaved with characters from other threads). Use a lock guard to ensure that the mutex resource cannot be leaked.

(b) Sleep for 100 milliseconds.

[Hint: The following will be helpful in this exercise: `std::thread`, `std::this_thread`, `std::mutex`, `std::scoped_lock`, and `std::chrono::milliseconds`.]

7.12 In this exercise, we consider a multithreaded program with two types of threads: a reader thread and writer thread. The `main` function starts one writer thread and three reader threads. All threads share the variable `value` of type **`int`**. The writer thread performs the following for v from 0 to 3 (inclusive):

(a) Set `value` to v.
(b) Print a message to standard output of the form "writer: v" followed by a newline character.
(c) Sleep for 1000 milliseconds.

The three reader threads should be uniquely numbered 0, 1, and 2. Each reader thread loops 20 times, performing the following in each iteration:

(a) Get the value of `value`.
(b) Print to standard output a message of the form "reader r: v" followed by a newline character, where r is the number of the reader thread (i.e., 0, 1, or 2) and v is the value read from `value` in the previous step.
(c) Sleep for 250 milliseconds.

Use a shared mutex for synchronizing access to `value`. This should be done in such a way that the writer thread uses an exclusive lock and each of the reader threads uses a shared lock. In addition, use a (regular) mutex in order to ensure that each line of output produced by the various threads is written atomically to the stream.

Condition Variables

7.13 (a) Develop a class called `IntQueue` that provides a simple thread-safe FIFO queue of **`int`**s. The class should provide the following two public members:

i. The member function `get` is used to remove an integer from the queue. The function blocks the calling thread until an element is available on the queue. Then, the integer from the front of the queue is removed and returned to the caller. This function must be thread safe.
ii. The member function `put` is used to insert an integer in the queue. The function inserts the given integer in the queue. This function must be thread safe.

For the `IntQueue` class, use a `std::list` for storing the elements of the queue. [Hint: The `IntQueue` class will also need a mutex (`std::mutex`) and a condition variable (`std::condition_variable`).]

(b) Write a program that consists of two threads: a producer thread and a consumer thread. These two threads share a single `IntQueue`. The producer thread should loop 1000 times, performing the following steps in each iteration:

i. Put an integer on the queue.
ii. Sleep for 1 millisecond.

After the loop completes, the thread should place the value `-1` on the queue. The consumer thread should performing the following steps:

i. Get an integer from the queue.
ii. Print the integer to standard output.
iii. If the integer is not negative, go to the first step (i.e., get another integer from the queue).

(c) Identify a potentially serious shortcoming of the `IntQueue` class, as specified above. In particular, consider what happens when elements are added to and removed from the queue at different rates.

7.14 In this exercise, we consider a program whose `main` function starts two types of threads: producer and consumer threads. A producer thread generates one or more integer values for use by consumer threads. A consumer thread uses integer values generated by producer threads. In order to transfer the integer data generated by producer threads to consumer threads, two shared (global) variables are employed (excluding synchronization variables). The first variable `data` is of type **`int`** and is used to pass an integer value generated by a producer thread to a consumer thread. The second variable `ready` is of type **`bool`** and is used to indicate whether `data` holds a value that is ready for use by a consumer thread. The preceding two variables are shared amongst all of the producer and consumer threads so that, in effect, these variables operate as a single shared queue with a maximum length of one. Write a program allowing the producer and consumer threads to operate correctly for each of the variants of this exercise listed below. Employ mutexes and condition variables, as appropriate, in order to ensure that access to shared data is properly synchronized.

(a) First variant (a single producer thread transferring a single value to a single consumer thread). In this variant of the exercise (which is the simplest variant), the `main` function should start one producer thread and one consumer thread. The producer thread should generate a single integer value and terminate. The consumer thread should read a single integer value, print the value to standard output, and terminate. [Hint: Your solution should employ one mutex and one condition variable.]

(b) Second variant (a single producer thread transferring multiple values to a single consumer thread). In this variant of the exercise, the `main` function should start one producer thread and one consumer thread. The producer thread should iterate 10000 times, generating each of the values 0 to 9999. The consumer thread should iterate 10000 times, with each iteration reading a value sent by the producer thread and printing the value to standard output. [Hint: Your solution should employ one mutex and two condition variables.]

(c) Third variant (multiple producer threads transferring multiple values to multiple consumer threads). In this variant of the exercise, the `main` function should start four producer threads and four consumer threads. Each producer thread should iterate 10000 times, generating each of the values 0 to 9999. Each consumer thread should iterate 10000 times, with each iteration reading a value generated by a producer thread and printing the value to standard output.

7.15 In this exercise, a multithreaded program is developed with a shared (global) variable `ready` of type **`bool`**. Write a program whose `main` function performs the following:

(a) Set `ready` to **`false`**.
(b) Start 10 threads with `doWork` as the function to execute, passing a different integer value (from 0 to 9) as an argument to `doWork` in each case.
(c) Sleep for 3 seconds.
(d) Set `ready` to **`true`**.
(e) Wait for the threads to finish executing.

The function `doWork` has a single parameter `id` of type **`int`**, and when the function is executed, it simply blocks waiting for `ready` to become **`true`** and then prints the value of `id` to standard output.

Atomics

7.16 Write a program whose `main` function starts one writer thread and one reader thread that behave as described below. All of the threads of the program share a single variable `data` of type **`int`** (excluding synchronization variables) that is used to transfer a single integer value from the writer thread to the reader thread. The writer thread performs the following:

(a) Sleep for 1 second.
(b) Set `data` to 42.

The reader thread performs the following:

(a) Wait until the writer thread has written a value to `data`.
(b) Print the value of `data` to standard output.

Use atomics in order to synchronize access to shared data. (Do not use any mutexes or condition variables.)

7.17 Write a multithreaded program that behaves as described below. The program consists of the main thread and four worker threads. All of the threads of the program share a single variable `counter` that holds an **`unsigned long long`** value. The `main` function should initialize `counter` to zero and then start four worker threads. Each worker thread should loop 20000 times, incrementing `counter` in each iteration. After the four worker threads complete execution, the main thread should print the value of `counter` to standard output. Use atomics in order to properly synchronize access to any shared variables.

7.18 Write a program with the behavior described below. The `main` function starts several threads, henceforth referred to as worker threads, that generate random numbers (in a manner to be explained in more detail shortly). All of the threads share the global variable `globalMaxValue` of type `std::atomic<`**`long`**`>`. This variable is used to track the largest random number generated by all of the threads so far during program execution. To begin, the `main` function should initialize `globalMaxValue` to `std::numeric_limits<`**`long`**`>::min()` and then start four worker threads. Each worker thread should loop 10000 times, performing the following in each iteration:

(a) Generate a random integer r using the `getRandom` function given below.
(b) If r is greater than `globalMaxValue` then set `globalMaxValue` to r. In other words, update `globalMaxValue` to accurately reflect the largest random number generated so far by all threads. Of course, this update must be properly synchronized.

After iteration completes, the worker thread should output to standard output the maximum random value v that the thread itself generated. This should be done using a message of the form "thread i: v" followed by a newline character, where i is a unique integer identifying the thread (i.e., 0, 1, 2, and 3). After all of the worker threads finish execution, the main thread should print to standard output the value of `globalMaxValue` using a message of the form "all: v" followed by a newline character, where v is value of `globalMaxValue`. [Hint: In order to update `globalMaxValue`, use `compare_exchange_strong` or `compare_exchange_weak`.]

All of the threads should use the `getRandom` function listed below in order to generate integers uniformly distributed on $[0, 999999]$. This function is thread safe and ensures that each thread will generate a different random number sequence.

```
#include <random>

// Get a random integer uniformly distributed on [0, 999999].
long getRandom()
{
    thread_local static std::default_random_engine
      gen((std::random_device()()));
    thread_local static std::uniform_int_distribution<long>
      dist(0, 999'999L);
    return dist(gen);
}
```


Latches and Barriers

7.19 A latch is basic one-time synchronization mechanism that allows threads to block until a particular event occurs a certain number of times. A latch maintains an internal count that is initialized to some nonzero value when the latch is created. Operations are provided by a latch to allow a thread to: 1) decrement the latch's count; and 2) block until the latch's count reaches zero. A latch can only be used once. In other words, the latch's count can never be changed again after it reaches zero. Write a `latch` class that represents a latch and has the following functionality:

- the constructor takes a single integer argument which is the initial count for the latch;
- the `count_down` member function decrements the latch's count;
- the `wait` member function blocks the calling thread until the latch's count reaches zero;
- the `count_down_and_wait` member function decrements the latch's count and then blocks the calling thread until the latch's count reaches zero;
- the `try_wait` member function returns true if the latch's count has reached zero and false otherwise;
- the class is not movable or copyable.

Use mutexes and condition variables as appropriate.

7.20 A barrier is basic synchronization mechanism that allows threads to block until a specific number of threads are waiting. A barrier maintains an internal thread counter that is initialized when the barrier is created. A thread can decrement the counter and then block waiting until the counter reaches zero. When the barrier's count reaches zero, all threads waiting on the barrier are awoken and the barrier's count is reset to its initial value. Develop a `barrier` class that provides the functionality of a barrier and has the following characteristics:

- the constructor takes a single integer argument that is the initial count value for the barrier;
- the `count_down_and_wait` member function decrements the barrier's count by one; if the resulting count is not zero, the calling thread is blocked until the count reaches zero; otherwise, the count is reset to its initial value and all threads waiting on the barrier are awoken.

Use mutexes and condition variables as appropriate.

Miscellany

7.21 In this exercise, we consider the testing of integers for primeness. Write a function `isPrime` that tests if an integer is prime. The function should take an **`unsigned long`** argument corresponding to the integer to be tested and return a value of type **`bool`** indicating if the integer is prime. The primality test need not be implemented in any sophisticated way. For example, to test if the integer i is prime, an algorithm that checks for divisibility by odd numbers from 3 to $i-1$ (inclusive) would suffice.

Write a multithreaded program that employs the `isPrime` function in order to test the following four numbers for primeness:

(a) 4,294,967,291
(b) 4,294,967,279
(c) 4,294,967,231
(d) 4,294,967,197

(If your computer is very slow, you may alternatively consider the following four integers: 16,777,213; 16,777,199; 16,777,183; and 16,777,153.) Each of the above numbers should be tested for primeness in a separate thread with the result of the primality test being made available to the main thread (i.e., the thread executing the `main` function) via a future. From the main thread, for each number v being tested, print to standard output a message of the form "The number v is prime." or "The number v is not prime.", as appropriate, followed by a newline character. The above program is to be written using each of the two approaches listed below.

(a) First approach. Use the `std::async` function to launch the threads to perform the desired computation.
(b) Second approach. Use the `std::packaged_task` template class to assist in launching threads to perform the desired computation. (Do not use the `std::async` function in this case.)

Chapter 8

Miscellany

8.1 Exercises

Value Categories

8.1 Consider the program whose source code is given below. In each case where a line of code is followed by one or more comments containing an expression (from that line of code), indicate whether the expression is an lvalue or rvalue (i.e., prvalue or xvalue).

```
#include <iostream>
#include <vector>
#include <utility>

std::vector<int>&& func1(std::vector<int>& x) {
    return static_cast<std::vector<int>&&>(x);
    // x?
    // static_cast<std::vector<int>&&>(x)?
}

int main() {
    std::vector<int> x = {1, 2, 3};
    std::vector<int> y;
    int a;

    for (auto i = x.begin(); i != x.end(); ++i) {
        // x.begin()?
        // ++i?
        *i += 5;
        // i?
        // *i?
        // *i += 5?
    }

    a = x[0];
    // x[0]?
    ++a; a++;
    // ++a?
    // a++?

    y = func1(x);
    // func1(x)?
```

```
    // y = func1(x)?
}
```

8.2 Consider the program whose source code is given below. In each case where a line of code is followed by one or more comments containing an expression (from that line of code), indicate whether the expression is an lvalue or rvalue (i.e., prvalue or xvalue).

```
#include <iostream>
#include <vector>
#include <utility>

std::vector<float>& half(std::vector<float>& a)
{
    for (auto&& x : a) {
        x *= 0.5f;
          // 0.5f
          // x *= 0.5f
    }
    return a;
}

std::vector<float> get_data()
{
    return std::vector<float>{1.0f, 0.5f, 0.25f, 0.125f};
      // std::vector<float>{1.0f, 0.5f, 0.25f, 0.125f}
}

int main() {
    std::vector<float> a{1.0f, 2.0f, 3.0f};
    std::vector<float> b(std::move(a));
    float x = 0.0f;
    float y = 42.0f;
    float* p = nullptr;
      // nullptr
    a = std::move(b);
      // std::move
      // std::move(b)
      // a = std::move(b)
    b = {1.0f, 2.0f, 3.0f};
      // {1.0f, 2.0f, 3.0f}
    half(b);
      // half(b)
    x += (- a[0] + 2.0f * a[1] - 42) / 3.0f;
      // 42
      // - a[0]
      // - a[0] + 2.0f * a[1] - 42
    x *= y + y;
      // y + y
    p = a.data();
      // a.data()
    x = *p + 1.0f;
      // *p
      // x = *p + 1.0f
    b = get_data();
      // get_data()
    std::cout << a[0] << '\n';
```



```
50        // std::cout
51        // std::cout << a[0]
52        // '\n'
53      std::vector<float>::iterator i = a.begin();
54      std::vector<float>::iterator j;
55      ++i;
56        // ++i
57      j = i;
58        // j = i
59      i--;
60        // i--
61      std::cout << j - i << '\n';
62        // j - i
63      std::cout << *i << "\n";
64        // *i
65        // "\n"
66  }
```

Overload Resolution

8.3 In the program whose source code is given below, the overloaded function `func` is invoked several times. For each function invocation, state which overload is called.

```
1  #include <utility>
2  #include <string>
3
4  void func(std::string&);
5  void func(const std::string&);
6  void func(std::string&&);
7  void func(const std::string&&);
8
9  const std::string getConstString()
10 {
11     return std::string("Bjarne");
12 }
13
14 int main()
15 {
16     const std::string cs("Hello");
17     std::string s = cs;
18     func(cs);
19     func(s);
20     func(s + "!");
21     func(s + s);
22     func(std::move(s));
23     func(getConstString());
24 }
```

8.4 Consider the program with the source code listing shown below. The function `main` makes several calls to the overloaded functions `func1` and `func2`. Determine which overload is invoked in each case and, in the process of doing so, determine the output that the program will produced when executed.

```
1  #include <complex>
2  #include <iostream>
3
```



```
4  typedef std::complex<double> Complex;
5
6  const Complex getConst()
7  {
8      return Complex(1.0, 2.0);
9  }
10
11 void func1(const Complex& a)
12 {
13     std::cout << "func1(const Complex&) called\n";
14 }
15
16 void func1(Complex& a)
17 {
18     std::cout << "func1(Complex&) called\n";
19 }
20
21 void func1(Complex&& a)
22 {
23     std::cout << "func1(Complex&&) called\n";
24 }
25
26 void func2(const Complex& a)
27 {
28     std::cout << "func2(const Complex&) called\n";
29 }
30
31 void func2(const Complex&& a)
32 {
33     std::cout << "func2(const Complex&&) called\n";
34 }
35
36 int main()
37 {
38     const Complex j(0.0, 1.0);
39     Complex a(1.0, 1.0);
40     const Complex* p = &j;
41
42     func1(a);
43     func1(j);
44     func1(a * a);
45     func1(getConst());
46     func1(*p);
47
48     func2(a + a);
49     func2(j);
50     func2(getConst());
51 }
```

References, Reference Binding, and Reference Collapsing

8.5 For each reference binding operation in the code fragment below, state whether the operation is legal, and if not why.

```
1  double getDouble();
2  const double cd = 42.0;
```



```
3  double d = cd;
4
5  double& lr1 = d;
6  double& lr2 = cd;
7  double& lr3 = getDouble();
8
9  const double& lrc1 = d;
10 const double& lrc2 = cd;
11 const double& lrc3 = 42.0;
12
13 double&& rr1 = d;
14 double&& rr2 = cd;
15 double&& rr3 = getDouble();
16
17 const double&& rrc1 = d;
18 const double&& rrc2 = cd;
19 const double&& rrc3 = 42.0;
```

8.6 For each of the identifiers `T1`, `T2`, . . ., `T16`, state the type to which the identifier corresponds.

```
1  typedef    char& LvRefChar;
2  typedef const char& LvRefConstChar;
3  typedef    char&& RvRefChar;
4  typedef const char&& RvRefConstChar;
5
6  typedef    LvRefChar& T1;
7  typedef const LvRefChar& T2;
8  typedef    LvRefChar&& T3;
9  typedef const LvRefChar&& T4;
10
11 typedef    LvRefConstChar& T5;
12 typedef const LvRefConstChar& T6;
13 typedef    LvRefConstChar&& T7;
14 typedef const LvRefConstChar&& T8;
15
16 typedef    RvRefChar& T9;
17 typedef const RvRefChar& T10;
18 typedef    RvRefChar&& T11;
19 typedef const RvRefChar&& T12;
20
21 typedef    RvRefConstChar& T13;
22 typedef const RvRefConstChar& T14;
23 typedef    RvRefConstChar&& T15;
24 typedef const RvRefConstChar&& T16;
```

8.7 In the code fragment below, the template function `func` is invoked several times. For each invocation of `func`, state the type deduced for the template parameter `T` as well as the type of the function parameter `p`.

```
1  #include <utility>
2
3  // ... (includes declaration of func2)
4
5  template <class T>
6  void func(T&& p)
7  {
```



```
8       func2(std::forward<T>(p));
9   }
10
11  int main()
12  {
13      const double cd = 42.42;
14      double d = cd;
15
16      func(d);
17      func(cd);
18      func(d + d);
19      func(std::move(d));
20  }
```

Temporaries

8.8 Consider the program with the source listing given below. For the purposes of this exercise, assume that no optimization is performed by the compiler. Identify each temporary object that is needed during the execution of the the `main` function. Only temporary objects local to the `main` function need be considered. For each temporary object, identify the line number where it is created and explain why it is needed.

```
1   int sqr(int x)
2   {
3       return x * x;
4   }
5
6   int main()
7   {
8       int x = 0;
9       int y;
10      int z;
11
12      y = ++x;
13      x = y++;
14      z = x + y;
15      x = sqr(z) + 42;
16  }
```

8.9 Consider the program with the source listing given below. For the purposes of this exercise, assume that no optimization is performed by the compiler. Identify each temporary object of type `counter` in the `main` function that results from the code on the lines marked by a comment `/* ??? */`. For each temporary object, identify the line of code where the temporary object is created and provide a brief description of the temporary object along with an explanation of why it is required.

```
1   #include <iostream>
2
3   // A counter class.
4   class counter {
5   public:
6
7       // The counter value type.
8       using value_type = std::size_t;
9
10      counter(value_type x = value_type(0)) : value_(x) {}
```



```
11
12      counter(const counter& x) = default;
13      counter& operator=(const counter& x) = default;
14      ~counter() = default;
15
16      // Increment the counter.
17      counter& operator++()
18      {
19          ++value_;
20          return *this;
21      }
22
23      // Increment the counter.
24      counter operator++(int)
25      {
26          counter old(*this);
27          ++value_;
28          return old;
29      }
30
31      // Add the value of another counter to the counter.
32      counter& operator+=(const counter& x)
33      {
34          value_ += x.value_;
35          return *this;
36      }
37
38      // Get the value of the counter.
39      value_type get_value() const
40      {
41          return value_;
42      }
43
44  private:
45
46      // The counter value.
47      value_type value_;
48  };
49
50  // Add two counters.
51  counter operator+(const counter& x, const counter& y)
52  {
53      return counter(x) += y;
54  }
55
56  // Output a counter to a stream.
57  std::ostream& operator<<(std::ostream& out, const counter& x)
58  {
59      return out << x.get_value();
60  }
61
62  int main() {
63      counter x;
64      counter y;
65      counter z;
66      // ... (code omitted here changes x, y, and z)
67      z = x + y; /* ??? */
```



```
    z = z + z; /* ??? */
    y = ++z;  /* ??? */
    z = y++;  /* ??? */
    x = z;    /* ??? */
    std::cout << x << ' ' << y << ' ' << z << '\n';
}
```

Copying, Moving, and Copy Elision

8.10 Copy and move constructors employ pass by reference. Explain why this must be so (i.e., why pass by value cannot be used).

8.11 Consider the program whose source code is given below. In the `main` function, move/copy construction and move/copy assignment for the `SharedInt` class is performed in several places. Identify each of these places, specifically indicating whether a copy or move is employed. Assume that the compiler does not perform any optimization.

```
#include <iostream>
#include <memory>

class SharedInt
{
public:
    // default constructor
    SharedInt(int i = 0) : p_(std::make_shared<int>(i)) {}

    // copy constructor
    SharedInt(SharedInt& i) : p_(i.p_) {}

    // move constructor
    SharedInt(SharedInt&& i) : p_(std::move(i.p_)) {}

    // copy assignment operator
    SharedInt& operator=(const SharedInt& i)
    {
        if (this != &i) {
            p_ = i.p_;
        }
        return *this;
    }

    // move assignment operator
    SharedInt& operator=(SharedInt&& i)
    {
        p_ = std::move(i.p_);
        return *this;
    }

    // get the underlying integer value
    int get() const
    {
        return *p_;
    }

    // compute the square
```



```
39      SharedInt square() const
40      {
41          return SharedInt((*p_) * (*p_));
42      }
43
44      // increment
45      SharedInt& operator++()
46      {
47          ++(*p_);
48          return *this;
49      }
50  private:
51      // shared pointer to integer
52      std::shared_ptr<int> p_;
53  };
54
55  int main()
56  {
57      SharedInt x(2);
58      SharedInt y(x.square());
59      SharedInt z(x);
60      SharedInt w(std::move(y));
61      y = std::move(w);
62      w = y.square();
63      x = y;
64      ++z;
65      std::cout << w.get() << " " << x.get() << " " << y.get() << " " <<
66        z.get() << "\n";
67  }
```

8.12 Identify all copy and move operations (i.e., copy construction, move construction, copy assignment, and move assignment), if any, associated with each line of code marked by a comment "`???`" in the program below. Be sure to distinguish between cases where a copy/move operation is: 1) required with certainty (since it cannot possibly be elided); and 2) may be required (depending on whether or not it can be elided). Also, identify cases where a copy/move operation is guaranteed to be elided.

```
1   #include <cstdlib>
2   #include <utility>
3
4   class Widget {
5   public:
6       Widget(int value = 0) : value_(value) {}
7       ~Widget() = default;
8       Widget(Widget&&) = default;
9       Widget(const Widget&) = default;
10      Widget& operator=(Widget&&) = default;
11      Widget& operator=(const Widget&) = default;
12      int get_value() const {return value_;}
13  private:
14      int value_;
15  };
16
17  Widget make_widget_1()
18  {
19      if (std::rand() % 2) {
20          return Widget(1);
```



```
    } else {
        return Widget(0);
    }
}

Widget make_widget_2(bool b)
{
    Widget w(0);
    Widget x(1);
    if (b) {
        return x;
    } else {
        return w;
    }
}

int func_1(Widget w)
{
    return w.get_value();
}

int func_2(Widget&& w)
{
    return w.get_value();
}

int main() {
    Widget a;
    Widget b(a); // ???
    Widget c = a; // ???
    Widget d(std::move(c)); // ???
    Widget e = std::move(d); // ???
    Widget f(make_widget_1()); // ???
    Widget g(make_widget_2(true)); // ???
    c = a; // ???
    b = std::move(c); // ???
    a = make_widget_1(); // ???
    a = make_widget_2(true); // ???
    func_1(a); // ???
    func_1(std::move(a)); // ???
    func_1(make_widget_1()); // ???
    func_2(std::move(b)); // ???
}
```

8.13 Consider the `get_data` function in the source listing shown below. The particular use of `std::move` in the return statement of this function is problematic. Explain why such a usage of `std::move` is problematic and how to fix it.

```
#include <utility>
#include <vector>

std::vector<float> get_data()
{
    return std::move(std::vector<float>{1.0f, 0.5f, 0.25f, 0.125f});
}
```


8.14 Consider the code for the `make_set`, `make_vector`, and `make_mutexes` functions in the source listing below. In each case, the function is written in such a way that copy elision is not mandatory. This means that, for each of these functions, propagating the return value back to the caller may require a copy/move operation. For each function, explain why copy elision is not mandatory. Then, rewrite the function so that copy elision is mandatory, thus, eliminating any possibility of using a copy/move operation to propagate the return value back to the caller.

```
#include <list>
#include <mutex>
#include <set>
#include <utility>
#include <vector>

std::set<int> make_set()
{
    std::set<int> s{1, 2, 4, 8};
    return s;
}

std::vector<int> make_vector(bool flag)
{
    std::vector<int> a{1, 2, 3, 4};
    std::vector<int> b{5, 6, 7, 8};
    if (flag) {
       return a;
    } else {
       return b;
    }
}

std::list<std::mutex> make_mutexes(int n)
{
    return std::move(std::list<std::mutex>(n));
}
```

8.15 For each line of code marked by a comment "`???`" in the program below, identify all copy and move operations (i.e., copy construction, move construction, copy assignment, and move assignment) associated with objects of type `std::vector<float>` (if any). Be sure to distinguish between cases where a copy/move operation is: 1) required with certainty (since it cannot possibly be elided); and 2) may be required (depending on whether or not it can be elided). Also, identify cases where a copy/move operation is guaranteed to be elided (i.e., mandatory copy elision).

```
#include <algorithm>
#include <functional>
#include <random>
#include <utility>
#include <vector>

std::vector<float> make_vector_1()
{
    return std::vector<float>(42, 42.0f);
}

std::vector<float> make_vector_2()
{
    std::default_random_engine gen;
```



```
    std::uniform_real_distribution<float> dist(0.0f, 10.0f);
    std::vector<float> a(10);
    std::generate(a.begin(), a.end(), [&](){return dist(gen);});
    return a;
}

std::vector<float> make_vector_3()
{
    return std::move(std::vector<float>{1.0f, 2.0f, 3.0f, 4.0f});
}

float sum_1(const std::vector<float>& a)
{
    return std::accumulate(a.begin(), a.end(), 0.0f);
}

float sum_2(std::vector<float> a)
{
    return std::accumulate(a.begin(), a.end(), 0.0f);
}

std::vector<float> a(make_vector_1()); // ???
std::vector<float> b(make_vector_2()); // ???
std::vector<float> c(make_vector_3()); // ???

int main()
{
    std::vector<float> v(a); // ???
    std::vector<float> w(std::move(c)); // ???
    std::vector<float> x(make_vector_1()); // ???
    std::vector<float> y(make_vector_2()); // ???
    std::vector<float> z(make_vector_3()); // ???
    v = std::move(b); // ???
    w = v; // ???
    w = make_vector_1(); // ???
    w = make_vector_2(); // ???
    w = make_vector_3(); // ???
    float s;
    s = sum_1(make_vector_1()); // ???
    s = sum_2(make_vector_1()); // ???
}
```

Algorithms and Data Structures

8.16 A binary tree is used to represent a set of integers. The partial source code for this tree type is shown below. A node in the tree is represented using the `Node` type. A function `find_element` is provided. This function finds a node that holds a specified value `v` by searching in a given node `n` and all of its descendants. If a node containing the desired value is found, a pointer to the node is returned; otherwise, a null pointer is returned. (To search the entire tree, the value of `n` would simply be chosen as the root node of the tree.)

```
// Tree node type.
struct Node {
    Node* parent; // pointer to parent
    Node* left; // pointer to left child
    Node* right; // pointer to right child
```



```
6       int value; // value
7   };
8
9   // Find tree node that contains specified element.
10  Node* find_element(Node* n, int v) {
11      if (v == n->value) {
12          // We have found the desired element.
13          return n;
14      } else if (v < n->value) {
15          // The element if present must be in the left subtree.
16          return n->left ? find_element(n->left, v) : nullptr;
17      } else {
18          // The element if present must be in the right subtree.
19          return n->right ? find_element(n->right, v) : nullptr;
20      }
21  }
```

(a) If the tree is balanced and contains n nodes, determine the asymptotic time complexity of `find_element` (as a function of n). (A tree is balanced if the depth of each leaf node is $O(\log n)$.)

(b) Determine the time complexity of the function if the tree contains n nodes but is not balanced.

[Hint: In both parts of this exercise, simply consider the number of nodes in the tree that must be visited in the worst case.]

8.17 Consider the function `sum_lower_triangle` whose source code is given below. This function sums the elements in the lower triangular part of an $n \times n$ matrix and returns the result.

(a) Determine the asymptotic time complexity of this function (where the problem size is n).

(b) Determine the asymptotic space complexity of this function (where the problem size is n).

```
1   template <int n, class T>
2   T sum_lower_triangle(const T (&a)[n][n]) {
3       T sum(0);
4       for (int i = 0; i < n; ++i) {
5           for (int j = 0; j <= i; ++j) {
6               sum += a[i][j];
7           }
8       }
9       return sum;
10  }
```

8.18 Consider the two functions `reverse_array_1` and `reverse_array_2` whose source code is given below. Each of these functions reverses the elements in a one-dimensional array.

```
1   #include <utility>
2
3   template <class T>
4   void reverse_array_1(T* a, int n)
5   {
6       for (int i = 0; i < n / 2; ++i) {
7           std::swap(a[i], a[n - 1 - i]);
8       }
9   }
```



```
#include <vector>

template <class T>
void reverse_array_2(T* a, int n)
{
    // Copy the array elements into a vector.
    std::vector<T> v(&a[0], &a[n]);
    // Copy the elements from the vector to the array in reverse order.
    for (int i = 0; i < n; ++i) {
        a[i] = v[n - 1 - i];
    }
}
```

(a) Find the asymptotic time and asymptotic space complexities of `reverse_array_1`. State any assumptions made in your answer.

(b) Find the asymptotic time and asymptotic space complexities of `reverse_array_2`. State any assumptions made in your answer.

(c) Which function would be preferable to use, based on asymptotic complexity analysis? Explain your answer.

8.19 Consider the two functions `factorial_1` and `factorial_2` whose source code is given below. Each of these functions computes a factorial.

```
unsigned long long factorial_1(unsigned int n)
{
    unsigned long long result = 1;
    for (auto i = n; i > 1; --i) {
        result *= i;
    }
    return result;
}
```

```
unsigned long long factorial_2(unsigned int n)
{
    if (n >= 2) {
        return n * factorial_2(n - 1);
    } else {
        return 1;
    }
}
```

(a) Find the asymptotic time and asymptotic space complexities of `factorial_1`.

(b) Find the asymptotic time and asymptotic space complexities of `factorial_2`.

(c) Which of these two functions would be preferable to use based on asymptotic complexity analysis? Explain your answer.

8.20 Consider the function template `recursive_sum` whose source code is given below. This function template computes the sum of the `n` elements of type `T` stored in the array pointed to by `first`.

```
template <class T>
T recursive_sum(const T* first, std::size_t n)
{
    if (n >= 2) {
```



```
5          std::size_t m = n / 2;
6          return recursive_sum(first, m) +
7            recursive_sum(first + m, n - m);
8       } else {
9          return *first;
10      }
11  }
```

(a) Determine the asymptotic time complexity of this function, where the problem size is `n`. (Assume that, for elements of type `T`, the addition operation is a constant-time algorithm.)

(b) Determine the asymptotic space complexity of this function, where the problem size is `n`.

8.21 The Hamming weight of an integer is the number of one digits in the binary representation of the number. Consider the following function for computing the Hamming weight of an integer.

```
1  unsigned int hamming_1(unsigned int x)
2  {
3     unsigned int result = 0;
4     while (x != 0) {
5        // Is the least significant bit nonzero?
6        if ((x & 1) != 0) {
7           // One more nonzero bit has been found.
8           ++result;
9        }
10       // Shift the bits in the value right by one position.
11       x >>= 1;
12    }
13    return result;
14 }
```

(a) Determine the asymptotic time and asymptotic space complexities of the `hamming_1` function, where the input problem size is the number n of bits in the integer whose Hamming weight is to be computed.

(b) As it turns out, the Hamming weight of an integer can be computed with a lower asymptotic time complexity than that achieved by `hamming_1`. Write a function `hamming_2` that implements such an algorithm. Identify the advantages and disadvantages of the new algorithm (used by `hamming_2`) relative to the original one (used by `hamming_1`).

(c) Comment on the reasonableness of using asymptotic analysis in situations like the one in this exercise.

8.22 Explain the difference between worst-case and amortized complexity. Give an example of a situation in which one might care more about: 1) worst-case complexity (relative to amortized complexity); 2) amortized complexity (relative to worst-case complexity).

8.23 Three parts A, B, C of a program have been identified as potential bottlenecks. Measurements have shown that the percentage of time spent in each of these parts of the code is as follows:

Part	Fraction of Time
A	5%
B	50%
C	10%

It is believed that through additional optimization the speedup achievable for each part of the code is as follows:

Part	Speedup Factor
A	10
B	1.05
C	3

There is only enough time before the next product release to optimize one of the three parts of the code. Which one should by optimized? Justify your answer.

8.24 In each of the following scenarios, identify the type of container that should be used to store the collection in question, and justify your choice, being careful to state any important assumptions made. (The justification of your choice is critically important, since more than one correct answer is possible.)

(a) A program needs to maintain a large set of integers using a container. The complete contents of the container are known at the time that the container is first created and do not change subsequently. The container is very frequently searched for particular values. The ability to iterate over the container elements in sorted order is frequently needed.

(b) A program needs to maintain a large set of integers using a container. The container contents are modified very frequently during program execution by inserting and removing elements. The values to be inserted in, and removed from, the container follow no particular pattern. The container is frequently searched for particular values. The ability to iterate over the container elements in sorted order is frequently needed.

(c) A program needs to maintain a large set of real numbers using a container. The maximum size of this set is known at the time that the container is created. The container contents are modified very frequently during program execution by inserting and removing elements. While the values inserted into the container follow no particular pattern, each element removed from the container is always the one with the largest value.

8.25 Some container types have the property that they provide stable element references. Some container types have the property that they provide stable iterators. For each of these properties, explain what the property means and why one might care whether a container type has the property. Also, give an example of a container that has the property and one that does not.

8.26 Explain why one might want to separate the operations of memory allocation and construction. Similarly, explain why one might want to separate the operations of destruction and memory deallocation.

8.27 Identify two advantages that array-based containers have over node-based containers.

8.28 Discuss the trade offs between (i.e., relative advantages and disadvantages of) array-based and node-based implementations of a stack.

8.29 In each of the following scenarios, indicate whether an intrusive or nonintrusive container should be used to store each collection in question, and justify your choice, being careful to state any important assumptions made.

(a) A operating system kernel represents a process (i.e., a thread plus the resources needed for its execution) using a type called `process`. The `process` type has a very substantial amount of state (and is neither movable nor copyable). The operating system must maintain two containers. The first holds all existing processes (i.e., the process list). The second holds all processes that are ready to run (i.e., the scheduling queue), which is used for process scheduling.

(b) A program needs to maintain a collection of mutexes of type `std::mutex` in a container. Due to other design constraints, none of the nonintrusive containers that are available for use provide stable references.

(c) A program needs to maintain a collection of mutexes of type `std::mutex` in a container. Nonintrusive containers are available that allow in-place construction and provide stable references. The mutexes only need to exist inside the container.

8.30 Draw the control-flow graph for the evaluation of each of the boolean expressions given below, assuming that only built-in operators are used.

(a) `a || (b && c)`
(b) `a && (b || c)`
(c) `(a || b) && (c || d)`
(d) `(a && b) || (c && d)`
(e) `(a || b) && c`
(f) `(a && b) || c`

8.31 Draw the control-flow graph for each of the functions defined in the source listings below.

(a)

```
int abs(int x)
{
    return (x < 0) ? -x : x;
}
```

(b)

```
// Precondition: min_value <= max_value
template <class T>
inline T clip(const T& x, const T& min_value, const T& max_value)
{
    if (x < min_value) {
        return min_value;
    } else if (x > max_value) {
        return max_value;
    }
    return x;
}
```

(c)

```
// Computes the ceiling of log base 2 of x (i.e., ceil(log2(x))).
// Precondition: x > 0
int ceilLog2(int x)
{
    int n = 0;
    --x;
    while (x > 0) {
        x >>= 1;
        ++n;
    }
    return n;
}
```

(d)

```
int hamming_weight(unsigned int n)
{
    int weight = 0;
    while (n) {
        if (n & 1) {
            ++weight;
        }
        n >>= 1;
```



```
    }
    return weight;
}
```

(e)

```
#include <tuple>
#include <limits>

std::pair<unsigned int, bool> reverse_digits(unsigned int x)
{
    unsigned int result = 0;
    while (x) {
        unsigned int d = x % 10;
        if (result > (std::numeric_limits<unsigned int>::max() - d) / 10) {
            return {0, false};
        }
        result = result * 10 + d;
        x /= 10;
    }
    return {result, true};
}
```

(f)

```
#include <limits>
#include <tuple>

// T is a signed integral type
template <class T>
std::pair<T, bool> safe_signed_multiply(T x, T y)
{
    constexpr auto min = std::numeric_limits<T>::min();
    constexpr auto max = std::numeric_limits<T>::max();
    if (x > 0) {
        if (y > 0) {
            // x and y are both positive
            if (x > max / y) {
                return {max, false};
            }
        } else {
            // x is positive, y is nonpositive
            if (y < min / x) {
                return {min, false};
            }
        }
    } else {
        if (y > 0) {
            // x is nonpositive, y is positive
            if (x < min / y) {
                return {min, false};
            }
        } else {
            // x and y are both nonpositive
            if (x != 0 && y < max / x) {
                return {max, false};
            }
        }
```



```
34        }
35        return {x * y, true};
36    }
```

Caches and Virtual Memory

8.32 A 4-way set associative cache has a size of 32 KB and a block size of 64 bytes. The number of bits in an address is 32. Determine the number of bits for the tag, index, and block offset.

8.33 Consider a 2-way set associative cache with a 14-bit tag, 8-bit index, and 4-byte block size. Suppose that all of the valid cache entries are as shown in the table.

Tag	Index	Data			
		Byte 00_2	Byte 01_2	Byte 10_2	Byte 11_2
01010101011110_2	00000000_2	DE_{16}	AD_{16}	BE_{16}	EF_{16}
11110110110111_2	00000000_2	ED_{16}	DA_{16}	EB_{16}	FE_{16}
01010101011110_2	00000001_2	$A0_{16}$	$A1_{16}$	$A2_{16}$	$A3_{16}$
11111111000000_2	00000001_2	$0A_{16}$	$1A_{16}$	$2A_{16}$	$3A_{16}$
01010101011110_2	01111110_2	90_{16}	91_{16}	92_{16}	93_{16}
11111001011110_2	01111110_2	09_{16}	19_{16}	29_{16}	39_{16}
01010101011110_2	01111111_2	$B0_{16}$	$B1_{16}$	$B2_{16}$	$B3_{16}$
11110000111100_2	01111111_2	$0B_{16}$	$1B_{16}$	$2B_{16}$	$3B_{16}$
01010101011110_2	10000000_2	$C0_{16}$	$C1_{16}$	$C2_{16}$	$C3_{16}$
01100110001011_2	10000000_2	$0C_{16}$	$1C_{16}$	$2C_{16}$	$3C_{16}$
01010101011110_2	10000001_2	$D0_{16}$	$D1_{16}$	$D2_{16}$	$D3_{16}$
11010100010110_2	10000001_2	$0D_{16}$	$1D_{16}$	$2D_{16}$	$3D_{16}$
01010101011110_2	11111111_2	$E0_{16}$	$E1_{16}$	$E2_{16}$	$E3_{16}$
11110110101111_2	11111111_2	$0E_{16}$	$1E_{16}$	$2E_{16}$	$3E_{16}$

(a) Determine the number of blocks in the cache.

(b) Determine the number of blocks in memory.

(c) Determine the cached value for the byte at address $557A02_{16}$ if present.

(d) Determine the cached value for the byte at address $FFFFFF_{16}$ if present.

8.34 Consider the code fragments A and B, as given in Listings 8.1 and 8.2, where the variable `a` is declared as

```
double a[1024][1024];
```

Determine the number of cache misses that occurs during the execution of each of the code fragments A and B, subject to the following assumptions:

- the system has a 8 KB direct-mapped cache with a block size of 64 bytes;
- the cache is initially empty;
- the variables `sum`, `i`, and `j` are kept in registers for the duration of the code fragment execution, so that accesses to these variables do not impact caching;
- an object of type **`double`** requires 8 bytes of storage (i.e., **`sizeof`**`(`**`double`**`)` is 8);
- the array `a` is aligned on a 64-byte boundary; and
- while the code fragment is running, no other code executes.

Listing 8.1: Code fragment A

```
sum = 0.0;
for (int i = 0; i < 1024; ++i) {
    for (int j = 0; j < 1024; ++j) {
        sum += a[i][j];
    }
}
```

Listing 8.2: Code fragment B

```
sum = 0.0;
for (int i = 0; i < 1024; ++i) {
    for (int j = 0; j < 1024; ++j) {
        sum += a[j][i];
    }
}
```

8.35 Consider the code fragment whose source listing is given below, where `n` is a compile-time constant (of type **`int`**). For each of the following values for `n`, determine the cache miss rate during the execution of the given code fragment: (a) 2048 and (b) 2064. For the purposes of this analysis, assume that:

- the cache is 8 KiB in size and 2-way set associative and has a block size of 64 bytes;
- the cache has a write-hit policy of write back, a write-miss policy of write allocate, and a replacement policy of LRU;
- the cache is initially empty;
- an object of type **`float`** requires 4 bytes of storage (i.e., **`sizeof`**`(`**`float`**`)` is 4);
- the array `a` is aligned on a 4096-byte boundary;
- the arrays `a`, `b`, and `c` are stored contiguously in memory in that order (with no padding between them);
- the variables `i` and `j` are kept in registers for the duration of the code fragment execution, so that accesses to these variables do not impact caching;
- while the code fragment is executing, no other code executes; and
- the compiler does not apply any code transformations (e.g., optimizations) that would change the order of memory accesses from what is shown in the source listing.

Listing 8.3: Variable declarations for code fragment

```
constexpr int n = /* ... */;
float a[4][n];
float b[4][n];
float c[4][n];
```

Listing 8.4: Code fragment

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < 4; ++j) {
        c[j][i] = a[j][i] + b[j][i];
    }
}
```

8.36 Consider a system with 24-bit virtual addresses, 16-bit physical addresses, and a 1 KB page size. Determine the number of virtual and physical pages, and the number of bits in a virtual page number, physical page number, and page offset.

8.37 Consider a system where the sizes of a virtual page number (VPN), physical page number (PPN), and page offset (PO) are 14, 6, and 10 bits, respectively. Suppose that the page table contains the following information for address translation and protection:

VPN	PPN	Flags
00000000000001_2	001000_2	present, readable, not writable, executable
00000000000010_2	001001_2	present, readable, not writable, not executable
00000000000011_2	001010_2	present, readable, writable, not executable
11111111111100_2	001111_2	present, readable, writable, not executable
11111111111101_2	001110_2	present, readable, writable, not executable
11111111111110_2	001101_2	present, readable, writable, not executable
11111111111111_2	001100_2	present, readable, writable, not executable

Determine the result of the address translation and protection check for each of the following accesses to memory:

(a) a data read (1 byte) at address $00000000000000000000000_2$;
(b) a data write (2 bytes) at address 00000000000011001100_2; and
(c) an instruction fetch (2 bytes) at address $11111111111110000000000_2$.

Containers and Iterators

8.38 Develop a template class `Vector` that represents a one-dimensional array and is parameterized on the type `T` of the elements in the vector. The class should use **`operator new`** and **`operator delete`** for memory allocation. Provide `iterator` and `const_iterator` types with the usual functionality along with and `begin` and `end` member functions.

8.39 Develop a template class `List` that represents a doubly-linked list and is parameterized on the type `T` of the elements in the list. For example, `List<int>` and `List<double>` would correspond to lists of **`int`**s and **`double`**s, respectively. The class should use **`operator new`** and **`operator delete`** for memory allocation. The `List` class should provide type members for mutating and nonmutating iterator types called `iterator` and `const_iterator`, respectively. The `List` class should support the basic operations associated with a list, including:

(a) create an empty list (via a default constructor);
(b) copy a list (via a constructor and assignment operator);
(c) move a list (via a constructor and assignment operator);
(d) query the number of elements in the list (`size` member function);
(e) get an iterator corresponding to the first element in the list (`begin` member function);
(f) get an iterator corresponding to the end position (i.e., after the last element) in the list (`end` member function);
(g) add a single element to the list (by copying) (`insert` member function);
(h) remove a single element from the list (`erase` member function);
(i) clear the list (i.e., delete all elements from the list) (`clear` member function);
(j) swap the contents of two lists (`swap` member function);
(k) splice a range of elements from one list to another (`splice` member function);

The iterator types (`List::iterator` and `List::const_iterator`) should provide the basic operations that one would expect of a bidirectional iterator type (e.g., copy, equality, inequality, increment, decrement, and dereference). The functionality provided by the `List` class should match the behavior of the corresponding functionality in the `std::list` class. For more information about the `std::list` class, see:

- http://en.cppreference.com/w/cpp/container/list

Note that the value returned by the `end` member function must not change as elements are added to, or removed from, the list. (Hint: This behavior can be guaranteed through the use of a dummy node, sometimes called a sentry node. The iterator returned by `end` is then associated with this dummy node.)

8.40 Develop a template class `Stack` that represents a stack and is parameterized on the type `T` of the elements in the stack. So, for example, `Stack<int>` would be a stack of **int**s. The class should use **operator new** and **operator delete** for memory allocation. The `Stack` class should provide the basic operations associated with a stack, including:

(a) default construct a stack;
(b) copy a stack;
(c) move a stack;
(d) test if the stack is empty (`empty`);
(e) query the number of elements on the stack (`size`);
(f) push an element on the stack (`push`); and
(g) pop an element off the stack if the stack is not empty and provide the popped value (`pop`).

Miscellany

8.41 In this exercise, a container class representing a collection of key-value pairs is developed. Each item in the class can be either marked or unmarked. The class allows (efficient) iteration over all items in the container as well as only those items that are marked. The class is called `Collection`. The items in the collection are of type `Item`. The `Collection` and `Item` classes must be placed in the namespace `ra`.

The `Item` class has the following public data members:

- `name`, a string of type `std::string` which holds the name of the item
- `value`, a **float** that holds some real value associated with the item
- any members needed to maintain links for the two linked lists into which an item may be inserted (each item can be potentially be inserted into multiple linked lists simultaneously)

No explicit boolean value can be used to track whether an items is marked. This must be determined by the state of the links for the relevant linked list. The `Item` class is neither copyable nor movable.

The `Collection` class is neither copyable nor movable. The `Collection` class provides the following public type members:

- `item_type`, which is an alias for the type used to represent items in the collection
- `all_iterator` and `all_const_iterator`, which are mutating and non-mutating iterator types used to iterate over all items in the container in the order of insertion
- `marked_iterator` and `all_marked_iterator`, which are mutating and non-mutating iterator types used to iterate over all marked items in the container

The `Collection` class provides the following public function members:

- a default constructor, which creates an empty collection
- a destructor
- `make_item`, a static member function (for convenience) that can be used to create a default-constructed `item_type` object on the heap; this function takes no parameters and returns a pointer to the created object
- `destroy_item` (for convenience) that can be used to destroy an object created with `make_item`; this function takes single parameter which is a reference to `item_type` object to be destroyed.
- `add_item` that allows a new item to be inserted in the collection; this function take a reference to an `item_type` object; no value is returned by the function
- `all_begin` and `all_end`, which are overloaded and return `all_iterator` and `all_const_iterator` type iterator, which allow iteration over all items in the collection
- `marked_begin` and `marked_end`, which are overloaded and return `marked_iterator` and `marked_const_iterator` type iterators, which allow iteration over marked items in the collection
- `is_marked` which tests if an item is marked; this function takes a single parameter that is a reference to a `item_type` object; the function has return type **bool**; if the item is marked, true is returned; otherwise, false is returned

- `mark`, which marks an item if it is not currently marked returning true if the marking was performed and false otherwise; this function takes a single parameter that is a reference to a `item_type` object (which, of course, must be in the collection)
- `unmark`, which unmarks an item if it is currently marked returning true if the unmarking was performed and false otherwise; this function takes a single parameter that is a reference to a `item_type` object (which, of course, must be in the collection)

The `Collection` type is neither copyable nor movable. Use the `boost::intrusive::list` type for intrusive linked lists. This minimal interface for the `Collection` class should allow the code in the listing below to function correctly.

```
#include <iostream>
#include "multilist_1.hpp"

int main() {

    ra::Collection c;

    // Read in name-value pairs and mark every other pair.
    std::string name;
    float value;
    std::size_t count = 0;
    while (std::cin >> name >> value) {
        ra::Collection::item_type* p = c.make_item();
        p->name = name;
        p->value = value;
        c.add_item(*p);
        if (!(count % 2)) {
            c.mark(*p);
        }
        ++count;
    }
    if (!std::cin.eof()) {
        std::cerr << "input error\n";
        return 1;
    }

    // Print all items in the collection.
    std::cout << "all items:\n";
    for (auto i = c.all_begin(); i != c.all_end(); ++i) {
        std::cout << i->name << ' ' << i->value << '\n';
    }

    // Print only the marked items in the collection.
    std::cout << "marked items:\n";
    for (auto i = c.marked_begin(); i != c.marked_end(); ++i) {
        std::cout << i->name << ' ' << i->value << '\n';
    }

    std::cout.flush();
    return std::cout ? 0 : 1;
}
```

The above program would take input resembling that shown below.

```
apple 1.0
banana 0.5
grape 3.14
```



```
orange 42
```

8.42 In this exercise, a class used to manage a product inventory will be developed. Each inventory item has the following attributes:

(a) product ID (which is a unique string)
(b) product name (which is a string that is not necessarily unique)
(c) quantity (which is an integer)
(d) unit cost (which is a real number)

An inventory item is to be represented with a class called `Item`. The `Item` class is neither movable nor copyable. The inventory is to be represented with a class called `Inventory`. This class is essentially a container for `Item` objects that maintains:

- a list of all of the inventory items in insertion order (i.e., the order that they were added to the inventory)
- a map with the product ID as key
- a multimap with the product name as key

The inventory-item information is not allowed to be duplicated in memory. Therefore, all containers employed should be intrusive. Containers from the Boost Intrusive library should be employed. The `Item` class should have only the following public data members:

- `id`
- `name`
- `quantity`
- `cost`
- any data members needed for intrusive containers

All of the above classes should be placed in the namespace `ra`. The `Inventory` class should provide the following public type members:

- `item_type`, which is an alias for the inventory item type (i.e., `Item`)
- `items_iterator` and `items_const_iterator`, which are used to iterate over items as a list (in insertion order)
- `items_by_id_iterator` and `items_by_id_const_iterator`, which are used to iterate over items sorted by product ID
- `items_iterator` and `items_const_iterator`, which are used to iterate over items sorted by product name

The `Inventory` class should provide the following public member functions:

- default constructor. This creates an empty inventory (i.e., an inventory with no items).
- destructor
- `add_item`. This function adds an item to the inventory. The return type is the iterator type `items_iterator`. If the addition is successful, an iterator referring to the newly added item is returned. Otherwise, the end iterator is returned.
- `items_begin`, `items_end`, `items_by_id_begin`, `items_by_id_end`, `items_by_name_begin`, `items_by_name_end`, which yield iterators for accessing the items in the container in various ways (e.g., as a list, as a map, as a multimap). Each of these functions should be overloaded to handle the case of const and non-const objects.
- `find_by_id`. This function finds the item with the specified ID (if one exists). This function takes a single parameter of `std::string`, which corresponds to the ID to be used for lookup. A pointer to an `Item` is returned. If an item is found, a pointer to it is returned. Otherwise, the null pointer is returned. This function is overloaded to handle the case of const and non-const objects.

The `Inventory` type is neither copyable nor movable. This minimal interface for the `Inventory` class should allow the code in the listing below to function correctly.


```
#include <iostream>
#include <string>
#include <vector>
#include "inventory_1_util.hpp"

int main(int argc, char** argv)
{
    ra::Inventory inventory;

    std::string id;
    std::string name;
    double cost;
    std::size_t quantity;
    while (std::cin >> id >> name >> quantity >> cost) {
       if (inventory.add_item(id, name, quantity, cost) ==
         inventory.items_end()) {
           std::cerr << "unable to add item to inventory\n";
           return 1;
       }
    }
    if (!std::cin.eof()) {
       std::cerr << "unable input error\n";
       return 1;
    }

    std::cout << "items in insertion order:\n";
    for (auto i = inventory.items_begin(); i != inventory.items_end(); ++i) {
       std::cout << i->id << ' ' << i->name << ' ' << i->quantity << ' ' <<
         i->cost << '\n';
    }
    std::cout << '\n';

    std::cout << "items sorted by ID:\n";
    for (auto i = inventory.items_by_id_begin();
      i != inventory.items_by_id_end(); ++i) {
       std::cout << i->id << ' ' << i->name << ' ' << i->quantity << ' ' <<
         i->cost << '\n';
    }
    std::cout << '\n';

    std::cout << "items sorted by name:\n";
    for (auto i = inventory.items_by_name_begin();
      i != inventory.items_by_name_end(); ++i) {
       std::cout << i->id << ' ' << i->name << ' ' << i->quantity << ' ' <<
         i->cost << '\n';
    }
    std::cout << '\n';

    std::cout << "results of find operations:\n";
    std::vector<std::string> ids{"apple-0000", "orange-1000"};
    for (auto&& id : ids) {
       ra::Inventory::item_type* item = inventory.find_by_id(id);
       if (item) {
           std::cout << "found " << item->id << ' ' << item->name << ' ' <<
             item->quantity << ' ' << item->cost << '\n';
       }
    }
```



```
58
59      std::cout.flush();
60      return std::cout ? 0 : 1;
61  }
```

The above program would take input resembling that shown below.

```
842 apple 100 0.25
843 apple 100 0.32
844 apple 100 0.50
100 zebra 100 0.50
000 grape 100 0.50
342 orange 100 0.50
343 orange 100 0.50
202 elephant 100 0.50
201 eel 100 0.50
042 aardvark 100 0.50
043 zulu 100 0.50
001 beta 100 0.50
apple-0000 apple 10 1000
grape-0001 grape 10 0.10
grape-0002 grape 10 0.10
```

8.43 Consider the code in the program below that invokes the `print_sorted_items` function in order to print the elements in a vector of `Item` objects in sorted order according to a specified sort key. Implement the function `print_sorted_items` subject to the constraint that it is not permitted to copy/move elements of type `Item`.

```
1   #include <vector>
2   #include <string>
3   #include <iostream>
4
5   // Inventory item
6   struct Item {
7       // product name
8       std::string name;
9       // per unit cost of item
10      double cost;
11      // quantity of item in stock
12      std::size_t quantity;
13  };
14
15  // Sorting criterion
16  enum Sort_key : int {
17      // ascending order alphabetically
18      name,
19      // ascending order by cost
20      increasing_cost,
21      // descending order by quantity
22      decreasing_quantity,
23  };
24
25  /*
26  This function prints the items in the specified inventory in sorted order,
27  where the items are sorted in ascending order by the specified key.
28  If successful, the function returns true; otherwise false is returned.
29  */
```



```
30 bool print_sorted_items(std::ostream& out, const std::vector<Item>& inventory,
31   enum Sort_key key);
32
33 int main(int argc, char** argv)
34 {
35     std::vector<Item> inventory;
36     std::string name;
37     double cost;
38     std::size_t quantity;
39     while (std::cin >> name >> cost >> quantity) {
40         inventory.emplace_back(Item{name, cost, quantity});
41     }
42     if (!std::cin.eof()) {
43         std::cerr << "input error\n";
44         return 1;
45     }
46     if (!print_sorted_items(std::cout, inventory, Sort_key::name)) {
47         std::cerr << "error\n";
48         return 1;
49     }
50     std::cout << '\n';
51     if (!print_sorted_items(std::cout, inventory, Sort_key::increasing_cost)) {
52         std::cerr << "error\n";
53         return 1;
54     }
55     std::cout << '\n';
56     if (!print_sorted_items(std::cout, inventory, Sort_key::decreasing_quantity)) {
57         std::cerr << "error\n";
58         return 1;
59     }
60     std::cout << '\n';
61     return std::cout.flush() ? 0 : 1;
62 }
```


Chapter 9

C Language

9.1 Exercises

C Compatibility

9.1 Give an example of a program that is valid C but not valid C++.

9.2 Give an example of a program whose source code is valid in both C and C++ but has differing semantics.

Appendix A

CGAL

A.1 Computational Geometry Algorithms Library (CGAL)

A.1.1 Reading

Before attempting any of the CGAL programming exercises, read the following from the most recent version of the CGAL Manual (https://doc.cgal.org/latest):

- *CGAL Manual Documentation* (https://doc.cgal.org/latest/Manual/index.html)
- In *Getting Started*:
  - *Hello World* (https://doc.cgal.org/latest/Manual/tutorial_hello_world.html)
  - *Organization of the Manual* (https://doc.cgal.org/latest/Manual/manual.html)
  - *Preliminaries* (https://doc.cgal.org/latest/Manual/preliminaries.html)
- In *Package Overview — Geometry Kernels*:
  - *2D and 3D Linear Geometry Kernel* (https://doc.cgal.org/latest/Kernel_23/index.html#Chapter_2D_and_3D_Geometry_Kernel); read the *User Manual* section and study at least a few of the code examples in the *Examples* section
- In *Package Overview — Triangulations and Delaunay Triangulations*:
  - *2D Triangulation* (https://doc.cgal.org/latest/Triangulation_2/index.html#Chapter_2D_Triangulation) read the *User Manual* section, study at least a few of the code examples in the *Examples* section, and look at the *Reference Manual* section for the part related to the types:
    - `Triangulation_hierarchy_2` (https://doc.cgal.org/latest/Triangulation_2/classCGAL_1_1Triangulation__hierarchy__2.html),
    - `Triangulation_2` (https://doc.cgal.org/latest/Triangulation_2/classCGAL_1_1Triangulation_2.html),
    - `Delaunay_triangulation_2` (https://doc.cgal.org/latest/Triangulation_2/classCGAL_1_1Delaunay__triangulation__2.html),
    - `Constrained_Delaunay_triangulation_2` (https://doc.cgal.org/latest/Triangulation_2/classCGAL_1_1Constrained__Delaunay__triangulation__2.html), and
    - the face/vertex/edge classes corresponding to the above triangulation types
      - `Triangulation_vertex_base_2` (https://doc.cgal.org/latest/Triangulation_2/classCGAL_1_1Triangulation__vertex__base__2.html)
      - `Triangulation_hierarchy_vertex_base_2` (https://doc.cgal.org/latest/Triangulation_2/classCGAL_1_1Triangulation__hierarchy__vertex__base__2.html)
      - `Triangulation_face_base_2` (https://doc.cgal.org/latest/Triangulation_2/classCGAL_1_1Triangulation__face__base__2.html)

  - `Constrained_triangulation_face_base_2` (https://doc.cgal.org/latest/Triangulation_2/classCGAL_1_1Constrained__triangulation__face__base__2.html)
- *2D Triangulation Data Structure* (https://doc.cgal.org/latest/TDS_2/index.html#Chapter_2D_Triangulation_Data_Structure); read the *User Manual* section

A.2 Exercises

In all of these exercises, you should comment your code so that someone who is unfamiliar with your code can understand what your code is doing without too much effort. Choosing good function/class/variable names will probably help to reduce the number of comments needed. The code that you write should be reasonably efficient (i.e., avoid gross inefficiency).

Geometry Kernels

A.1 (a) What is a (geometry) kernel?

(b) What is a construction? Give at least three examples of constructions.

(c) What is a predicate? Give at least three examples of predicates.

A.2 (a) What is the difference between an exact and inexact construction?

(b) What is the difference between an exact and inexact predicate?

(c) What are the advantages and disadvantages of exact constructions relative to inexact ones?

A.3 (a) What is the difference between an exact and inexact construction kernel? Give an example of each.

(b) What is the difference between an exact and inexact predicate kernel? Give an example of each.

Triangulations

A.4 The `Triangulation_hierarchy_2` class provides a similar interface as the other triangulation classes in CGAL (such as `Triangulation_2` and `Delaunay_triangulation_2`). What is the difference between a triangulation hierarchy and a normal (i.e., non-hierarchy) triangulation? For example, what is the difference between a `Triangulation_hierarchy_2<Triangulation_2>` and `Triangulation_2`? When would the former be preferred over the latter?

A.5 Triangulation convex-hull example program (`convex_hull`).

(a) From the directory `cgal/exercises` in the Git repository for this book, obtain the file `convex_hull.cpp`. Compile and run the code in the file `convex_hull.cpp` with the input dataset in the file `convex_hull.dat`. Record the output.

(b) The `convex_hull_1` program. Change the program in part (a) so that it can use either a basic triangulation (e.g., `Triangulation_2`) or Delaunay triangulation (e.g., `Delaunay_triangulation_2`) for the convex-hull computation. The new program should be called `convex_hull_1`. The command-line interface for the program should provide a `-d` option to select the type of triangulation to use. If the `-d` option is specified, a Delaunay triangulation should be used; otherwise, a basic triangulation should be employed. Your solution must **avoid unnecessary duplication of code**.

(c) The `convex_hull_2` program. Change the program in part (b) so that the selected triangulation type can optionally be used with a triangulation hierarchy (e.g., `Triangulation_hierarchy_2`). The new program should be called `convex_hull_2`. The command-line interface for the program should provide `-d` and `-h` options. If the `-d` option is specified, a Delaunay triangulation should be used; otherwise, a basic triangulation should be employed. If the `-h` option is specified, the selected triangulation type should be used with a triangulation hierarchy; otherwise, the selected triangulation type should be used without a triangulation hierarchy. The program must support all (four) possible combinations of either specifying or not specifying each of the `-d` and `-h` options. Your solution must **avoid unnecessary duplication of code**.

A.6 Triangulation statistics program (`triangulation_statistics`). Write a program that does the following:

(a) Read 2-D points from standard input until the end-of-file is reached. (The input consists of pairs of x and y coordinates separated by whitespace.)

(b) Compute the triangulation of the points (using the specified type of triangulation).

(c) Output the following in order:

i. the number of (finite) vertices in the triangulation

ii. the number of (finite) faces in the triangulation

iii. the minimum, maximum, and average valence of the (finite) vertices in the triangulation

iv. the minimum and maximum interior angle of the (finite) faces in the triangulation.

All angles should be output in units of degrees (not radians). The valence of a (finite) vertex is simply the number of (finite) edges incident on that vertex.

(d) Output the vertices of a (finite) face that contains the point (0,0) if any such face exists.

If the program is run with no options, a basic (i.e., `Triangulation_2`) triangulation should be used. If the program is run with the "`-d`" option, a Delaunay triangulation (i.e., `Delaunay_triangulation_2`) should be used. To parse the command-line options, use the `getopt` function. You must **avoid duplication of code** between the basic and Delaunay triangulation cases.

Record the output obtained with your program for the datasets in `triangulation_statistics_1.dat` and `triangulation_statistics_2.dat`. The files `triangulation_statistics_1.dat` and `triangulation_statistics_2.dat` are available from the Git repository for this book.

A.7 Triangulation output program (`triangulation_ouput`). Write a program that does the following:

(a) Read 2-D points from standard input until the end-of-file is reached.

(b) Compute a triangulation of the points (using the specified type of triangulation).

(c) Output the triangulation to standard output using the OFF format (which is described below).

If the program is run with no options, a basic (i.e., `Triangulation_2`) triangulation should be used. If the program is run with the "`-d`" option, a Delaunay triangulation (i.e., `Delaunay_triangulation_2`) should be used. To parse the command-line options, use the `getopt` function. You must **avoid duplication of code** between the basic and Delaunay triangulation cases. A `std::map` might be useful for keeping track of the correspondence between vertices and their indices when generating the OFF format data.

An example of a data file that could be used as input to the `triangulation_output` program is available from the Git repository for this book. See the file `triangulation_output_1.dat` in the directory `cgal/exercises`. For both the basic and Delaunay cases, run the program for the input dataset `triangulation_output_1.dat` and save the triangulation output to a file. Use a polygon mesh viewer that supports the OFF format (such as `meshlab` or `geomview`) to view the triangulation corresponding to the OFF file. Describe the difference in the appearance of the triangulations obtained in basic and Delaunay cases.

OFF FORMAT. The OFF format consists of the following items (in order): a header, vertex information, face information, and edge information. The header consists of the following whitespace-delimited fields (in order):

(a) The character string "OFF".

(b) The number of vertices.

(c) The number of faces.

(d) The number of edges, which may be specified as zero if no edge information is provided.

The vertex information has one line per vertex consisting of the following whitespace-delimited fields (in order):

(a) The x coordinate of the vertex.

(b) The y coordinate of the vertex.

(c) The z coordinate of the vertex, which can simply be chosen as zero for a triangulation in the xy-plane.

Each vertex is associated with an integer index, where the first vertex has index 0, the second vertex has index 1, and so on. The face information has one line per face consisting of the following whitespace-delimited fields (in order):

(a) The number of vertices in the face, which is always three for a triangulation.

(b) The indices of the face's vertices in CCW order.

The edge information can be omitted, by specifying the number of edges as zero in the header.

Rasterization

A.8 Triangle mesh rasterization program (`triangle_mesh_rasterize`). Rasterizing mesh models of images. In this exercise, a program will be developed that rasterizes a mesh model of an image. For the purposes of this exercise, a mesh model of an image is a representation of a function ϕ defined on $\Lambda \subset \mathbb{R}^2$ that is completely characterized by:

(a) a set $P = \{p_i\}$ of sample points, where $p_i = (x_i, y_i) \in \mathbb{Z}^2$; and

(b) the set $Z = \{z_i\}$ of the corresponding sample values (i.e., $z_i = \phi(p_i)$), where $z_i \in \mathbb{Z}$.

The set P is always chosen to include all of the extreme convex hull points of Λ so that Λ is the convex hull of P. This implies that any triangulation of P will completely cover Λ. From P and Z, the function ϕ is defined as follows. First, we form the Delaunay triangulation of P, which is ensured to be uniquely determined by a scheme such as symbolic perturbation (as is done in CGAL). Then, for each face of the triangulation with vertices (x_i, y_i), (x_j, y_j), and (x_k, y_k), we form the unique linear function that passes through the points (x_i, y_i, z_i), (x_j, y_j, z_j), and (x_k, y_k, z_k). By combining the functions obtained for all of the faces, we obtain the continuous piecewise-linear function ϕ.

Given the above definition of a mesh model, write a program that does the following:

(a) Read the mesh model of an image from standard input. The data is formatted as triplets of x, y, and z coordinates separated by whitespace. The parameters x_i, y_i, and z_i of the mesh model correspond to the x, y, and z coordinates read for the ith point in the data, respectively.

(b) Compute the Delaunay trianguation of the points $\{(x_i, y_i)\}$.

(c) Compute the bounding box B of the points $\{(x_i, y_i)\}$.

(d) Sample the function ϕ defined by the mesh model at each point in $\mathbb{Z}^2 \cap B$ in order to produce a digital image.

(e) Write the digital image to standard output in PNM format.

Your solution must be reasonably efficient. On the topic of efficiency, a few comments are in order. First, an orientation test (i.e., determining where a point lies with respect to a line) is relatively expensive. Second, point location (i.e., finding the face in a triangulation that contains a point) typically requires many orientation tests. Consequently, a solution to this exercise that performs many orientation tests (or point-location operations) will be grossly inefficient.

The PNM format consists of the following (in order):

(a) the characters "P2" followed by a newline character;
(b) the width of the image, followed by the height of the image, followed by a newline character;
(c) the maximum sample value (which should be 255), followed by a newline character;
(d) each of the sample values in raster-scan order (i.e., left-to-right, top-to-bottom).

Run the program for the input dataset `triangle_mesh_rasterize_1.dat` and save the image output to a file. The file `triangle_mesh_rasterize_1.dat` can be found in the directory `cgal/exercises` of the Git repository for this book. You can use the `xv` or `display` command on Linux to display the image file.

Appendix B

CMake

B.1 Exercises

Basics

B.1 `hello` Exercise. Write a CMakeLists file that can be used with CMake to build the `hello` program in the directory `cmake/exercises/hello` (in the Git repository for this book).

B.2 `basic` Exercise. Write a CMakeLists file that can be used with CMake to build the `sinc` and `unit_step` programs in the directory `cmake/exercises/basic` (in the Git repository for this book).

Assertions and Code Sanitizers

B.3 `assertions` Exercise.

(a) Write a CMakeLists file that can be used with CMake to build the `assert_false` program in the directory `cmake/exercises/assertions` (in the Git repository for this book). The CMakeLists file must provide an option called `ALWAYS_ENABLE_ASSERTIONS`, with a default value of `true`. When this option is enabled, your CMakeLists file must ensure that assertion checking is enabled, for the case of both release and debug builds. (Normally, assertion checking will be enabled for debug builds and disabled for release builds.) The disabling of assertion checking is accomplished by defining the preprocessor symbol `NDEBUG`, at compile time. So, enabling assertion checking is accomplished by ensuring that `NDEBUG` is not defined while compiling. Your solution need only handle the case of compilers that use a "`-D`" option to define a preprocessor symbol (such as GCC and Clang).

The recommended approach for this exercise is to use the "`REPLACE`" (or "`REGEX REPLACE`") operation of the `string` function to remove any occurance of "`-DNDEBUG`" from the value of each of the following variables: `CMAKE_CXX_FLAGS_DEBUG` and `CMAKE_CXX_FLAGS_RELEASE`. (To be more thorough, the `CMAKE_CXX_FLAGS_MINSIZEREL` and `CMAKE_CXX_FLAGS_RELWITHDEBINFO` variables should also be handled, but this is not required in this exercise.)

(b) A quick inspection of the source code for the `assert_false` program will confirm that this program should terminate with a failed assertion when run, if assertion checking is enabled. Using your CMakeLists file, build the `assert_false` program for each of a release and debug build. For each build, confirm that assertion checking is enabled by verifying that the `assert_false` program terminates with a failed assertion, as expected.

(c) Move the functionality of the `ALWAYS_ENABLE_ASSERTIONS` option into a CMake module named `EnableAssertions`. Then, use this module in your CMakeLists file. Confirm that your solution works correctly.

B.4 `sanitizers` Exercise.

(a) Write a CMakeLists file that can be used with CMake to build the `asan_fail` and `ubsan_fail` programs in the directory `cmake/exercises/sanitizers` (in the Git repository for this book). The CMakeLists file must provide two options:

i. `ENABLE_ASAN`. This option has a boolean value, which defaults to `false`, indicating if the Address Sanitizer (ASAN) should be enabled.

ii. `ENABLE_UBSAN`. This option has a boolean value, which defaults to `false`, indicating if the Undefined Behavior Sanitizer (UBSAN) should be enabled.

The recommended approach to this exercise is to modify `CMAKE_CXX_FLAGS` and `CMAKE_EXE_LINKER_FLAGS` as appropriate, depending on the values of `ENABLE_ASAN` and `ENABLE_UBSAN`. For the purposes of this exercise, you may assume the following:

- ASAN requires the use of the "`-fsanitize=address`" flag for both the compiler and linker.
- UBSAN requires the use of the "`-fsanitize=undefined`" flag for both the compiler and linker.

The above assumptions are valid for the GCC and Clang compilers.

(b) An inspection of the code will verify that the program `asan_fail` should trigger a failure from ASAN and the program `ubsan_fail` should trigger warnings/errors from UBSAN. Using your CMakeLists file, build the code with `ENABLE_ASAN` and `ENABLE_UBSAN` set to `true`. To verify that the build process works correctly, run the `asan_fail` and `ubsan_fail` programs and confirm that they behave as expected.

(c) Move the functionality of the `ENABLE_ASAN` and `ENABLE_UBSAN` options into a CMake module named `Sanitizers`. Then, use this module in your CMakeLists file. Confirm that your solution still works correctly.

Appendix C

Git

C.1 Exercises

C.1 In this exercise, you will be creating several repositories and performing some basic tasks with these repositories. To begin, select an empty directory for use in this exercise. This directory will be henceforth denoted `$TOP_DIR`. In this exercise, you need to perform the tasks listed below (in order). When performing these tasks you are not permitted to use the `git pull` command (i.e., pull operations must be decomposed into separate fetch and merge operations). Also, you should assume that the two users involved in this exercise do not know about the changes each other is making. This implies that a user must attempt a push operation (and have it fail) before they can know that a fetch operation is required. When a repository is first created and each time its commit history changes, you should draw the commit history of the repository (including local and remote-tracking branches and `HEAD`).

(a) Create an empty bare Git repository in the directory `$TOP_DIR/hello_remote.git` to be used as a remote in this exercise. This can be accomplished by typing:

```
cd $TOP_DIR
git init --bare hello_remote.git
```

(b) User 1. Clone the repository `$TOP_DIR/hello_remote.git` to the directory `$TOP_DIR/hello_1`. In the top-level directory of the repository `$TOP_DIR/hello_1`, create two files as follows:

i. a file `numbers.txt` with the following three lines of text:

```
4
5
6
```

ii. a file `words.txt` with the following six lines of text:

```
Alpha
Bravo
Charlie
Delta
Echo
Foxtrot
```

Propagate the changes to the remote repository `$TOP_DIR/hello_remote.git`. [Hint: You will need to use: `git add`, `git clone`, `git commit`, and `git push`.]

(c) User 2. Clone the repository `$TOP_DIR/hello_remote.git` to the directory `$TOP_DIR/hello_2`. In the repository `$TOP_DIR/hello_2`, modify the file `numbers.txt` by inserting the following three lines prior to the first line in the file:


```
1
2
3
```

Propagate the changes to the remote repository `$TOP_DIR/hello_remote.git`. [Hint: You will need to use: `git add`, `git clone`, `git commit`, and `git push`.]

(d) User 1. In the repository `$TOP_DIR/hello_1`, perform the following. Append the following lines to the file `numbers.txt`:

```
7
8
```

Append the following lines to the file `words.txt`:

```
Golf
Hotel
```

Propagate the changes to the remote repository `$TOP_DIR/hello_remote.git`. [Hint: You will need to use: `git add`, `git commit`, `git fetch`, `git merge`, and `git push`.]

(e) User 2. In the repository `$TOP_DIR/hello_2`, append the following line to the file `words.txt`:

```
Golf
```

If any merge conflicts arise, keep only the changes from user 1. Propagate the changes to the remote repository `$TOP_DIR/hello_remote.git`. [Hint: You will need to use: `git add`, `git commit`, `git fetch`, `git merge`, and `git push`.]

(f) Check that the repository `$TOP_DIR/hello_remote.git` now contains the desired contents. In particular, the repository should contain the following two files:

i. the file `numbers.txt` with the following contents:

```
1
2
3
4
5
6
7
8
```

ii. the file `words.txt` with the following contents:

```
Alpha
Bravo
Charlie
Delta
Echo
Foxtrot
Golf
Hotel
```

[Hint: You will need to use `git clone`.]

Appendix D

Video Lectures

D.1 Introduction

The author has prepared video lectures for some of the material covered in this book. All of the videos are hosted by YouTube and available through the author's YouTube channel:

- https://www.youtube.com/iamcanadian1867

The most up-to-date information about this video-lecture content can be found at:

- https://www.ece.uvic.ca/~mdadams/cppbook/#video_lectures

For the convenience of the reader, some information on the video-lecture content available at the time of this writing is provided in the remainder of this appendix.

D.2 2019-05 SENG 475 Video Lectures

The author prepared video lectures for all of the material covered in the 2019-05 offering of the course SENG 475 (titled "Advanced Programming Techniques for Robust Efficient Computing") in the Department of Electrical and Computer Engineering at the University of Victoria, Victoria, Canada. All of these videos are available from the author's YouTube channel. The video lectures for the above course can be found in the following YouTube playlist:

- https://www.youtube.com/playlist?list=PLbHYdvrWBMxazo1_B6vhW9gpd1wlkdeEW

An information package for the video lectures is available that includes:

- a copy of the edition of the lecture slides used in the video lectures (in PDF format);
- a copy of all of the supplemental handouts used in the video lectures (in PDF format); and
- a fully-cataloged list of the slides covered in the video lectures, where each slide in the list has a link to the corresponding time offset in the YouTube video where the slide is covered.

This information package is available from the video lecture section of the web site for the book:

- https://www.ece.uvic.ca/~mdadams/cppbook/#video_lectures

For the convenience of the reader, the catalog of the video lectures is also included in what follows.

D.2.1 Video-Lecture Catalog

To allow the content in the video lectures to be more easily located and navigated, a catalog of the video lectures is included below. This catalog contains a list of all slides covered in the lectures, where each slide in the list has a link to the corresponding time offset in the YouTube video where the slide is covered. By using this catalog, it is a trivial exercise to jump to the exact point in the video lectures where a specific slide/topic is covered (i.e., simply click on the appropriate hyperlink).

D.2.1.1 Lecture 1 (2019-05-07) — Course Introduction [2019-05-07]

The following is a link to the full video:

- https://youtu.be/-Jyf-U18_gI [duration: 00:48:37]

The following are links to particular offsets within the video:

- 00:00: [course_intro] SENG 475 & ECE 596C
- 00:24: [course_intro] Course Overview [multiple slides]
- 02:11: [course_intro] Prerequisites and Requirements
- 05:33: [course_intro] Course Topics
- 07:07: [course_intro] Learning Outcomes
- 09:42: [course_intro] Course Outline and Various Other Handouts
- 32:02: [course_intro] Video Lectures
- 32:37: [course_intro] Computer-Based Tutorial
- 37:10: [course_intro] Plagiarism and Other Forms of Academic Misconduct
- 41:54: [course_intro] Software Development Environment (SDE)
- 42:57: [course_intro] Prelude to SDE Demonstration
- 45:55: [course_intro] SDE Demonstration

D.2.1.2 Lecture 2 (2019-05-08) — Algorithms and Data Structures [2019-05-08]

The following is a link to the full video:

- https://youtu.be/JOUZZVLMJvI [duration: 00:49:42]

The following are links to particular offsets within the video:

- 00:00: [algorithms] Algorithms [title slide]
- 01:07: [algorithms] Software Performance
- 02:16: [algorithms] Random-Access Machine (RAM) Model
- 04:17: [algorithms] Worst-Case, Average, and Amortized Complexity
- 08:21: [algorithms] Asymptotic Analysis of Algorithms
- 09:55: [algorithms] Big Theta (Θ) Notation
  - [algorithms] Big Theta (Θ) Notation (Continued)
- 12:12: [algorithms] Big Oh (O) Notation
  - [algorithms] Big Oh (O) Notation (Continued)
- 13:01: [algorithms] Big Omega (Ω) Notation
  - [algorithms] Big Omega (Ω) Notation (Continued)
- 15:32: [algorithms] Asymptotic Notation in Equations and Inequalities
- 17:06: [algorithms] Properties of Θ, O, and Ω
- 18:30: [algorithms] Additional Remarks
- 18:49: [algorithms] Remarks on Asymptotic Complexity
- 22:30: [algorithms] Some Common Complexities
- 23:32: [algorithms] Recurrence Relations
- 25:12: [algorithms] Solving Recurrence Relations
- 26:24: [algorithms] Solutions for Some Common Recurrence Relations
- 27:39: [algorithms] Iterative Fibonacci Algorithm: Time Complexity
- 30:10: [algorithms] Iterative Fibonacci Algorithm: Space Complexity
- 31:04: [algorithms] Recursive Fibonacci Algorithm: Time Complexity

- 32:47: [algorithms] Recursive Fibonacci Algorithm: Space Complexity
- 34:34: [algorithms] Amdahl's Law
- 38:02: [data_structures] Abstract Data Types (ADTs)
- 41:14: [data_structures] Container ADTs
- 43:17: [data_structures] Container ADTs (Continued)
- 45:35: [data_structures] Iterator ADTs

D.2.1.3 Lecture 3 (2019-05-10) — Data Structures [2019-05-10]

The following is a link to the full video:

- https://youtu.be/1swLQCO-1Cg [duration: 00:46:23]

The following are links to particular offsets within the video:

- 00:00: [data_structures] Container and Iterator Considerations
- 03:26: [data_structures] Container and Iterator Considerations (Continued)
- 08:23: [data_structures] List ADT
- 10:43: [data_structures] Array-Based Lists
  - [data_structures] Array-Based Lists: Diagram
- 14:38: [data_structures] Remarks on Array-Based Lists
- 19:15: [data_structures] Singly-Linked Lists
  - [data_structures] Singly-Linked Lists: Code
  - [data_structures] Singly-Linked Lists: Diagram
- 29:52: [data_structures] Remarks on Singly-Linked Lists
- 33:19: [data_structures] Singly-Linked List With Header Node
  - [data_structures] Singly-Linked List With Header Node: Code
  - [data_structures] Singly-Linked List With Header Node: Diagram
- 40:52: [data_structures] Remarks on Singly-Linked List With Header Node
- 41:49: [data_structures] Doubly-Linked Lists
  - [data_structures] Doubly-Linked Lists: Code
  - [data_structures] Doubly-Linked Lists: Diagram
- 45:55: [data_structures] Remarks on Doubly-Linked Lists [starting from end of preceding slide]

D.2.1.4 Lecture 4 (2019-05-14) — Data Structures, Some C++ Review (Const and Other Stuff) [2019-05-14]

The following is a link to the full video:

- https://youtu.be/hSEUXnb0cFY [duration: 00:49:38]

The following are links to particular offsets within the video:

- 00:00: [data_structures] Doubly-Linked List With Sentinel Node
  - [data_structures] Doubly-Linked List With Sentinel Node: Code
  - [data_structures] Doubly-Linked List With Sentinel Node: Diagram
- 05:46: [data_structures] Remarks on Doubly-Linked Lists With Sentinel Node
- 07:23: [data_structures] Stack ADT
- 08:25: [data_structures] Array Implementation of Stack
  - [data_structures] Array Implementation of Stack: Diagram
- 09:13: [data_structures] Remarks on Array Implementation of Stack
- 10:52: [data_structures] Node-Based Implementation of Stack
  - [data_structures] Node-Based Implementation of Stack: Diagram
- 11:29: [data_structures] Remarks on Node-Based Implementation of Stack
- 13:28: [data_structures] Queue ADT
- 14:43: [data_structures] Array Implementation of Queue
- 16:32: [data_structures] Remarks on Array Implementation of Queue
- 17:40: [data_structures] Array of Arrays Implementation of Queue
  - [data_structures] Array of Arrays Implementation of Queue: Diagram

- 22:03: [data_structures] Remarks on Array of Arrays Implementation of Queue
- 22:22: [data_structures] Node-Based Implementation of Queue
  - [data_structures] Node-Based Implementation of Queue: Diagram
- 22:51: [data_structures] Remarks on Node-Based Implementation of Queue
- 23:02: [data_structures] Trees
- 24:11: [data_structures] Tree Terminology (Continued 1)
- 24:42: [data_structures] Tree Terminology (Continued 2)
- 25:20: [data_structures] Binary Trees
- 25:58: [data_structures] Perfect and Complete Trees
- 26:24: [data_structures] Balanced Binary Trees
- 27:25: [data_structures] Node-Based Binary Tree
  - [data_structures] Node-Based Binary Tree: Diagram
  - [data_structures] Remarks on Node-Based Binary Tree
- 29:11: [data_structures] Array-Based Binary Tree
- 29:49: [data_structures] Array-Based Binary Tree: Diagram
  - [data_structures] Remarks on Array-Based Binary Tree
- 31:19: [data_structures] Binary Search Trees
- 33:33: [data_structures] Heaps
- 34:34: [data_structures] Set and Multiset ADTs
- 36:20: [data_structures] Map and Multimap ADTs
  - [data_structures] Remarks on Implementation of Sets and Maps
- 38:04: [data_structures] Priority Queue ADT
- 41:01: [data_structures] Remarks on Priorty Queue Implementations
- 41:40: [basics] References Versus Pointers
- 45:15: [basics] The const Qualifier
- 45:34: [basics] The const Qualifier and Non-Pointer/Non-Reference Types

D.2.1.5 Lecture 5 (2019-05-15) — Some C++ Review (Const and Other Stuff) [2019-05-15]

The following is a link to the full video:

- https://youtu.be/1nDMJrwta24 [duration: 00:50:13]

The following are links to particular offsets within the video:

- 00:00: [basics] The const Qualifier and Non-Pointer/Non-Reference Types
- 01:27: [basics] The const Qualifier and Pointer Types
- 05:07: [basics] The const Qualifier and Reference Types
- 09:39: [basics] The constexpr Qualifier for Variables
- 16:08: [basics] The const Qualifier and Functions
- 20:43: [basics] String Length Example: Not Const Correct
- 20:53: [basics] Square Example: Not Const Correct
  - [basics] Square Example: Const Correct
- 25:51: [basics] Square Example: Const Correct
- 27:29: [basics] Function Types and the const Qualifier
- 32:30: [exercises] [Q.1] What is Wrong With This Code?
  - [exercises] [Q.1] Solution: Use Const Qualifier Correctly

D.2.1.6 Lecture 6 (2019-05-17) — Some C++ Review (Const and Other Stuff), Compile-Time Computation [2019-05-17]

The following is a link to the full video:

- https://youtu.be/KTT9boX3wyg [duration: 00:51:14]

The following are links to particular offsets within the video:

- 00:00: [exercises] [Q.2] What is Wrong With This Code?

    - ◇ [exercises] [Q.2] Solution: Use Const Qualifier Correctly
- ◇ 08:10: [exercises] [Q.3] What is Wrong With This Code?
    - ◇ [exercises] [Q.3] Solution: Functions Should Be Inline
- ◇ 16:17: [exercises] [Q.4] What is Wrong With This Code?
    - ◇ [exercises] [Q.4] Solution: Place Inline Function Definitions in Header File
- ◇ 19:22: [exercises] [Q.5] What is Wrong With This Code?
    - ◇ [exercises] [Q.5] Solution 1: Explicit Template Instantiation
    - ◇ [exercises] [Q.5] Solution 2: Define Function Template in Header File
- ◇ 27:07: [exercises] Remarks on Header Files and Function Declarations
- ◇ 32:33: [exercises] [Q.6] What is Wrong With This Code?
    - ◇ [exercises] [Q.6] Solution: Place Default Arguments in Header File
- ◇ 41:02: [basics] The constexpr Qualifier for Functions

D.2.1.7 Lecture 7 (2019-05-21) — Compile-Time Computation [2019-05-21]

The following is a link to the full video:

- ◇ https://youtu.be/GZWsV7KpAw8 [duration: 00:48:50]

The following are links to particular offsets within the video:

- ◇ 00:30: [basics] Constexpr Function Example: power_int (Iterative)
- ◇ 15:55: [basics] Compile-Time Versus Run-Time Computation
- ◇ 21:01: [classes] constexpr Member Functions
- ◇ 23:19: [classes] constexpr Constructors
- ◇ 24:49: [classes] Example: Constexpr Constructors and Member Functions
- ◇ 31:51: [classes] Why Constexpr Member Functions Are Not Implicitly Const
- ◇ 37:27: [classes] Literal Types
- ◇ 44:26: [classes] Example: Literal Types
- ◇ 46:48: [classes] Constexpr Variable Requirements

D.2.1.8 Lecture 8 (2019-05-22) — Compile-Time Computation, Temporary Objects [2019-05-22]

The following is a link to the full video:

- ◇ https://youtu.be/eULv_AiAFII [duration: 00:49:28]

The following are links to particular offsets within the video:

- ◇ 00:00: [classes] Example: Constexpr Variable Requirement Violations
- ◇ 02:03: [classes] Constexpr Function Requirements
- ◇ 06:22: [classes] Example: Constexpr Function Requirement Violations
- ◇ 10:50: [classes] Constexpr Constructor Requirements
- ◇ 12:42: [classes] Example: Constexpr Constructor Requirement Violations
- ◇ 15:16: [classes] Example: Constexpr and Accessing External State
- ◇ 18:15: [classes] Example: Constexpr and Immediate Initialization
- ◇ 21:55: [classes] Debugging Constexpr Functions
- ◇ 28:50: [classes] Example: Debugging Strategies for Constexpr Functions
- ◇ 30:55: [exercises] [Q.7] What is Wrong With This Code?
    - ◇ [exercises] [Q.7] Solution: Define Constexpr Function in Header
- ◇ 33:25: [exercises] [Q.8] What is Wrong With This Code?
    - ◇ [exercises] [Q.8] Answer: Invalid Constexpr Function
- ◇ 36:05: [exercises] [Q.9] What is Wrong With This Code?
    - ◇ [exercises] [Q.9] Solution: Initialize Constexpr Function Variables
- ◇ 40:48: [exercises] [Q.10] What is Wrong With This Code?
    - ◇ [exercises] [Q.10] Solution: Constexpr Requires Literal Types
- ◇ 42:16: [temporaries] Temporary Objects

D.2.1.9 Lecture 9 (2019-05-24) — Temporary Objects, Moving/Copying, Value Categories [2019-05-24]

The following is a link to the full video:
- https://youtu.be/LhCHHfMh4Gg [duration: 00:48:29]

The following are links to particular offsets within the video:
- 00:00: [temporaries] Temporary Objects
- 02:51: [temporaries] Temporary Objects (Continued)
- 06:51: [temporaries] Temporary Objects Example
- 07:54: [temporaries] Temporary Objects Example (Continued)
- 09:06: [temporaries] Prefix Versus Postfix Increment/Decrement
- 18:24: [rval_refs] Propagating Values: Copying and Moving
- 22:04: [rval_refs] Copying and Moving
- 23:50: [rval_refs] Buffer Example: Moving Versus Copying
- 25:09: [rval_refs] Buffer Example: Copying
- 27:49: [rval_refs] Buffer Example: Moving
- 30:55: [rval_refs] Moving Versus Copying
- 33:35: [lrvalues] Value Categories of Expressions
- 36:39: [lrvalues] Value Categories of Expressions (Continued)
- 40:36: [lrvalues] Lvalues
- 43:39: [lrvalues] Lvalues (Continued 1)

D.2.1.10 Lecture 10 (2019-05-28) — Value Categories, Moving/Copying [2019-05-28]

The following is a link to the full video:
- https://youtu.be/C1ONBX9-vdo [duration: 00:48:36]

The following are links to particular offsets within the video:
- 00:00: [lrvalues] Lvalues (Continued 2)
- 03:14: [lrvalues] Moving and Lvalues
- 07:17: [lrvalues] Rvalues
- 11:33: [lrvalues] Prvalues
- 14:11: [lrvalues] Prvalues (Continued)
- 19:38: [lrvalues] Xvalues
- 23:55: [lrvalues] Moving and Rvalues
- 34:43: [lrvalues] Moving and Lvalues/Rvalues
- 40:20: [lrvalues] Moving/Copying and Lvalues/Rvalues

D.2.1.11 Lecture 11 (2019-05-29) — Copy Elision [2019-05-29]

The following is a link to the full video:
- https://youtu.be/LCRKHycBhsQ [duration: 00:48:31]

The following are links to particular offsets within the video:
- 00:00: [copy_elision] Copy Elision and Implicit Moving [title slide]
- 00:36: [copy_elision] Copy Elision
- 06:55: [copy_elision] Copy Elision and Returning by Value
- 31:11: [copy_elision] Return-By-Value Example 1: Summary
- 35:32: [copy_elision] Return-By-Value Example 2: Summary
- 38:54: [copy_elision] Example Where Copy Elision Allowed But Likely Impossible
- 44:09: [copy_elision] Copy Elision and Passing by Value

D.2.1.12 Lecture 12 (2019-05-31) — Copy Elision, Implicit Move [2019-05-31]

The following is a link to the full video:
- https://youtu.be/QgfH-RFAFhI [duration: 00:50:32]

The following are links to particular offsets within the video:
- 00:00: [copy_elision] Pass-By-Value Example: Summary
- 04:11: [copy_elision] Copy Elision and Initialization
- 21:27: [copy_elision] Mandatory Copy Elision Example: Factory Function
- 25:02: [copy_elision] Return Statements and Moving/Copying
- 36:36: [copy_elision] Example: Return Statements and Moving/Copying
- 40:38: [copy_elision] Use of std::move in Return Statements
- 43:03: [copy_elision] Example: Moving/Copying, Copy Elision, and Implicit Move a.k.a. [exercises] [Q.MC1] Copy, Move, or Copy Elision?

D.2.1.13 Lecture 13 (2019-06-04) — Copy Elision, Implicit Move, Exceptions [2019-06-04]

The following is a link to the full video:
- https://youtu.be/yoA7fFfBRII [duration: 00:52:24]

The following are links to particular offsets within the video:
- 00:00: [exericses] [Q.MC1] Answer
- 09:44: [rval_refs] Allowing Move Semantics in Other Contexts via std::move
- 10:49: [rval_refs] Old-Style Swap
- 12:20: [rval_refs] Improved Swap
- 14:27: [rval_refs] Implication of Rvalue-Reference Type Function Parameters
- 17:34: [exceptions] Exceptions
- 18:52: [exceptions] The Problem
- 20:35: [exceptions] Traditional Error Handling
- 23:24: [exceptions] Example: Traditional Error Handling
- 25:09: [exceptions] Error Handling With Exceptions
- 27:55: [exceptions] Example: Exceptions
- 29:55: [exceptions] safe_divide Example: Traditional Error Handling
- 30:37: [exceptions] safe_divide Example: Exceptions
- 31:29: [exceptions] Exceptions Versus Traditional Error Handling
- 34:28: [exceptions] Exceptions
- 36:58: [exceptions] Standard Exception Classes
  - [exceptions] Standard Exception Classes (Continued 1)
  - [exceptions] Standard Exception Classes (Continued 2)
- 37:42: [exceptions] Throwing Exceptions
- 38:39: [exceptions] Throwing Exceptions (Continued)
- 40:45: [exceptions] Catching Exceptions
- 41:41: [exceptions] Catching Exceptions (Continued)
- 43:29: [exceptions] Rethrowing Exceptions
- 44:23: [exceptions] Transfer of Control from Throw Site to Handler
- 50:22: [exceptions] Stack Unwinding Example

D.2.1.14 Lecture 14 (2019-06-05) — Exceptions [2019-06-05]

The following is a link to the full video:
- https://youtu.be/_jyR6uel2k4 [duration: 00:47:00]

The following are links to particular offsets within the video:
- 00:00: [exceptions] Stack Unwinding Example
- 08:38: [exceptions] Function Try Blocks
- 09:49: [exceptions] Exceptions and Construction/Destruction
- 14:06: [exceptions] Construction/Destruction Example
- 18:09: [exceptions] Function Try Block Example
- 24:53: [exceptions] The noexcept Specifier

- 29:13: [exceptions] The noexcept Specifier (Continued 1)
  - [exceptions] The noexcept Specifier (Continued 2)
- 30:34: [exceptions] The noexcept Specifier (Continued 3)
- 37:33: [exceptions] Exceptions and Function Calls
- 42:06: [exceptions] Avoiding Exceptions Due to Function Calls

D.2.1.15 Lecture 15 (2019-06-07) — Exceptions, Interval Arithmetic [2019-06-07]

The following is a link to the full video:
- https://youtu.be/xMZl2vghJF4 [duration: 00:48:56]

The following are links to particular offsets within the video:
- 00:00: [exceptions] noexcept Operator
- 08:34: [exceptions] noexcept Operator (Continued)
- 17:00: [arithmetic] Interval Arithmetic
- 21:21: [arithmetic] Applications of Interval Arithmetic
- 24:11: [arithmetic] Real Interval Arithmetic
- 26:22: [arithmetic] Addition and Subtraction
- 27:54: [arithmetic] Multiplication and Division
- 28:46: [arithmetic] Floating-Point Interval Arithmetic
- 31:52: [arithmetic] Floating-Point Interval Arithmetic (Continued)
- 34:12: [arithmetic] Floating-Point Interval Arithmetic Operations
- 35:35: [arithmetic] Comparisons
- 44:18: [arithmetic] Setting and Querying Rounding Mode

D.2.1.16 Lecture 16 (2019-06-11) — Interval Arithmetic, Geometric Predicates and Applications [2019-06-11]

The following is a link to the full video:
- https://youtu.be/EcOOzgwRPw4 [duration: 00:46:42]

The following are links to particular offsets within the video:
- 00:00: [arithmetic] Impact of Current Rounding Mode
- 03:55: [arithmetic] Rounding Mode Example
- 04:53: [arithmetic] Geometric Predicates
- 07:18: [arithmetic] Filtered Geometric Predicates
- 11:44: [arithmetic] Two-Dimensional Orientation Test
- 13:50: [arithmetic] Example: Two-Dimensional Orientation Test
- 14:16: [arithmetic] Convex Polygons
- 17:08: [arithmetic] Polygon Convexity Test
- 20:42: [arithmetic] Three-Dimensional Orientation Test
- 25:58: [arithmetic] Side-of-Oriented-Circle Test
- 28:37: [arithmetic] Preferred-Direction Test
- 30:32: [arithmetic] Triangulations
- 33:40: [arithmetic] Delaunay Triangulations
- 35:37: [arithmetic] Nonuniqueness of Delaunay Triangulations
  - [arithmetic] Comments on Delaunay Triangulations
- 39:37: [arithmetic] Edge Flips
- 42:21: [arithmetic] Locally-Delaunay Test
- 45:49: [arithmetic] Locally Preferred-Directions Delaunay Test

D.2.1.17 Lecture 17 (2019-06-12) — Geometric Predicates and Applications, Memory Management [2019-06-12]

The following is a link to the full video:
- https://youtu.be/x3Z7Kxb32ew [duration: 00:41:34]

The following are links to particular offsets within the video:
- 00:00: [arithmetic] Locally Preferred-Directions Delaunay Test [plus related slides]
- 08:08: [arithmetic] Lawson Local Optimization Procedure
- 11:32: [arithmetic] Finding Delaunay Triangulations with Lawson LOP
- 13:43: [data_structures] Naive Triangle-Mesh Data Structure
- 16:04: [data_structures] Naive Triangle-Mesh Data Structure Example
- 20:11: [data_structures] Half-Edge Data Structure
- 20:46: [data_structures] Half-Edge Data Structure (Continued)
- 30:05: [data_structures] Object File Format (OFF)
- 30:40: [data_structures] OFF Example (Triangle Mesh)
- 34:01: [memory_management] Memory Management
- 36:18: [memory_management] Potential Problems Arising in Memory Management
- 38:42: [memory_management] Alignment
- 39:06: [memory_management] The alignof Operator

D.2.1.18 Lecture 18 (2019-06-14) — Memory Management [2019-06-14]

The following is a link to the full video:
- https://youtu.be/E31oR6H-Lv8 [duration: 00:41:56]

The following are links to particular offsets within the video:
- 00:09: [memory_management] The alignas Specifier
- 02:04: [memory_management] New Expressions
- 03:07: [memory_management] New Expressions (Continued)
- 05:49: [memory_management] Delete Expressions
- 07:22: [memory_management] Delete Expressions (Continued 1)
- 10:13: [memory_management] Delete Expressions (Continued 2)
- 11:58: [memory_management] Typical Strategy for Determining Array Size in Array Delete
- 19:21: [memory_management] New Expressions and Allocation
- 22:54: [memory_management] Allocation Function Overload Resolution
- 26:11: [memory_management] Allocation Function Overload Resolution (Continued)
- 29:03: [memory_management] New Expressions and Deallocation
- 30:37: [memory_management] Delete Expressions and Deallocation
- 31:04: [memory_management] Single-Object Operator New (i.e., operator new)
- 34:03: [memory_management] Single-Object Operator New Overloads
- 36:34: [memory_management] Single-Object Operator New Overloads (Continued)
- 37:28: [memory_management] Single-Object Operator New Examples

D.2.1.19 Lecture 19 (2019-06-18) — Memory Management [2019-06-18]

The following is a link to the full video:
- https://youtu.be/W_GazLV6qcg [duration: 00:48:04]

The following are links to particular offsets within the video:
- 00:00: [memory_management] Array Operator New (i.e., operator new[])
- 01:50: [memory_management] Array Operator New Overloads
- 02:57: [memory_management] Array Operator New Overloads (Continued)
- 03:31: [memory_management] Array Operator New Examples
- 11:54: [memory_management] Single-Object Operator Delete (i.e., operator delete)
- 13:44: [memory_management] Single-Object Operator Delete Overloads
- 14:16: [memory_management] Single-Object Operator Delete Examples
- 20:57: [memory_management] Array Operator Delete (i.e., operator delete[])
- 21:36: [memory_management] Array Operator Delete Overloads
- 21:42: [memory_management] Array Operator Delete Examples

- 22:14: [memory_management] Motivation for Placement New
  - [memory_management] Motivation for Placement New: Diagram
- 31:00: [memory_management] Placement New
- 36:59: [memory_management] Placement New Examples
- 43:24: [memory_management] Direct Destructor Invocation
- 46:15: [memory_management] Pseudodestructors

D.2.1.20 Lecture 20 (2019-06-19) — Memory Management [2019-06-19]

The following is a link to the full video:

- https://youtu.be/xKObs70kzC8 [duration: 00:49:07]

The following are links to particular offsets within the video:

- 00:00: [memory_management] std::addressof Function Template
- 02:29: [memory_management] std::addressof Example
- 04:25: [memory_management] The std::aligned_storage Class Template
- 05:48: [memory_management] Optional Value Example
- 07:17: [memory_management] Optional Value Example: Diagram
- 08:12: [memory_management] Optional Value Example: optval.hpp
- 19:57: [memory_management] Optional Value Example: User Code
- 22:10: [memory_management] Handling Uninitialized Storage
- 22:55: [memory_management] Functions for Uninitialized Storage
- 26:37: [memory_management] Functions for Uninitialized Storage (Continued)
- 27:47: [memory_management] Some Example Implementations
- 31:04: [memory_management] Bounded Array Example
- 31:19: [memory_management] Bounded Array Example: Diagram
- 32:46: [memory_management] Bounded Array Example: aligned_buffer.hpp
- 34:44: [memory_management] Bounded Array Example: array.hpp (1)
- 39:00: [memory_management] Bounded Array Example: array.hpp (2)
- 44:22: [memory_management] Bounded Array Example: array.hpp (3)
- 48:40: [memory_management] Bounded Array Example: array.hpp (4)

D.2.1.21 Lecture 21 (2019-06-21) — Memory Management, Intrusive Containers, Pointers to Members [2019-06-21]

The following is a link to the full video:

- https://youtu.be/Tlo0KliV-xY [duration: 00:49:10]

The following are links to particular offsets within the video:

- 00:00: [memory_management] Vector Example
- 01:48: [memory_management] Vector Example: Diagram
- 02:43: [memory_management] Vector Example: vec.hpp (1)
- 06:55: [memory_management] Vector Example: vec.hpp (2)
- 12:48: [memory_management] Vector Example: vec.hpp (3)
- 17:01: [memory_management] Vector Example: vec.hpp (4)
- 20:49: [memory_management] Vector Example: vec.hpp (5)
- 24:02: [memory_management] Vector Example: vec.hpp (6)
- 27:38: [data_structures] Intrusive Containers
- 33:25: [data_structures] Shortcomings of Non-Intrusive Containers
- 35:28: [data_structures] Advantages of Intrusive Containers
- 38:27: [data_structures] Disadvantages of Intrusive Containers
- 42:40: [data_structures] Disadvantages of Intrusive Containers (Continued)
- 45:21: [classes] Pointers to Members
- 47:58: [classes] Pointers to Members (Continued)

D.2.1.22 Lecture 22 (2019-06-25) — Pointers to Members, Intrusive Containers, Caches [2019-06-25]

The following is a link to the full video:

- https://youtu.be/3rCHYD5VE2U [duration: 00:52:44]

The following are links to particular offsets within the video:

- 00:00: [classes] Pointers to Members for Data Members
- 06:05: [classes] Pointers to Members Example: Accumulate
- 14:53: [data_structures] Intrusive Doubly-Linked List With Sentinel Node
  - [data_structures] Intrusive Doubly-Linked List With Sentinel Node: Code (Continued)
  - [data_structures] Intrusive Doubly-Linked List With Sentinel Node: Code
  - [data_structures] Intrusive Doubly-Linked List With Sentinel Node: Diagram
- 25:39: [data_structures] Remarks on Intrusive Doubly-Linked List With Sentinel Node
- 25:52: [data_structures] Examples of Intrusive Containers
- 27:03: [cache] The Memory Latency Problem
- 28:32: [cache] Principle of Locality
- 31:05: [cache] Memory Hierarchy
- 32:48: [cache] Caches
- 35:57: [cache] Memory and Cache
- 37:38: [cache] Block Placement
- 40:04: [cache] Block Placement (Continued)
- 42:35: [cache] Direct-Mapped Cache Example
- 43:31: [cache] K-Way Set-Associative Cache Example
- 44:28: [cache] Fully Associative Cache
- 45:03: [cache] Block Identificiation
- 46:43: [cache] Decomposition of Memory Address
- 48:53: [cache] Block Replacement
- 50:26: [cache] Write Policy

D.2.1.23 Lecture 23 (2019-06-26) — Caches, Cache-Efficient Algorithms [2019-06-26]

The following is a link to the full video:

- https://youtu.be/ZV3LOrsHuV0 [duration: 00:50:24]

The following are links to particular offsets within the video:

- 00:00: [cache] Cache Misses
- 02:14: [cache] Virtual Memory
- 03:20: [cache] Virtual Address Space
- 05:38: [cache] Address Translation
- 07:21: [supplemental] [Q.C2] Virtual Memory Exercise
- 08:39: [supplemental] [Q.C2] Virtual Memory Exercise (Continued)
- 14:03: [cache] Translation Lookaside Buffer (TLB)
- 15:59: [cache] Virtual and Physical Caches
- 17:28: [cache] Virtual Versus Physical Caches
- 19:37: [cache] Virtually-Indexed Physically-Tagged (VIPT) Caches
- 20:15: [cache] VIPT Cache Example
- 23:06: [cache] Cache Performance
- 23:50: [cache] Intel Core i7
- 24:42: [cache] ARM Cortex A8
- 25:43: [cache] Cache-Efficient Algorithms
- 26:56: [cache] Code Transformations to Improve Cache Efficiency
- 28:30: [data_structures] Row-Major Versus Column-Major Order
- 29:42: [cache] Array Merging Example
- 31:50: [cache] Loop Interchange Example
- 33:17: [cache] Loop Fusion Example

- 35:25: [cache] Blocking Example
- 37:20: [cache] Blocking Example (Continued 0.5)
- 40:54: [cache] Blocking Example (Continued 1)
- 42:11: [cache] Blocking Example (Continued 2)
- 44:48: [cache] Cache-Aware Versus Cache-Oblivious Algorithms
- 47:24: [cache] Tall Caches

D.2.1.24 Lecture 24 (2019-06-28) — Cache-Efficient Algorithms [2019-06-28]

The following is a link to the full video:

- https://youtu.be/BC-eOhw6kAQ [duration: 00:44:45]

The following are links to particular offsets within the video:

- 00:00: [cache] Idealized Cache Model
- 02:20: [cache] Remarks on Assumption of Optimal-Replacement Policy
- 03:45: [cache] Cache-Oblivious Algorithms
- 04:32: [cache] Scanning
- 09:44: [cache] Array Reversal
- 14:48: [cache] Naive Matrix Transposition
- 16:29: [cache] Naive Matrix Transposition: Performance
- 21:31: [cache] Cache-Oblivious Matrix Transposition
- 22:50: [cache] Cache-Oblivious Matrix Transposition (Continued)
- 24:47: [cache] Cache-Oblivious Matrix Transposition Example 1A [Part 1]
- 26:52: [handout] Transpose Algorithm Pseudocode
- 29:38: [handout] Matrix Subblock Characterization
- 30:57: [cache] Cache-Oblivious Matrix Transposition Example 1A [Part 2]
- 32:48: [cache] Cache-Oblivious Matrix Transposition Example 2
- 34:47: [cache] Cache-Oblivious Matrix Transposition: Performance
- 36:40: [cache] Naive Matrix Multiplication
- 39:07: [cache] Naive Matrix Multiplication: Performance

D.2.1.25 Lecture 25 (2019-07-03) — Cache-Efficient Algorithms, Concurrency [2019-07-03]

The following is a link to the full video:

- https://youtu.be/NTUnun-YjyQ [duration: 00:46:39]

The following are links to particular offsets within the video:

- 00:00: [cache] Cache-Oblivious Matrix Multiplication
- 02:16: [cache] Cache-Oblivious Matrix Multiplication (Continued 1)
- 05:55: [cache] Cache-Oblivious Matrix Multiplication (Continued 2)
- 06:44: [cache] Cache-Oblivious Matrix Multiplication Example 1
- 13:02: [cache] Cache-Oblivious Matrix Multiplication: Performance
- 15:14: [cache] Cache-Oblivious Matrix Multiplication Revisited
- 17:52: [cache] Cache-Oblivious Matrix Multiplication Revisited Example 2
- 20:48: [cache] Discrete Fourier Transform (DFT)
- 24:03: [cache] Cache-Oblivious Fast Fourier Transform (FFT)
- 29:41: [cache] Example: Four-Point DFT
- 32:15: [cache] Example: Four-Point DFT (Continued 1)
- 33:41: [cache] Example: Four-Point DFT (Continued 2)
- 34:01: [cache] Cache-Oblivious FFT: Performance
- 37:40: [concurrency] Processors
- 39:38: [concurrency] Processors (Continued)
- 41:29: [concurrency] Why Multicore Processors?
- 44:35: [concurrency] Concurrency

D.2.1.26 Lecture 26 (2019-07-05) — Concurrency [2019-07-05]

The following is a link to the full video:

- https://youtu.be/U__YDW14DA0 [duration: 00:47:06]

The following are links to particular offsets within the video:

- 00:00: [concurrency] Why Multithreading?
- 03:51: [concurrency] Memory Model
- 06:47: [concurrency] Sequential Consistency (SC)
- 09:36: [concurrency] Sequential-Consistency (SC) Memory Model
- 12:34: [concurrency] Load/Store Reordering Example: Single Thread
- 15:20: [concurrency] Load/Store Reordering Example: Multiple Threads
- 20:00: [concurrency] Atomicity of Memory Operations
- 21:46: [concurrency] Data Races
- 25:34: [concurrency] Torn Reads
- 28:57: [concurrency] Torn Writes
- 31:11: [concurrency] SC Data-Race Free (SC-DRF) Memory Model
- 34:36: [concurrency] C++ Memory Model
- 39:53: [concurrency] The std::thread Class
- 43:03: [concurrency] The std::thread Class (Continued)

D.2.1.27 Lecture 27 (2019-07-09) — Concurrency [2019-07-09]

The following is a link to the full video:

- https://youtu.be/1CkqUsDFPnE [duration: 00:45:55]

The following are links to particular offsets within the video:

- 00:00: [concurrency] std::thread Members
- 01:49: [concurrency] std::thread Members (Continued)
- 03:06: [concurrency] Example: Hello World With Threads [First Half]
- 05:15: [lambdas] Hello World Program Revisited
- 09:22: [lambdas] Linear-Function Functor Example
- 21:27: [concurrency] Example: Hello World With Threads [Second Hal]
- 23:00: [concurrency] Example: Thread-Function Argument Passing (Copy/Move Semantics)
- 25:23: [concurrency] Example: Thread-Function Argument Passing (Reference Semantics)
- 30:32: [concurrency] Example: Moving Threads
- 33:16: [concurrency] Example: Lifetime Bug
- 36:38: [concurrency] The std::thread Class and Exception Safety
- 38:21: [concurrency] The std::thread Class and Exception Safety (Continued)

D.2.1.28 Lecture 28 (2019-07-10) — Concurrency [2019-07-10]

The following is a link to the full video:

- https://youtu.be/U_hiEvfgf0Q [duration: 00:43:18]

The following are links to particular offsets within the video:

- 00:00: [concurrency] Happens-Before Relationships
- 03:12: [concurrency] “Earlier in Time” Versus Happens Before
- 09:02: [concurrency] Sequenced-Before Relationships
- 10:21: [concurrency] Sequenced-Before Relationships (Continued)
- 11:14: [concurrency] Inter-Thread Happens-Before Relationships
- 12:37: [concurrency] Summary of Happens-Before Relationships
- 13:15: [concurrency] Synchronizes-With Relationships
- 17:01: [concurrency] Examples of Synchronizes-With Relationships
- 17:50: [concurrency] Synchronizes-With Relationship: Thread Create and Join
- 23:19: [concurrency] Shared Data

- 24:50: [concurrency] Race Conditions
- 28:42: [concurrency] Critical Sections
- 30:43: [concurrency] Data-Race Example
- 32:33: [concurrency] Example: Data Race (Counter)
- 34:46: [concurrency] Example: Data Race and/or Race Condition (IntSet)

D.2.1.29 Lecture 29 (2019-07-12) — Concurrency [2019-07-12]

The following is a link to the full video:
- https://youtu.be/nHll640_vh0 [duration: 00:47:21]

The following are links to particular offsets within the video:
- 00:00: [concurrency] Mutexes
- 03:10: [concurrency] The std::mutex Class
- 05:44: [concurrency] std::mutex Members
- 08:02: [concurrency] Example: Avoiding Data Race Using Mutex (Counter) (mutex)
- 11:00: [concurrency] Synchronizes-With Relationships: Mutex Lock/Unlock
- 18:57: [concurrency] The std::scoped_lock Template Class
- 21:22: [concurrency] std::scoped_lock Members
- 22:14: [concurrency] Example: Avoiding Data Race Using Mutex (Counter) (scoped_lock)
- 24:22: [concurrency] Example: Avoiding Data Race Using Mutex (IntSet) (scoped_lock)
- 32:44: [concurrency] Acquisition of Multiple Locks
- 35:26: [concurrency] Example: Acquiring Two Locks for Swap (Incorrect)
- 38:56: [concurrency] Example: Acquiring Two Locks for Swap [scoped_lock]
- 39:20: [concurrency] The std::unique_lock Template Class
- 41:55: [concurrency] std::unique_lock Members
- 43:28: [concurrency] std::unique_lock Members (Continued)
- 43:55: [concurrency] Example: Avoiding Data Race Using Mutex (Counter) (unique_lock)

D.2.1.30 Lecture 30 (2019-07-16) — Concurrency [2019-07-16]

The following is a link to the full video:
- https://youtu.be/0LT1FMvkIoA [duration: 00:44:37]

The following are links to particular offsets within the video:
- 00:00: [concurrency] The std::lock Template Function
- 01:01: [concurrency] Example: Acquiring Two Locks for Swap [unique_lock and lock]
- 01:51: [concurrency] Static Local Variable Initialization and Thread Safety
- 03:16: [concurrency] Condition Variables
- 07:40: [concurrency] The std::condition_variable Class
- 13:26: [concurrency] std::condition_variable Members
- 14:30: [concurrency] std::condition_variable Members (Continued)
- 15:32: [concurrency] Example: Condition Variable (IntStack)
- 27:50: [concurrency] Latches
- 29:56: [concurrency] Latch Example: User Code
- 32:03: [concurrency] Latch Example: latch_1.hpp
- 37:15: [concurrency] The std::condition_variable_any Class
- 38:44: [concurrency] Thread Pools
- 42:07: [concurrency] Simple Thread Pool Interface Example

D.2.1.31 Lecture 31 (2019-07-17) — Concurrency, More Exceptions [2019-07-17]

The following is a link to the full video:
- https://youtu.be/DeLPO3S_cVM [duration: 00:45:53]

The following are links to particular offsets within the video:

- ◇ 00:00: [concurrency] Simple Thread Pool Interface Example
- ◇ 03:44: [exceptions] Resource Management
- ◇ 05:31: [exceptions] Resource Leak Example
- ◇ 07:17: [exceptions] Cleanup
- ◇ 08:43: [exceptions] Exception Safety and Exception Guarantees
- ◇ 13:13: [exceptions] Exception Guarantees
- ◇ 20:24: [exceptions] Resource Acquisition Is Initialization (RAII)
- ◇ 21:43: [exceptions] Resource Leak Example Revisited
- ◇ 30:25: [exceptions] RAII Example: Stream Formatting Flags
- ◇ 35:15: [exceptions] Other RAII Examples
- ◇ 37:55: [exceptions] Appropriateness of Using Exceptions
- ◇ 41:40: [exceptions] Enforcing Invariants: Exceptions Versus Assertions

D.2.1.32 Lecture 32 (2019-07-19) — Smart Pointers [2019-07-19]

The following is a link to the full video:

- ◇ https://youtu.be/_VV1BlJ97ug [duration: 00:42:43]

The following are links to particular offsets within the video:

- ◇ 00:00: [smart_ptrs] Memory Management, Ownership, and Raw Pointers
- ◇ 02:36: [smart_ptrs] Smart Pointers
- ◇ 05:15: [smart_ptrs] The std::unique_ptr Template Class
- ◇ 08:27: [smart_ptrs] The std::unique_ptr Template Class (Continued)
- ◇ 10:37: [handout] Move Operation for unique_ptr
- ◇ 13:17: [handout] Why unique_ptr Is Not Copyable
- ◇ 16:16: [smart_ptrs] std::unique_ptr Member Functions
- ◇ 17:41: [smart_ptrs] std::unique_ptr Member Functions (Continued)
- ◇ 18:13: [smart_ptrs] std::unique_ptr Example 1
- ◇ 21:48: [smart_ptrs] Temporary Heap-Allocated Objects
- ◇ 24:07: [smart_ptrs] Decoupled Has-A Relationship
- ◇ 28:19: [smart_ptrs] The std::shared_ptr Template Class
- ◇ 31:25: [smart_ptrs] The std::shared_ptr Template Class (Continued)
- ◇ 39:09: [smart_ptrs] std::shared_ptr Reference Counting Example
  - ◇ [smart_ptrs] std::shared_ptr Reference Counting Example (Continued 1)
  - ◇ [smart_ptrs] std::shared_ptr Reference Counting Example (Continued 2)

D.2.1.33 Lecture 33 (2019-07-23) — Smart Pointers, Vectorization [2019-07-23]

The following is a link to the full video:

- ◇ https://youtu.be/D_8Hfchp09A [duration: 00:48:07]

The following are links to particular offsets within the video:

- ◇ 00:00: [smart_ptrs] std::shared_ptr Member Functions
- ◇ 00:48: [smart_ptrs] std::shared_ptr Member Functions (Continued)
- ◇ 02:23: [smart_ptrs] Prefer Use of std::make_shared
- ◇ 04:08: [smart_ptrs] std::shared_ptr Example
- ◇ 12:31: [smart_ptrs] std::shared_ptr and const
- ◇ 13:46: [smart_ptrs] Factory Function
- ◇ 15:17: [smart_ptrs] Example: Shared Pointer to Subobject of Managed Object
- ◇ 18:04: [smart_ptrs] Example: Shared Pointer to Subobject of Managed Object (Continued 1)
- ◇ 20:51: [smart_ptrs] Example: Shared Pointer to Subobject of Managed Object (Continued 2)
- ◇ 24:35: [smart_ptrs] Example: Shared Pointer to Subobject of Managed Object (Continued 3)
- ◇ 25:17: [smart_ptrs] Example: Shared Pointer to Subobject of Managed Object (Continued 4)
- ◇ 27:36: [smart_ptrs] Example: std::shared_ptr

- ◇ 30:00: [smart_ptrs] Example: std::shared_ptr (Continued)
- ◇ 32:58: [vectorization] Vector Processing
- ◇ 34:33: [vectorization] Scalar Versus Vector Instructions
- ◇ 36:10: [vectorization] Vector-Memory and Vector-Register Architectures
- ◇ 38:13: [vectorization] Vector-Register Architectures
- ◇ 40:56: [vectorization] Vector Extensions
- ◇ 42:53: [vectorization] Intel x86/x86-64 Streaming SIMD Extensions (SSE)
- ◇ 44:18: [vectorization] Intel x86/x86-64 Advanced Vector Extensions (AVX)
- ◇ 46:09: [vectorization] ARM NEON

D.2.1.34 Lecture 34 (2019-07-24) — Vectorization [2019-07-24]

The following is a link to the full video:

- ◇ https://youtu.be/Thv9FA60XH8 [duration: 00:47:52]

The following are links to particular offsets within the video:

- ◇ 00:00: [vectorization] Checking for Processor Vector Support on Linux
- ◇ 01:06: [vectorization] Vectorization
- ◇ 03:14: [vectorization] Conceptualizing Loop Vectorization
- ◇ 06:56: [vectorization] Approaches to Vectorization
- ◇ 14:17: [vectorization] Auto-Vectorization
- ◇ 16:34: [vectorization] GCC Compiler and Vectorization
- ◇ 17:36: [vectorization] GCC Compiler Options Related to Vectorization
- ◇ 18:58: [vectorization] GCC Compiler Options Related to Vectorization (Continued)
- ◇ 21:09: [vectorization] Clang Compiler and Vectorization
- ◇ 21:39: [vectorization] Clang Compiler Options Related to Vectorization
- ◇ 22:58: [vectorization] Assessing Quality of Vectorized Code
- ◇ 24:48: [vectorization] Assessing Quality of Vectorized Code (Continued)
- ◇ 27:57: [vectorization] Auto-Vectorization with Hints
- ◇ 29:43: [vectorization] Obstacles to Vectorization
- ◇ 34:04: [vectorization] Data Dependencies and Vectorization
- ◇ 35:05: [vectorization] Flow Dependencies
- ◇ 37:38: [vectorization] Flow Dependence Example 1
- ◇ 40:34: [vectorization] Flow Dependence Example 1: Sequential Loop
- ◇ 41:54: [vectorization] Flow Dependence Example 1: Vectorized Loop
- ◇ 44:38: [vectorization] Flow Dependence Example 2
- ◇ 46:55: [vectorization] Output Dependencies

D.2.1.35 Lecture 35 (2019-07-26) — Vectorization [2019-07-26]

The following is a link to the full video:

- ◇ https://youtu.be/dIpS5ME6SKs [duration: 00:49:29]

The following are links to particular offsets within the video:

- ◇ 00:00: [vectorization] Control-Flow Dependencies and Vectorization
- ◇ 02:07: [vectorization] Aliasing
- ◇ 04:15: [vectorization] Aliasing and Optimization: An Example
- ◇ 06:18: [vectorization] Aliasing and Vectorization: An Example
- ◇ 12:29: [vectorization] The __restrict__ Keyword
- ◇ 19:13: [vectorization] Noncontiguous Memory Accesses
- ◇ 20:54: [vectorization] Data Alignment
- ◇ 24:57: [vectorization] Handling Misaligned Data
- ◇ 26:54: [handout] Example: Handling Misaligned Data
- ◇ 29:44: [vectorization] Controlling Alignment of Data

- ◇ 32:07: [vectorization] Informing Compiler of Data Alignment
- ◇ 35:56: [vectorization] Profitability of Vectorization
- ◇ 38:00: [vectorization] Vectorization Example: Version 1
- ◇ 40:12: [vectorization] Vectorization Example: Version 2
- ◇ 41:31: [vectorization] Vectorization Example: Version 3
- ◇ 45:33: [vectorization] Vectorization Example: Invoking add Function
- ◇ 47:02: [vectorization] Basic Requirements for Vectorizable Loops

D.2.1.36 Lecture 36 (2019-07-30) — Vectorization [2019-07-30]

The following is a link to the full video:

- ◇ https://youtu.be/gjnI4khPj5k [duration: 00:14:39]

The following are links to particular offsets within the video:

- ◇ 00:00: [vectorization] OpenMP SIMD Constructs
- ◇ 02:09: [vectorization] OpenMP simd Pragma
- ◇ 05:28: [vectorization] OpenMP declare simd Pragma
- ◇ 07:05: [vectorization] OpenMP SIMD-Related Pragma Clauses
- ◇ 08:29: [vectorization] OpenMP SIMD-Related Pragma Clauses (Continued)
- ◇ 08:50: [vectorization] Example: Vectorized Loop
- ◇ 12:34: [vectorization] Example: Vectorized Loop and Function

D.2.1.37 Lecture 37 (2019-07-31) — Final Course Wrap-Up [2019-07-31]

The following is a link to the full video:

- ◇ https://youtu.be/li216eCidB0 [duration: 00:30:16]

The following are links to particular offsets within the video:

- ◇ 00:00: [wrapup] Any Questions About the Final Exam?
- ◇ 14:31: [wrapup] Open Discussion on Ways to Improve Course
- ◇ 15:56: [wrapup] Lecture Slides and Videos
- ◇ 20:45: [wrapup] Course Experience Survey (CES)

D.2.1.38 Extra (2019-07-25) — Preliminary Information for Final Exam [2019-07-25]

The following is a link to the full video:

- ◇ https://youtu.be/HQx3F--UzYA [duration: 00:13:48]

The following are links to particular offsets within the video:

- ◇ 00:00: Final Exam Information

D.3 Rudimentary C++

The sections that follow have some information on video lectures that cover the basics of the C++ programming language and standard library.

D.3.1 Video-Lecture Catalog

To allow the content in the video lectures to be more easily located and navigated, a catalog of the video lectures is included below. This catalog contains a list of all slides covered in the lectures, where each slide in the list has a link to the corresponding time offset in the YouTube video where the slide is covered. By using this catalog, it is a trivial exercise to jump to the exact point in the video lectures where a specific slide/topic is covered (i.e., simply click on the appropriate hyperlink).

D.3.1.1 Getting Started — Compiling and Linking [2017-04-13]

The following is a link to the full video:

- https://youtu.be/w5s7XgnLHoo [duration: 00:13:19]

The following are links to particular offsets within the video:

- 00:00: [start] Title
- 00:16: [start] Section: Getting Started
- 00:42: [start] Section: Building Programs: Compiling and Linking
- 00:45: [start] hello Program: hello.cpp
- 02:52: [start] Software Build Process
- 03:26: [start] Software Build Process
- 05:10: [start] GNU Compiler Collection (GCC) C++ Compiler
- 05:54: [start] GNU Compiler Collection (GCC) C++ Compiler
- 11:28: [start] Manually Building hello Program

D.3.1.2 Version Control — Introduction [2017-04-06]

The following is a link to the full video:

- https://youtu.be/9s9_DLH1jaY [duration: 00:06:50]

The following are links to particular offsets within the video:

- 00:00: [vcs] Title
- 00:16: [vcs] Section: Version Control Systems
- 00:21: [vcs] Version Control Systems
- 01:46: [vcs] Centralized Version Control
- 03:38: [vcs] Distributed Version Control
- 04:34: [vcs] Pros and Cons of Distributed Version Control

D.3.1.3 Git — Introduction [2017-04-08]

The following is a link to the full video:

- https://youtu.be/scm2kxsX_Rk [duration: 00:17:25]

The following are links to particular offsets within the video:

- 00:00: [git] Title
- 00:16: [git] Section: Git
- 00:22: [git] Git
- 01:30: [git] Users of Git
- 01:55: [git] Repositories
- 02:42: [git] Revision History and Directed Acyclic Graphs
- 04:31: [git] Branching Workflows
- 05:48: [git] Local Picture
- 07:32: [git] Local and Remote Picture
- 09:21: [git] HEAD
- 10:46: [git] Remote-Tracking Branches
- 11:57: [git] Remote-Tracking Branches (Continued)
- 14:27: [git] Git Configuration
- 15:16: [git] Git on One Slide

D.3.1.4 Git — Demonstration [2017-04-05]

The following is a link to the full video:

- https://youtu.be/8VHC7vzWihw [duration: 00:13:10]

The following are links to particular offsets within the video:

- 00:00: [git] Title

- ◇ 00:16: [git] Section: Git Demonstration
- ◇ 00:21: [git] Demonstration

D.3.1.5 Build Systems — Introduction [2017-04-12]

The following is a link to the full video:

- ◇ https://youtu.be/FPcK_swg-f8 [duration: 00:03:25]

The following are links to particular offsets within the video:

- ◇ 00:00: [build] Title
- ◇ 00:16: [build] Section: Build Tools
- ◇ 00:25: [build] Build Tools
- ◇ 02:23: [build] Examples of Build Tools

D.3.1.6 Make — Introduction [2017-04-12]

The following is a link to the full video:

- ◇ https://youtu.be/FsGAM2pXP_Y [duration: 00:27:56]

The following are links to particular offsets within the video:

- ◇ 00:00: [make] Title
- ◇ 00:16: [make] Section: Make
- ◇ 00:20: [make] Make
- ◇ 02:34: [make] make Command
- ◇ 05:47: [make] Makefiles
- ◇ 07:18: [make] Makefiles (Continued 1)
- ◇ 09:46: [make] Makefiles (Continued 2)
- ◇ 11:43: [make] Makefile for hello Program
- ◇ 17:00: [make] Makefile for hello Program
- ◇ 17:49: [make] Makefile for hello Program
- ◇ 18:48: [make] Makefile for hello Program
- ◇ 26:47: [make] Commentary on Makefile for hello Program

D.3.1.7 CMake — Introduction [2017-04-16]

The following is a link to the full video:

- ◇ https://youtu.be/Ak6cGZshduY [duration: 00:22:13]

The following are links to particular offsets within the video:

- ◇ 00:00: [cmake] Title
- ◇ 00:16: [cmake] Section: CMake
- ◇ 00:21: [cmake] CMake
- ◇ 01:30: [cmake] Users of CMake
- ◇ 01:47: [cmake] Build Process
- ◇ 03:16: [cmake] Build Process (Diagram)
- ◇ 03:43: [cmake] CMake Basics
- ◇ 06:41: [cmake] In-Source Versus Out-of-Source Builds
- ◇ 08:37: [cmake] The cmake Command
- ◇ 09:41: [cmake] The cmake Command (Options)
- ◇ 10:37: [cmake] The cmake Command (Continued 1)
- ◇ 12:18: [cmake] The cmake Command for Building
- ◇ 13:28: [cmake] The cmake Command for Building (Command Usage)
- ◇ 14:59: [cmake] Hello World Example [Part 1]
- ◇ 17:19: [cmake] Hello World Example [Part 2]
- ◇ 17:56: [cmake] Hello World Demonstration [Part 1]
- ◇ 20:03: [cmake] Hello World Demonstration [Part 2]

D.3.1.8 CMake — Examples [2017-04-18]

The following is a link to the full video:
- https://youtu.be/cDWOECgupDg [duration: 00:27:43]

The following are links to particular offsets within the video:
- 00:00: [cmake] Title
- 00:16: [cmake] Section: Examples
- 00:21: [cmake] OpenGL/GLUT Example
- 00:44: [cmake] OpenGL/GLUT Example: Source Code
- 01:05: [cmake] OpenGL/GLUT Example: CMakeLists File
- 03:48: [cmake] OpenGL/GLUT Example: Demonstration
- 05:25: [cmake] CGAL Example
- 05:57: [cmake] CGAL Example: Source Code
- 06:20: [cmake] CGAL Example: CMakeLists File
- 08:43: [cmake] CGAL Example: Demonstration
- 10:25: [cmake] HG2G Example: Overview
- 11:09: [cmake] HG2G Example: Library Source Code
- 12:36: [cmake] HG2G Example: Application Source Code
- 13:13: [cmake] HG2G Example: CMakeLists Files
- 16:13: [cmake] HG2G Example: CMakeLists Files
- 17:24: [cmake] HG2G Example: CMakeLists Files (Continued 1)
- 21:01: [cmake] HG2G Example: Demonstration [Part 1]
- 25:46: [cmake] HG2G Example: Demonstration [Part 2]

D.3.1.9 Basics — Introduction [2015-04-06]

The following is a link to the full video:
- https://youtu.be/mP1wuVWKQmg [duration: 00:06:07]

The following are links to particular offsets within the video:
- 00:00: [basics] Title
- 00:17: [basics] Disclaimer
- 00:40: [basics] Section: C++ Basics
- 00:45: [basics] The C++ Programming Language
- 02:59: [basics] Comments
- 04:10: [basics] Identifiers
- 05:44: [basics] Reserved Keywords

D.3.1.10 Basics — Objects, Types, and Values [2015-04-08]

The following is a link to the full video:
- https://youtu.be/FlbBgg5IamY [duration: 01:09:06]

The following are links to particular offsets within the video:
- 00:00: [basics] Title
- 00:17: [basics] Section: Objects, Types, and Values
- 00:23: [basics] Fundamental Types
- 02:08: [basics] Fundamental Types (Continued)
- 04:21: [basics] Literals
- 05:02: [basics] Character Literals
- 06:34: [basics] Character Literals (Continued)
- 07:20: [basics] String Literals
- 09:11: [basics] Integer Literals
- 11:39: [basics] Integer Literals (Continued)
- 14:02: [basics] Floating-Point Literals

- ◇ 15:38: [basics] Boolean and Pointer Literals
- ◇ 16:24: [basics] Declarations and Definitions
- ◇ 18:25: [basics] Examples of Declarations and Definitions
- ◇ 20:18: [basics] Arrays
- ◇ 22:20: [basics] Array Example
- ◇ 24:23: [basics] Pointers
- ◇ 28:28: [basics] Pointer Example
- ◇ 32:22: [basics] References
- ◇ 36:28: [basics] References Example
- ◇ 39:29: [basics] Addresses, Pointers, and References
- ◇ 49:24: [basics] Type Aliases with typedef Keyword
- ◇ 50:47: [basics] Type Aliases with using Statement
- ◇ 52:40: [basics] The extern Keyword
- ◇ 54:21: [basics] The const Qualifier
- ◇ 01:03:46: [basics] The volatile Qualfier
- ◇ 01:06:26: [basics] The auto Keyword

D.3.1.11 Basics — Operators and Expressions [2016-03-20]

The following is a link to the full video:

- ◇ https://youtu.be/hwI4IHEUMZs [duration: 00:22:11]

The following are links to particular offsets within the video:

- ◇ 00:00: [basics] Title
- ◇ 00:16: [basics] Section: Operators and Expressions
- ◇ 00:25: [basics] Operators
- ◇ 01:00: [basics] Operators (Continued 1)
- ◇ 02:09: [basics] Operators (Continued 2)
- ◇ 02:49: [basics] Operators (Continued 3)
- ◇ 03:05: [basics] Operators (Continued 4)
- ◇ 03:22: [basics] Operator Precedence
- ◇ 04:47: [basics] Operator Precedence (Continued 1)
- ◇ 04:52: [basics] Operator Precedence (Continued 2)
- ◇ 05:55: [basics] Operator Precedence (Continued 3)
- ◇ 06:00: [basics] Operator Precedence (Continued 4)
- ◇ 06:30: [basics] Alternative Tokens
- ◇ 06:55: [basics] Expressions
- ◇ 09:56: [basics] Short-Circuit Evaluation
- ◇ 13:42: [basics] The sizeof Operator
- ◇ 15:08: [basics] The constexpr Qualifier for Variables
- ◇ 20:11: [basics] The static_assert Statement

D.3.1.12 Basics — Control-Flow Constructs [2015-04-09]

The following is a link to the full video:

- ◇ https://youtu.be/kEKy_TwNbEE [duration: 00:23:20]

The following are links to particular offsets within the video:

- ◇ 00:00: [basics] Title
- ◇ 00:17: [basics] Section: Control-Flow Constructs: Selection and Looping
- ◇ 00:29: [basics] The if Statement
- ◇ 01:24: [basics] The if Statement (Continued)
- ◇ 02:43: [basics] The if Statement: Example
- ◇ 03:51: [basics] The switch Statement

- 05:21: [basics] The switch Statement: Example
- 06:47: [basics] The while Statement
- 07:50: [basics] The while Statement: Example
- 09:02: [basics] The for Statement
- 11:01: [basics] The for Statement (Continued)
- 11:50: [basics] The for Statement: Example
- 13:45: [basics] The Range-Based for Statement
- 15:55: [basics] The do Statement
- 17:14: [basics] The do Statement: Example
- 18:05: [basics] The break Statement
- 19:37: [basics] The continue Statement
- 21:06: [basics] The goto Statement

D.3.1.13 Basics — Functions [2016-03-20]

The following is a link to the full video:

- https://youtu.be/NHS1726zvmE [duration: 01:00:57]

The following are links to particular offsets within the video:

- 00:00: [basics] Title
- 00:16: [basics] Section: Functions
- 00:24: [basics] Parameters and Arguments
- 02:08: [basics] Function Declarations and Definitions
- 03:52: [basics] Functions
- 06:15: [basics] Functions (Continued)
- 07:15: [basics] Function: Examples
- 08:28: [basics] The main Function
- 10:15: [basics] The main Function (Continued)
- 11:43: [basics] The main Function (Continued)
- 12:58: [basics] Lifetime
- 14:15: [basics] Pass-By-Value Versus Pass-By-Reference
- 16:00: [basics] Pass-By-Value Versus Pass-By-Reference
- 18:26: [basics] Pass By Value
- 20:43: [basics] Pass By Reference
- 22:51: [basics] Pass-By-Reference Example
- 26:19: [basics] Pass-By-Reference Example (Continued)
- 30:02: [basics] Inline Functions
- 32:13: [basics] Inlining of a Function
- 33:44: [basics] The constexpr Qualifier for Functions
- 37:01: [basics] Constexpr Function Example: power_int (Iterative)
- 39:53: [basics] Compile-Time Versus Run-Time Computation
- 42:59: [basics] Function Overloading
- 46:19: [basics] Default Arguments
- 48:13: [basics] Argument Matching
- 51:17: [basics] Argument Matching: Example
- 58:01: [basics] The assert Macro

D.3.1.14 Basics — Input/Output [2016-03-21]

The following is a link to the full video:

- https://youtu.be/MFSAl-ld2Bc [duration: 00:12:42]

The following are links to particular offsets within the video:

- 00:00: [basics] Title

- ◇ 00:16: [basics] Section: Input/Output (I/O)
- ◇ 00:22: [basics] Basic I/O
- ◇ 02:33: [basics] Basic I/O Example
- ◇ 04:55: [basics] I/O Manipulators
- ◇ 06:26: [basics] I/O Manipulators (Continued)
- ◇ 08:23: [basics] I/O Manipulators Example

D.3.1.15 Basics — Miscellany [2016-03-21]

The following is a link to the full video:
- ◇ https://youtu.be/IcPgHnmWy-8 [duration: 00:08:13]

The following are links to particular offsets within the video:
- ◇ 00:00: [basics] Title
- ◇ 00:16: [basics] Section: Miscellany
- ◇ 00:23: [basics] Namespaces
- ◇ 02:28: [basics] Namespaces: Example
- ◇ 05:11: [basics] Memory Allocation: new and delete

D.3.1.16 Classes — Introduction [2016-03-05]

The following is a link to the full video:
- ◇ https://youtu.be/8XIdrmAS4Aw [duration: 00:02:10]

The following are links to particular offsets within the video:
- ◇ 00:00: [classes] Title
- ◇ 00:16: [classes] Section: Classes
- ◇ 00:25: [classes] Classes

D.3.1.17 Classes — Members and Access Specifiers [2016-03-05]

The following is a link to the full video:
- ◇ https://youtu.be/ZdtqC6zASEI [duration: 00:35:14]

The following are links to particular offsets within the video:
- ◇ 00:00: [classes] Title
- ◇ 00:16: [classes] Section: Members and Access Specifiers
- ◇ 00:25: [classes] Class Members
- ◇ 01:47: [classes] Access Specifiers
- ◇ 03:04: [classes] Class Example
- ◇ 06:19: [classes] Default Member Access
- ◇ 06:58: [classes] The struct Keyword
- ◇ 07:54: [classes] Data Members
- ◇ 09:23: [classes] Function Members
- ◇ 12:59: [classes] The this Keyword
- ◇ 16:34: [classes] const Member Functions
- ◇ 25:39: [classes] Definition of Function Members in Class Body
- ◇ 27:31: [classes] Type Members
- ◇ 29:56: [classes] Friends
- ◇ 31:59: [classes] Class Example

D.3.1.18 Classes — Constructors and Destructors [2016-03-06]

The following is a link to the full video:
- ◇ https://youtu.be/9NVA6AGtccc [duration: 00:30:27]

The following are links to particular offsets within the video:

- 00:00: [classes] Title
- 00:16: [classes] Section: Constructors and Destructors
- 00:27: [classes] Propagating Values: Copying and Moving
- 01:42: [classes] Propagating Values: Copying Versus Moving
- 04:31: [classes] Constructors
- 06:21: [classes] Default Constructor
- 09:30: [classes] Copy Constructor
- 13:32: [classes] Move Constructor
- 16:56: [classes] Constructor Example
- 21:02: [classes] Initializer Lists
- 22:32: [classes] Initialize List Example
- 26:23: [classes] Destructors
- 28:07: [classes] Destructor Example

D.3.1.19 Classes — Operator Overloading [2016-03-09]

The following is a link to the full video:

- https://youtu.be/kpIJzSEIe4Y [duration: 00:41:05]

The following are links to particular offsets within the video:

- 00:00: [classes] Title
- 00:16: [classes] Section: Operator Overloading
- 00:24: [classes] Operator Overloading
- 01:32: [classes] Operator Overloading (Continued 1)
- 06:02: [classes] Operator Overloading (Continued 2)
- 07:59: [classes] Operator Overloading (Continued 3)
- 09:50: [classes] Operator Overloading Example: Vector
- 14:17: [classes] Operator Overloading Example: Array10
- 22:34: [classes] Operator Overloading: Member vs. Nonmember Functions
- 24:07: [classes] Operator Overloading: Member vs. Nonmember Functions
- 28:29: [classes] Copy Assignment Operator
- 31:55: [classes] Self-Assignment Example
- 34:09: [classes] Move Assignment Operator
- 36:20: [classes] Copy/Move Assignment Operator Example: Complex

D.3.1.20 Classes — More on Classes [2016-03-22]

The following is a link to the full video:

- https://youtu.be/esPOG-FQhdc [duration: 00:12:25]

The following are links to particular offsets within the video:

- 00:00: [classes] Title
- 00:16: [classes] Section: Miscellany
- 00:23: [classes] Explicitly Deleted/Defaulted Special Member Functions
- 03:07: [classes] Static Data Members
- 05:16: [classes] Static Member Functions
- 08:22: [classes] Stream Inserters
- 10:31: [classes] Stream Extractors

D.3.1.21 Classes — Temporary Objects [2016-03-24]

The following is a link to the full video:

- https://youtu.be/TT0TcIUo88E [duration: 00:14:01]

The following are links to particular offsets within the video:

- 00:00: [classes] Title

- ◇ 00:16: [classes] Section: Temporary Objects
- ◇ 00:27: [classes] Temporary Objects
- ◇ 04:10: [classes] Temporary Objects (Continued)
- ◇ 06:56: [classes] Temporary Objects Example
- ◇ 07:37: [classes] Temporary Objects Example (Continued)
- ◇ 10:08: [classes] Prefix Versus Postfix Increment/Decrement

D.3.1.22 Classes — Functors [2016-03-24]

The following is a link to the full video:
- ◇ https://youtu.be/qM2kvcSW4_E [duration: 00:08:14]

The following are links to particular offsets within the video:
- ◇ 00:00: [classes] Title
- ◇ 00:16: [classes] Section: Functors
- ◇ 00:22: [classes] Functors
- ◇ 01:50: [classes] Functor Example: Less Than
- ◇ 04:05: [classes] Functor Example With State

D.3.1.23 Templates — Introduction [2016-03-14]

The following is a link to the full video:
- ◇ https://youtu.be/q9Wx-kB7MRw [duration: 00:01:25]

The following are links to particular offsets within the video:
- ◇ 00:00: [templates] Title
- ◇ 00:16: [templates] Section: Templates
- ◇ 00:24: [templates] Templates

D.3.1.24 Templates — Function Templates [2016-03-17]

The following is a link to the full video:
- ◇ https://youtu.be/hD9MN9aVfi4 [duration: 00:25:49]

The following are links to particular offsets within the video:
- ◇ 00:00: [templates] Title
- ◇ 00:16: [templates] Section: Function Templates
- ◇ 00:30: [templates] Motivation for Function Templates
- ◇ 02:29: [templates] Function Templates
- ◇ 06:05: [templates] Function Templates (Continued)
- ◇ 10:47: [templates] Function Template Examples
- ◇ 13:14: [templates] Template Function Overload Resolution
- ◇ 18:29: [templates] Qualified Names
- ◇ 19:27: [templates] Dependent Names
- ◇ 20:29: [templates] Qualified Dependent Names
- ◇ 23:12: [templates] Why typename is Needed

D.3.1.25 Templates — Class Templates [2016-03-17]

The following is a link to the full video:
- ◇ https://youtu.be/NXUR5tfiHtE [duration: 00:18:56]

The following are links to particular offsets within the video:
- ◇ 00:00: [templates] Title
- ◇ 00:16: [templates] Section: Class Templates
- ◇ 00:30: [templates] Motivation for Class Templates
- ◇ 03:13: [templates] Class Templates

- ◇ 06:14: [templates] Class Templates (Continued)
- ◇ 10:36: [templates] Class Template Example
- ◇ 13:16: [templates] Class-Template Default Parameters
- ◇ 15:22: [templates] Qualified Dependent Names

D.3.1.26 Templates — Variable Templates [2016-03-14]

The following is a link to the full video:
- ◇ https://youtu.be/tb1e6t8uFGk [duration: 00:04:04]

The following are links to particular offsets within the video:
- ◇ 00:00: [templates] Title
- ◇ 00:16: [templates] Section: Variable Templates
- ◇ 00:26: [templates] Variable Templates
- ◇ 02:05: [templates] Variable Template Example: pi

D.3.1.27 Templates — Alias Templates [2016-03-14]

The following is a link to the full video:
- ◇ https://youtu.be/mzM0MHQIQcI [duration: 00:04:51]

The following are links to particular offsets within the video:
- ◇ 00:00: [templates] Title
- ◇ 00:16: [templates] Section: Alias Templates
- ◇ 00:26: [templates] Alias Templates
- ◇ 02:56: [templates] Alias Template Example

D.3.1.28 Standard Library — Introduction [2016-03-30]

The following is a link to the full video:
- ◇ https://youtu.be/-TY7_GniLig [duration: 00:10:16]

The following are links to particular offsets within the video:
- ◇ 00:00: [lib] Title
- ◇ 00:16: [lib] Section: C++ Standard Library
- ◇ 00:22: [lib] C++ Standard Library
- ◇ 00:57: [lib] C++ Standard Library (Continued)
- ◇ 03:11: [lib] Commonly-Used Header Files
- ◇ 04:24: [lib] Commonly-Used Header Files (Continued 1)
- ◇ 05:48: [lib] Commonly-Used Header Files (Continued 2)
- ◇ 07:14: [lib] Commonly-Used Header Files (Continued 3)
- ◇ 07:53: [lib] Commonly-Used Header Files (Continued 4)
- ◇ 09:19: [lib] Commonly-Used Header Files (Continued 5)

D.3.1.29 Standard Library — Containers, Iterators, and Algorithms [2016-04-05]

The following is a link to the full video:
- ◇ https://youtu.be/TxufBysSPK0 [duration: 00:38:57]

The following are links to particular offsets within the video:
- ◇ 00:00: [lib] Title
- ◇ 00:16: [lib] Section: Containers, Iterators, and Algorithms
- ◇ 00:22: [lib] Standard Template Library (STL)
- ◇ 01:05: [lib] Containers
- ◇ 02:13: [lib] Sequence Containers and Container Adapters
- ◇ 03:18: [lib] Associative Containers
- ◇ 04:50: [lib] Typical Container Member Functions

- 05:54: [lib] Container Example
- 07:36: [lib] Motivation for Iterators
- 09:47: [lib] Motivation for Iterators (Continued)
- 12:43: [lib] Iterators
- 15:03: [lib] Abilities of Iterator Categories
- 17:03: [lib] Input Iterators
- 17:45: [lib] Output Iterators
- 18:34: [lib] Forward Iterators
- 19:17: [lib] Bidirectional Iterators
- 19:36: [lib] Random-Access Iterators
- 21:17: [lib] Iterator Example
- 24:17: [lib] Iterator Gotchas
- 25:13: [lib] Algorithms
- 26:43: [lib] Functions
- 27:19: [lib] Functions (Continued 1)
- 28:03: [lib] Functions (Continued 2)
- 28:27: [lib] Functions (Continued 3)
- 29:13: [lib] Functions (Continued 4)
- 29:34: [lib] Functions (Continued 5)
- 29:42: [lib] Functions (Continued 6)
- 30:04: [lib] Functions (Continued 7)
- 30:37: [lib] Algorithms Example
- 33:40: [lib] Prelude to Functor Example
- 35:52: [lib] Functor Example

D.3.1.30 Standard Library — The vector Class Template [2016-03-30]

The following is a link to the full video:

- https://youtu.be/T8uaiYTIwjc [duration: 00:27:44]

The following are links to particular offsets within the video:

- 00:00: [lib] Title
- 00:16: [lib] Section: The vector Class Template
- 00:23: [lib] The vector Class Template
- 01:35: [lib] Member Types
- 04:08: [lib] Member Functions
- 05:36: [lib] Member Functions (Continued 1)
- 07:48: [lib] Member Functions (Continued 2)
- 09:14: [lib] Invalidation of References, Iterators, and Pointers
- 11:23: [lib] Iterator Invalidation Example
- 14:55: [lib] vector Example: Constructors
- 16:50: [lib] vector Example: Iterators
- 21:32: [lib] vector Example
- 24:40: [lib] vector Example: Emplace

D.3.1.31 Standard Library — The basic_string Class Template [2016-04-01]

The following is a link to the full video:

- https://youtu.be/J6POJIHactU [duration: 00:15:16]

The following are links to particular offsets within the video:

- 00:00: [lib] Title
- 00:16: [lib] Section: The basic_string Class Template
- 00:24: [lib] The basic_string Class Template

- 02:01: [lib] Member Types
- 04:47: [lib] Member Functions
- 05:51: [lib] Member Functions (Continued 1)
- 07:46: [lib] Member Functions (Continued 2)
- 08:47: [lib] Member Functions (Continued 3)
- 10:37: [lib] Member Functions (Continued 4)
- 10:51: [lib] Non-Member Functions
- 11:48: [lib] string Example
- 14:07: [lib] Numeric/String Conversion Example

D.3.1.32 Standard Library — Time Measurement [2016-04-02]

The following is a link to the full video:

- https://youtu.be/UeCiNGRZAyA [duration: 00:04:58]

The following are links to particular offsets within the video:

- 00:00: [lib] Title
- 00:16: [lib] Section: Time Measurement
- 00:24: [lib] Time Measurement
- 01:35: [lib] std::chrono Types
- 02:47: [lib] std::chrono Example: Measuring Elapsed Time

D.3.1.33 Concurrency — Preliminaries [2015-02-12]

The following is a link to the full video:

- https://youtu.be/oM1VxfrTQWg [duration: 01:01:50]

The following are links to particular offsets within the video:

- 00:00: [concurrency] Title
- 00:22: [concurrency] Disclaimer
- 00:32: [concurrency] Disclaimer
- 01:17: [concurrency] Section: Concurrency
- 01:41: [concurrency] Section: Preliminaries
- 02:08: [concurrency] Processors
- 03:58: [concurrency] Processors (Continued)
- 05:15: [concurrency] Memory Hierarchy
- 08:55: [concurrency] Examples of Multicore Processors
- 09:47: [concurrency] Examples of Multicore SoCs
- 10:57: [concurrency] Why Multicore Processors?
- 13:28: [concurrency] Section: Multithreaded Programming
- 13:35: [concurrency] Concurrency
- 17:58: [concurrency] Memory Model
- 21:00: [concurrency] Sequential Consistency (SC)
- 24:15: [concurrency] Sequential-Consistency (SC) Memory Model
- 27:51: [concurrency] Load/Store Reordering Example: Single Thread
- 30:28: [concurrency] Load/Store Reordering Example: Multiple Threads
- 35:10: [concurrency] Store-Buffer Example: Store Buffer
- 36:54: [concurrency] Store-Buffer Example: Without Store Buffer
- 40:18: [concurrency] Store-Buffer Example: With Store Buffer (Not SC)
- 46:21: [concurrency] Atomicity of Memory Operations
- 48:19: [concurrency] Data Races
- 51:40: [concurrency] Torn Reads
- 54:29: [concurrency] Torn Writes
- 56:53: [concurrency] SC Data-Race Free (SC-DRF) Memory Model

- ◇ 01:00:31: [concurrency] C++ Memory Model

D.3.1.34 Concurrency — Threads [2015-02-17]

The following is a link to the full video:

- ◇ https://youtu.be/fqG8BgVbmcM [duration: 00:33:45]

The following are links to particular offsets within the video:

- ◇ 00:00: [concurrency] Title
- ◇ 00:22: [concurrency] Disclaimer
- ◇ 00:32: [concurrency] Disclaimer
- ◇ 01:17: [concurrency] Section: Thread Management
- ◇ 01:40: [concurrency] The std::thread Class
- ◇ 04:38: [concurrency] The std::thread Class (Continued)
- ◇ 07:14: [concurrency] std::thread Members
- ◇ 07:52: [concurrency] std::thread Members (Continued)
- ◇ 09:52: [concurrency] Example: Hello World With Threads
- ◇ 12:24: [concurrency] Example: Thread-Function Argument Passing (Copy Semantics)
- ◇ 14:49: [concurrency] Example: Thread-Function Argument Passing (Reference Semantics)
- ◇ 17:58: [concurrency] Example: Thread-Function Argument Passing (Move Semantics)
- ◇ 18:43: [concurrency] Example: Moving Threads
- ◇ 21:46: [concurrency] The std::this_thread Namespace
- ◇ 23:12: [concurrency] Example: Identifying Threads
- ◇ 25:38: [concurrency] Example: Lifetime Bug
- ◇ 28:47: [concurrency] Thread Local Storage
- ◇ 30:47: [concurrency] Example: Thread Local Storage

D.3.1.35 Concurrency — Mutexes [2015-02-23]

The following is a link to the full video:

- ◇ https://youtu.be/Vm-Jsno58Y0 [duration: 01:25:24]

The following are links to particular offsets within the video:

- ◇ 00:00: [concurrency] Title
- ◇ 00:22: [concurrency] Disclaimer
- ◇ 00:32: [concurrency] Disclaimer
- ◇ 01:17: [concurrency] Section: Sharing Data Between Threads
- ◇ 01:30: [concurrency] Shared Data
- ◇ 02:46: [concurrency] Race Conditions
- ◇ 09:01: [concurrency] Critical Sections
- ◇ 11:02: [concurrency] Data-Race Example
- ◇ 12:46: [concurrency] Example: Data Race (Counter)
- ◇ 14:09: [concurrency] Example: Data Race and/or Race Condition (IntSet)
- ◇ 17:49: [concurrency] Section: Mutexes
- ◇ 18:16: [concurrency] Mutexes
- ◇ 22:01: [concurrency] The std::mutex Class
- ◇ 23:32: [concurrency] std::mutex Members
- ◇ 24:39: [concurrency] Example: Avoiding Data Race Using Mutex (Counter) (mutex)
- ◇ 27:38: [concurrency] Example: Avoiding Data Race Using Mutex (Counter) (mutex)
- ◇ 28:54: [concurrency] The std::lock_guard Template Class
- ◇ 32:15: [concurrency] std::lock_guard Members
- ◇ 32:54: [concurrency] Example: Avoiding Data Race Using Mutex (Counter) (lock_guard)
- ◇ 36:47: [concurrency] Example: Avoiding Data Race Using Mutex (IntSet) (lock_guard)
- ◇ 42:05: [concurrency] The std::unique_lock Template Class

- 45:11: [concurrency] std::unique_lock Members
- 46:48: [concurrency] Example: Avoiding Data Race Using Mutex (Counter) (unique_lock)
- 49:36: [concurrency] The std::lock Template Function
- 50:07: [concurrency] Example: Acquiring Two Locks for Swap (Incorrect)
- 55:51: [concurrency] Example: Acquiring Two Locks for Swap
- 58:32: [concurrency] The std::timed_mutex Class
- 59:42: [concurrency] Example: Acquiring Mutex With Timeout (std::timed_mutex)
- 01:03:10: [concurrency] Recursive Mutexes
- 01:06:08: [concurrency] Recursive Mutex Classes
- 01:07:13: [concurrency] Shared Mutexes
- 01:09:49: [concurrency] The std::shared_timed_mutex Class
- 01:10:04: [concurrency] std::shared_timed_mutex Members
- 01:11:53: [concurrency] The std::shared_lock Template Class
- 01:13:29: [concurrency] Example: std::shared_timed_mutex
- 01:18:18: [concurrency] std::once_flag and std::call_once
- 01:19:12: [concurrency] Example: One-Time Action
- 01:20:57: [concurrency] Example: One-Time Initialization
- 01:23:39: [concurrency] Static Local Variable Initialization and Thread Safety

D.3.1.36 Concurrency — Condition Variables [2015-02-27]

The following is a link to the full video:

- https://youtu.be/Wsk56vrKOng [duration: 00:17:37]

The following are links to particular offsets within the video:

- 00:00: [concurrency] Title
- 00:22: [concurrency] Disclaimer
- 00:32: [concurrency] Disclaimer
- 01:17: [concurrency] Section: Condition Variables
- 01:43: [concurrency] Condition Variables
- 05:27: [concurrency] The std::condition_variable Class
- 08:43: [concurrency] std::condition_variable Members
- 09:14: [concurrency] std::condition_variable Members (Continued)
- 10:12: [concurrency] Example: Condition Variable (IntStack)
- 16:38: [concurrency] The std::condition_variable_any Class

D.3.1.37 Concurrency — Promises and Futures [2015-04-02]

The following is a link to the full video:

- https://youtu.be/hic5W_UMjqU [duration: 00:47:45]

The following are links to particular offsets within the video:

- 00:00: [concurrency] Title
- 00:22: [concurrency] Disclaimer
- 00:32: [concurrency] Disclaimer
- 01:17: [concurrency] Section: Promises and Futures
- 01:25: [concurrency] Promises and Futures
- 04:35: [concurrency] Promises and Futures (Continued)
- 06:28: [concurrency] The std::promise Template Class
- 09:39: [concurrency] std::promise Members
- 10:03: [concurrency] std::promise Members (Continued)
- 11:44: [concurrency] The std::future Template Class
- 14:05: [concurrency] std::future Members
- 16:00: [concurrency] Example: Promises and Futures (Without std::async)

- 20:26: [concurrency] The std::shared_future Template Class
- 21:54: [concurrency] Example: std::shared_future
- 25:11: [concurrency] The std::async Template Function
- 28:44: [concurrency] The std::async Template Function (Continued)
- 29:54: [concurrency] Example: Promises and Futures (With std::async)
- 34:11: [concurrency] Example: Futures and Exceptions
- 37:50: [concurrency] The std::packaged_task Template Class
- 39:40: [concurrency] std::packaged_task Members
- 41:09: [concurrency] Example: Packaged Task
- 44:01: [concurrency] Example: Packaged Task With Arguments

D.3.1.38 CGAL — Introduction [2015-06-29]

The following is a link to the full video:

- https://youtu.be/Mk-NH2-_hMo [duration: 00:08:04]

The following are links to particular offsets within the video:

- 00:00: [cgal] Title
- 00:24: [cgal] Section: Computational Geometry Algorithms Library (CGAL)
- 00:33: [cgal] Computational Geometry Algorithms Library (CGAL)
- 02:29: [cgal] CGAL (Continued)
- 03:35: [cgal] Handles
- 04:27: [cgal] Linear Sequences Versus Circular Sequences
- 06:14: [cgal] Circulators

D.3.1.39 CGAL — Polygon Meshes [2015-07-02]

The following is a link to the full video:

- https://youtu.be/R8hlJCR4xO0 [duration: 00:33:54]

The following are links to particular offsets within the video:

- 00:00: [cgal] Title
- 00:24: [cgal] Section: Polygon Meshes
- 00:30: [cgal] Polyhedron_3 Class
- 02:53: [cgal] Polyhedron_3 Type Members
- 04:27: [cgal] Polyhedron_3 Type Members (Continued 1)
- 06:13: [cgal] Polyhedron_3 Type Members (Continued 2)
- 07:34: [cgal] Polyhedron_3 Function Members
- 09:40: [cgal] Polyhedron_3 Function Members (Continued 1)
- 10:31: [cgal] Polyhedron_3 Function Members (Continued 2)
- 12:51: [cgal] Polyhedron_3::Facet
- 14:43: [cgal] Facet Function Members
- 16:11: [cgal] Polyhedron_3::Vertex
- 17:54: [cgal] Vertex Function Members
- 19:21: [cgal] Polyhedron_3::Halfedge
- 21:04: [cgal] Polyhedron_3::Halfedge (Continued)
- 22:36: [cgal] Halfedge Function Members
- 23:52: [cgal] Halfedge Function Members (Continued 1)
- 25:32: [cgal] Halfedge Function Members (Continued 2)
- 26:25: [cgal] Adjacency Example
- 31:12: [cgal] Polyhedron_3 I/O
- 31:37: [cgal] Polyhedron_3 Gotchas

D.3.1.40 CGAL — Subdivision Surface Methods [2015-06-29]

The following is a link to the full video:
- ◇ https://youtu.be/t_zvp9dTTBY [duration: 00:03:23]

The following are links to particular offsets within the video:
- ◇ 00:00: [cgal] Title
- ◇ 00:24: [cgal] Section: Surface Subdivision Methods
- ◇ 00:29: [cgal] Subdivision Methods
- ◇ 01:23: [cgal] Subdivision Functions

D.3.1.41 CGAL — Example Programs [2015-07-01]

The following is a link to the full video:
- ◇ https://youtu.be/sTfG7tStFvI [duration: 00:34:03]

The following are links to particular offsets within the video:
- ◇ 00:00: [cgal] Title
- ◇ 00:24: [cgal] Section: Example Programs
- ◇ 00:31: [cgal] Mesh Generation Program: makeMesh
- ◇ 00:55: [cgal] meshMake Source-Code Walkthrough
- ◇ 09:40: [cgal] Mesh Information Program: meshInfo
- ◇ 10:20: [cgal] meshInfo Source-Code Walkthrough
- ◇ 25:36: [cgal] Mesh Subdivision Program: meshSubdivide
- ◇ 26:12: [cgal] meshSubdivide Source-Code Walkthrough

D.3.1.42 Text Formatting in C++20 [2021-02-03]

The following is a link to the full video:
- ◇ https://youtu.be/E8456triH_g [duration: 00:55:23]

The following are links to particular offsets within the video:
- ◇ 00:00: [format] Text Formatting in C++20
- ◇ 01:02: [format] Text Formatting
- ◇ 02:27: [format] Motivating Example 1: sprintf Family Functions
- ◇ 05:58: [format] Motivating Example 2: I/O Streams
- ◇ 09:52: [format] Text Formatting and std::format Family Functions
- ◇ 12:07: [format] std::format Family of Functions
- ◇ 13:46: [format] Format Strings
- ◇ 17:31: [format] Format String (Continued)
- ◇ 21:09: [format] Format Specifiers
- ◇ 22:48: [format] Type Options for Integer Types
- ◇ 25:07: [format] Type Options for Character Types
- ◇ 26:12: [format] Type Options for Boolean Types
- ◇ 27:14: [format] Type Options for Floating-Point Types
- ◇ 30:03: [format] Type Options for String Types
- ◇ 30:17: [format] Sign Options
- ◇ 32:13: [format] Field Width and Precision Options
- ◇ 36:36: [format] Fill Characters and Alignment Options
- ◇ 38:08: [format] Locale-Specific Formatting
- ◇ 39:12: [format] Locale-Specific Formatting Example
- ◇ 41:06: [format] Example: Formatting to a Buffer [format_to, format_to_n, formatted_size]
- ◇ 46:21: [format] Formatting User-Defined Types
- ◇ 48:11: [format] Point Formatter Example: custom_1.hpp [1]
- ◇ 48:15: [format] Point Formatted Example: User [1]
- ◇ 48:19: [format] Point Formatter Example: custom_1.hpp [2]

- ◇ 48:59: [format] Point Formatted Example: User [2]
- ◇ 49:52: [format] Point Formatter Example: custom_1.hpp [3]
- ◇ 53:16: [format] Point Formatted Example: User [3]
- ◇ 53:47: [format] References
- ◇ 54:37: [format] Questions

D.4 Miscellaneous Video Presentations

The sections that follow have some information on video lectures that cover various miscellaneous topics.

D.4.1 Video-Lecture Catalog

A catalog of the video lectures in this category is provided in the sections that follow.

D.4.1.1 Meshlab/Geomview Demo [2019-06-16]

The following is a link to the full video:
- ◇ https://youtu.be/X7A_7REjrsk [duration: 00:02:08]

The following are links to particular offsets within the video:
- ◇ 00:00: [misc] meshlab/geomview Demo

D.4.1.2 Accessing the SDE Using VM Software [2020-04-26]

The following is a link to the full video:
- ◇ https://youtu.be/Sv6dpmZWxgE [duration: 00:04:59]

The following are links to particular offsets within the video:
- ◇ 00:00: [sde] Demonstration: Accessing the SDE Using a VM

D.4.1.3 Assertions and CMake Build Types Demonstration [2020-04-30]

The following is a link to the full video:
- ◇ https://youtu.be/lwp7BZpHrog [duration: 00:08:12]

The following are links to particular offsets within the video:
- ◇ 00:00: [cmake] Assertions and CMake Build Types Demonstration

D.4.1.4 Address Sanitizer (ASan) Demonstration [2020-04-26]

The following is a link to the full video:
- ◇ https://youtu.be/nkxGxWo2THo [duration: 00:07:28]

The following are links to particular offsets within the video:
- ◇ 00:00: [asan] Address Sanitizer Demo

D.4.1.5 Undefined-Behavior Sanitizer (UBSan) Demonstration [2020-04-26]

The following is a link to the full video:
- ◇ https://youtu.be/HvYn5pHgVsg [duration: 00:05:19]

The following are links to particular offsets within the video:
- ◇ 00:00: [ubsan] Undefined-Behavior Sanitizer Demo

D.4.1.6 Lcov Demonstration [2020-04-30]

The following is a link to the full video:

- https://youtu.be/_KM0rDQYFSg [duration: 00:14:48]

The following are links to particular offsets within the video:

- 00:00: [lcov] Lcov Demonstration

Bibliography

[1] D. Abrahams and A. Gurtovoy. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond*. Addison Wesley, Boston, MA, USA, 2004.

[2] A. Alexandrescu. *Modern C++ Design*. Addison Wesley, Upper Saddle River, NJ, USA, 2001.

[3] Boost web site. http://www.boost.org, 2021.

[4] B. Eckel. *Thinking in C++—Volume 1: Introduction to Standard C++*. Prentice Hall, 2nd edition, 2000.

[5] B. Eckel and C. Allison. *Thinking in C++—Volume 2: Practical Programming*. Prentice Hall, 1st edition, 2003.

[6] D. Di Gennaro. *Advanced C++ Metaprogramming*. CreateSpace Independent Publishing Platform, 2011.

[7] ISO/IEC 14882:1998 — programming languages — C++, September 1998.

[8] ISO/IEC 14882:2003 — programming languages — C++, October 2003.

[9] ISO/IEC 14882:2011 — information technology — programming languages — C++, September 2011.

[10] ISO/IEC 14882:2014 — information technology — programming languages — C++, December 2014.

[11] ISO/IEC 14882:2017 — information technology — programming languages — C++, December 2017.

[12] ISO/IEC 14882:2020 — information technology — programming languages — C++, December 2020.

[13] N. M. Josuttis. *The C++ Standard Library: A Tutorial and Reference*. Addison Wesley, Upper Saddle River, NJ, USA, 2nd edition, 2012.

[14] B. Karlsson. *Beyond the C++ Standard Library: An Introduction to Boost*. Addison Wesley, Upper Saddle River, NJ, USA, 2005.

[15] M. Kilpelainen. Overload resolution — selecting the function. *Overload*, 66:22–25, April 2005. Available online at http://accu.org/index.php/journals/268.

[16] A. Koenig and B. E. Moo. *Accelerated C++: Practical Programming by Example*. Addison Wesley, Upper Saddle River, NJ, USA, 2000.

[17] A. Langer and K. Kreft. *Standard C++ IOStreams and Locales*. Addison Wesley, 2000.

[18] S. B. Lippman, J. Lajoie, and B. E. Moo. *C++ Primer*. Addison Wesley, Upper Saddle River, NJ, USA, 4th edition, 2005.

[19] S. Meyers. *Effective C++: 50 Specific Ways to Improve Your Programs and Designs*. Addison Wesley, Menlo Park, California, 1992.

[20] S. Meyers. *More Effective C++: 35 New Ways to Improve Your Programs and Designs*. Addison Wesley, Menlo Park, California, 1996.

[21] S. Meyers. *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*. Addison Wesley, 2001.

[22] S. Meyers. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media, Cambridge, MA, USA, 2015.

[23] V. A. Punathambekar. How to interpret complex C/C++ declarations. http://www.codeproject.com/Articles/7042/How-to-interpret-complex-C-C-declarations, 2004.

[24] B. Schaling. *The Boost C++ Libraries*. XML Press, 2nd edition, 2014.

[25] Standard C++ Foundation web site. http://www.isocpp.org, 2021.

[26] B. Stroustrup. *Programming: Principles and Practice Using C++*. Addison Wesley, Upper Saddle River, NJ, USA, 2009.

[27] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, 4th edition, 2013.

[28] H. Sutter. *Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions*. Addison Wesley, 1999.

[29] H. Sutter. *More Exceptional C++: 40 New Engineering Puzzles, Programming Problems, and Solutions*. Addison Wesley, 2001.

[30] H. Sutter. *Exceptional C++ Style: 40 New Engineering Puzzles, Programming Problems, and Solutions*. Addison Wesley, 2004.

[31] H. Sutter and A. Alexandrescu. *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*. Addison Wesley, 2004.

[32] D. Vandevoorde and N. M. Josuttis. *C++ Templates: The Complete Guide*. Addison Wesley, 2002.

[33] A. Williams. *C++ Concurrency in Action*. Manning Publications, Shelter Island, NY, USA, 2nd edition, 2019.

