

BJARNE STROUSTRUP

THE CREATOR OF C++

Using
C++11
and
C++14



PROGRAMMING

Principles and Practice Using C++

SECOND EDITION



Programming

Principles and Practice

Using C++

Second Edition

Bjarne Stroustrup

◆◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

A complete list of photo sources and credits appears on pages 1273–1274.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Stroustrup, Bjarne, author.

Programming : principles and practice using C++ / Bjarne Stroustrup. — Second edition.

pages cm

Includes bibliographical references and index.

ISBN 978-0-321-99278-9 (pbk. : alk. paper)

1. C++ (Computer program language) I. Title.

QA76.73.C153S82 2014

005.13'3—dc23

2014004197

Copyright © 2014 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-99278-9

ISBN-10: 0-321-99278-4

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
Second printing, January 2015

Preface

“Damn the torpedoes!
Full speed ahead.”

—Admiral Farragut

Programming is the art of expressing solutions to problems so that a computer can execute those solutions. Much of the effort in programming is spent finding and refining solutions. Often, a problem is only fully understood through the process of programming a solution for it.

This book is for someone who has never programmed before but is willing to work hard to learn. It helps you understand the principles and acquire the practical skills of programming using the C++ programming language. My aim is for you to gain sufficient knowledge and experience to perform simple useful programming tasks using the best up-to-date techniques. How long will that take? As part of a first-year university course, you can work through this book in a semester (assuming that you have a workload of four courses of average difficulty). If you work by yourself, don't expect to spend less time than that (maybe 15 hours a week for 14 weeks).

Three months may seem a long time, but there's a lot to learn and you'll be writing your first simple programs after about an hour. Also, all learning is gradual: each chapter introduces new useful concepts and illustrates them with examples inspired by real-world uses. Your ability to express ideas in code – getting a computer to do what you want it to do – gradually and steadily increases as you go along. I never say, “Learn a month's worth of theory and then see if you can use it.”

Why would you want to program? Our civilization runs on software. Without understanding software you are reduced to believing in “magic” and will be locked out of many of the most interesting, profitable, and socially useful technical fields of work. When I talk about programming, I think of the whole spectrum of computer programs from personal computer applications with GUIs (graphical user interfaces), through engineering calculations and embedded systems control applications (such as digital cameras, cars, and cell phones), to text manipulation applications as found in many humanities and business applications. Like mathematics, programming – when done well – is a valuable intellectual exercise that sharpens our ability to think. However, thanks to feedback from the computer, programming is more concrete than most forms of math, and therefore accessible to more people. It is a way to reach out and change the world – ideally for the better. Finally, programming can be great fun.

Why C++? You can’t learn to program without a programming language, and C++ directly supports the key concepts and techniques used in real-world software. C++ is one of the most widely used programming languages, found in an unsurpassed range of application areas. You find C++ applications everywhere from the bottom of the oceans to the surface of Mars. C++ is precisely and comprehensively defined by a nonproprietary international standard. Quality and/or free implementations are available on every kind of computer. Most of the programming concepts that you will learn using C++ can be used directly in other languages, such as C, C#, Fortran, and Java. Finally, I simply like C++ as a language for writing elegant and efficient code.

This is not the easiest book on beginning programming; it is not meant to be. I just aim for it to be the easiest book from which you can learn the basics of real-world programming. That’s quite an ambitious goal because much modern software relies on techniques considered advanced just a few years ago.

My fundamental assumption is that you want to write programs for the use of others, and to do so responsibly, providing a decent level of system quality; that is, I assume that you want to achieve a level of professionalism. Consequently, I chose the topics for this book to cover what is needed to get started with real-world programming, not just what is easy to teach and learn. If you need a technique to get basic work done right, I describe it, demonstrate concepts and language facilities needed to support the technique, provide exercises for it, and expect you to work on those exercises. If you just want to understand toy programs, you can get along with far less than I present. On the other hand, I won’t waste your time with material of marginal practical importance. If an idea is explained here, it’s because you’ll almost certainly need it.

If your desire is to use the work of others without understanding how things are done and without adding significantly to the code yourself, this book is not for you. If so, please consider whether you would be better served by another book and another language. If that is approximately your view of programming, please

also consider from where you got that view and whether it in fact is adequate for your needs. People often underestimate the complexity of programming as well as its value. I would hate for you to acquire a dislike for programming because of a mismatch between what you need and the part of the software reality I describe. There are many parts of the “information technology” world that do not require knowledge of programming. This book is aimed to serve those who do want to write or understand nontrivial programs.

Because of its structure and practical aims, this book can also be used as a second book on programming for someone who already knows a bit of C++ or for someone who programs in another language and wants to learn C++. If you fit into one of those categories, I refrain from guessing how long it will take you to read this book, but I do encourage you to do many of the exercises. This will help you to counteract the common problem of writing programs in older, familiar styles rather than adopting newer techniques where these are more appropriate. If you have learned C++ in one of the more traditional ways, you’ll find something surprising and useful before you reach Chapter 7. Unless your name is Stroustrup, what I discuss here is not “your father’s C++.”

Programming is learned by writing programs. In this, programming is similar to other endeavors with a practical component. You cannot learn to swim, to play a musical instrument, or to drive a car just from reading a book – you must practice. Nor can you learn to program without reading and writing lots of code. This book focuses on code examples closely tied to explanatory text and diagrams. You need those to understand the ideals, concepts, and principles of programming and to master the language constructs used to express them. That’s essential, but by itself, it will not give you the practical skills of programming. For that, you need to do the exercises and get used to the tools for writing, compiling, and running programs. You need to make your own mistakes and learn to correct them. There is no substitute for writing code. Besides, that’s where the fun is!

On the other hand, there is more to programming – much more – than following a few rules and reading the manual. This book is emphatically not focused on “the syntax of C++.” Understanding the fundamental ideals, principles, and techniques is the essence of a good programmer. Only well-designed code has a chance of becoming part of a correct, reliable, and maintainable system. Also, “the fundamentals” are what last: they will still be essential after today’s languages and tools have evolved or been replaced.

What about computer science, software engineering, information technology, etc.? Is that all programming? Of course not! Programming is one of the fundamental topics that underlie everything in computer-related fields, and it has a natural place in a balanced course of computer science. I provide brief introductions to key concepts and techniques of algorithms, data structures, user interfaces, data processing, and software engineering. However, this book is not a substitute for a thorough and balanced study of those topics.

Code can be beautiful as well as useful. This book is written to help you see that, to understand what it means for code to be beautiful, and to help you to master the principles and acquire the practical skills to create such code. Good luck with programming!

A note to students

Of the many thousands of first-year students we have taught so far using this book at Texas A&M University, about 60% had programmed before and about 40% had never seen a line of code in their lives. Most succeeded, so you can do it, too.

You don't have to read this book as part of a course. The book is widely used for self-study. However, whether you work your way through as part of a course or independently, try to work with others. Programming has an – unfair – reputation as a lonely activity. Most people work better and learn faster when they are part of a group with a common aim. Learning together and discussing problems with friends is not cheating! It is the most efficient – as well as most pleasant – way of making progress. If nothing else, working with friends forces you to articulate your ideas, which is just about the most efficient way of testing your understanding and making sure you remember. You don't actually have to personally discover the answer to every obscure language and programming environment problem. However, please don't cheat yourself by not doing the drills and a fair number of exercises (even if no teacher forces you to do them). Remember: programming is (among other things) a practical skill that you need to practice to master. If you don't write code (do several exercises for each chapter), reading this book will be a pointless theoretical exercise.

Most students – especially thoughtful good students – face times when they wonder whether their hard work is worthwhile. When (not if) this happens to you, take a break, reread this Preface, and look at Chapter 1 (“Computers, People, and Programming”) and Chapter 22 (“Ideals and History”). There, I try to articulate what I find exciting about programming and why I consider it a crucial tool for making a positive contribution to the world. If you wonder about my teaching philosophy and general approach, have a look at Chapter 0 (“Notes to the Reader”).

You might find the weight of this book worrying, but it should reassure you that part of the reason for the heft is that I prefer to repeat an explanation or add an example rather than have you search for the one and only explanation. The other major reason is that the second half of the book is reference material and “additional material” presented for you to explore only if you are interested in more information about a specific area of programming, such as embedded systems programming, text analysis, or numerical computation.

And please don't be too impatient. Learning any major new and valuable skill takes time and is worth it.

A note to teachers

No. This is not a traditional Computer Science 101 course. It is a book about how to construct working software. As such, it leaves out much of what a computer science student is traditionally exposed to (Turing completeness, state machines, discrete math, Chomsky grammars, etc.). Even hardware is ignored on the assumption that students have used computers in various ways since kindergarten. This book does not even try to mention most important CS topics. It is about programming (or more generally about how to develop software), and as such it goes into more detail about fewer topics than many traditional courses. It tries to do just one thing well, and computer science is not a one-course topic. If this book/course is used as part of a computer science, computer engineering, electrical engineering (many of our first students were EE majors), information science, or whatever program, I expect it to be taught alongside other courses as part of a well-rounded introduction.

Please read Chapter 0 (“Notes to the Reader”) for an explanation of my teaching philosophy, general approach, etc. Please try to convey those ideas to your students along the way.

ISO standard C++

C++ is defined by an ISO standard. The first ISO C++ standard was ratified in 1998, so that version of C++ is known as C++98. I wrote the first edition of this book while working on the design of C++11. It was most frustrating not to be able to use the novel features (such as uniform initialization, range-**for**-loops, move semantics, lambdas, and concepts) to simplify the presentation of principles and techniques. However, the book was designed with C++11 in mind, so it was relatively easy to “drop in” the features in the contexts where they belonged. As of this writing, the current standard is C++11 from 2011, and facilities from the upcoming 2014 ISO standard, C++14, are finding their way into mainstream C++ implementations. The language used in this book is C++11 with a few C++14 features. For example, if your compiler complains about

```
vector<int> v1;
vector<int> v2 {v1};    // C++14-style copy construction
```

use

```
vector<int> v1;
vector<int> v2 = v1;    // C++98-style copy construction
```

instead.

If your compiler does not support C++11, get a new compiler. Good, modern C++ compilers can be downloaded from a variety of suppliers; see www.stroustrup.com/compilers.html. Learning to program using an earlier and less supportive version of the language can be unnecessarily hard.

Support

The book's support website, www.stroustrup.com/Programming, contains a variety of material supporting the teaching and learning of programming using this book. The material is likely to be improved with time, but for starters, you can find

- Slides for lectures based on the book
- An instructor's guide
- Header files and implementations of libraries used in the book
- Code for examples in the book
- Solutions to selected exercises
- Potentially useful links
- Errata

Suggestions for improvements are always welcome.

Acknowledgments

I'd especially like to thank my late colleague and co-teacher Lawrence "Pete" Petersen for encouraging me to tackle the task of teaching beginners long before I'd otherwise have felt comfortable doing that, and for supplying the practical teaching experience to make the course succeed. Without him, the first version of the course would have been a failure. We worked together on the first versions of the course for which this book was designed and together taught it repeatedly, learning from our experiences, improving the course and the book. My use of "we" in this book initially meant "Pete and me."

Thanks to the students, teaching assistants, and peer teachers of ENGR 112, ENGR 113, and CSCE 121 at Texas A&M University who directly and indirectly helped us construct this book, and to Walter Daugherty, Hyunyoung Lee, Teresa Leyk, Ronnie Ward, and Jennifer Welch, who have also taught the course. Also thanks to Damian Dechev, Tracy Hammond, Arne Tolstrup Madsen, Gabriel Dos Reis, Nicholas Stroustrup, J. C. van Winkel, Greg Versoonder, Ronnie Ward, and Leor Zolman for constructive comments on drafts of this book. Thanks to Mogens Hansen for explaining about engine control software. Thanks to Al Aho, Stephen Edwards, Brian Kernighan, and Daisy Nguyen for helping me hide away from distractions to get writing done during the summers.

Thanks to Art Werschulz for many constructive comments based on his use of the first edition of this book in courses at Fordham University in New York City and to Nick Maclaren for many detailed comments on the exercises based on his use of the first edition of this book at Cambridge University. His students had dramatically different backgrounds and professional needs from the TAMU first-year students.

Thanks to the reviewers that Addison-Wesley found for me. Their comments, mostly based on teaching either C++ or Computer Science 101 at the college level, have been invaluable: Richard Enbody, David Gustafson, Ron McCarty, and K. Narayanaswamy. Also thanks to my editor, Peter Gordon, for many useful comments and (not least) for his patience. I'm very grateful to the production team assembled by Addison-Wesley; they added much to the quality of the book: Linda Begley (proofreader), Kim Arney (compositor), Rob Mauhar (illustrator), Julie Nahil (production editor), and Barbara Wood (copy editor).

Thanks to the translators of the first edition, who found many problems and helped clarify many points. In particular, Loïc Joly and Michel Michaud did a thorough technical review of the French translation that led to many improvements.

I would also like to thank Brian Kernighan and Doug McIlroy for setting a very high standard for writing about programming, and Dennis Ritchie and Kristen Nygaard for providing valuable lessons in practical language design.

Contents

Preface xxv

Chapter 0 Notes to the Reader 1

- 0.1 The structure of this book 2
 - 0.1.1 General approach 3
 - 0.1.2 Drills, exercises, etc. 4
 - 0.1.3 What comes after this book? 5
- 0.2 A philosophy of teaching and learning 6
 - 0.2.1 The order of topics 9
 - 0.2.2 Programming and programming language 10
 - 0.2.3 Portability 11
- 0.3 Programming and computer science 12
- 0.4 Creativity and problem solving 12
- 0.5 Request for feedback 12
- 0.6 References 13
- 0.7 Biographies 13
 - Bjarne Stroustrup 14
 - Lawrence “Pete” Petersen 15

Chapter 1 Computers, People, and Programming 17

- 1.1 Introduction 18
- 1.2 Software 19
- 1.3 People 21
- 1.4 Computer science 24
- 1.5 Computers are everywhere 25
 - 1.5.1 Screens and no screens 26
 - 1.5.2 Shipping 26
 - 1.5.3 Telecommunications 28
 - 1.5.4 Medicine 30

- 1.5.5 Information 31
- 1.5.6 A vertical view 33
- 1.5.7 So what? 34
- 1.6 Ideals for programmers 34

Part I The Basics 41

Chapter 2 Hello, World! 43

- 2.1 Programs 44
- 2.2 The classic first program 45
- 2.3 Compilation 47
- 2.4 Linking 51
- 2.5 Programming environments 52

Chapter 3 Objects, Types, and Values 59

- 3.1 Input 60
- 3.2 Variables 62
- 3.3 Input and type 64
- 3.4 Operations and operators 66
- 3.5 Assignment and initialization 69
 - 3.5.1 An example: detect repeated words 71
- 3.6 Composite assignment operators 73
 - 3.6.1 An example: find repeated words 73
- 3.7 Names 74
- 3.8 Types and objects 77
- 3.9 Type safety 78
 - 3.9.1 Safe conversions 79
 - 3.9.2 Unsafe conversions 80

Chapter 4 Computation 89

- 4.1 Computation 90
- 4.2 Objectives and tools 92
- 4.3 Expressions 94
 - 4.3.1 Constant expressions 95
 - 4.3.2 Operators 97
 - 4.3.3 Conversions 99
- 4.4 Statements 100
 - 4.4.1 Selection 102
 - 4.4.2 Iteration 109
- 4.5 Functions 113
 - 4.5.1 Why bother with functions? 115
 - 4.5.2 Function declarations 117

- 4.6 **vector** 117
 - 4.6.1 Traversing a **vector** 119
 - 4.6.2 Growing a **vector** 119
 - 4.6.3 A numeric example 120
 - 4.6.4 A text example 123
- 4.7 Language features 125

Chapter 5 **Errors** 133

- 5.1 Introduction 134
- 5.2 Sources of errors 136
- 5.3 Compile-time errors 136
 - 5.3.1 Syntax errors 137
 - 5.3.2 Type errors 138
 - 5.3.3 Non-errors 139
- 5.4 Link-time errors 139
- 5.5 Run-time errors 140
 - 5.5.1 The caller deals with errors 142
 - 5.5.2 The callee deals with errors 143
 - 5.5.3 Error reporting 145
- 5.6 Exceptions 146
 - 5.6.1 Bad arguments 147
 - 5.6.2 Range errors 148
 - 5.6.3 Bad input 150
 - 5.6.4 Narrowing errors 153
- 5.7 Logic errors 154
- 5.8 Estimation 157
- 5.9 Debugging 158
 - 5.9.1 Practical debug advice 159
- 5.10 Pre- and post-conditions 163
 - 5.10.1 Post-conditions 165
- 5.11 Testing 166

Chapter 6 **Writing a Program** 173

- 6.1 A problem 174
- 6.2 Thinking about the problem 175
 - 6.2.1 Stages of development 176
 - 6.2.2 Strategy 176
- 6.3 Back to the calculator! 178
 - 6.3.1 First attempt 179
 - 6.3.2 Tokens 181
 - 6.3.3 Implementing tokens 183
 - 6.3.4 Using tokens 185
 - 6.3.5 Back to the drawing board 186

- 6.4 Grammars 188
 - 6.4.1 A detour: English grammar 193
 - 6.4.2 Writing a grammar 194
- 6.5 Turning a grammar into code 195
 - 6.5.1 Implementing grammar rules 196
 - 6.5.2 Expressions 197
 - 6.5.3 Terms 200
 - 6.5.4 Primary expressions 202
- 6.6 Trying the first version 203
- 6.7 Trying the second version 208
- 6.8 Token streams 209
 - 6.8.1 Implementing `Token_stream` 211
 - 6.8.2 Reading tokens 212
 - 6.8.3 Reading numbers 214
- 6.9 Program structure 215

Chapter 7 Completing a Program 221

- 7.1 Introduction 222
- 7.2 Input and output 222
- 7.3 Error handling 224
- 7.4 Negative numbers 229
- 7.5 Remainder: `%` 230
- 7.6 Cleaning up the code 232
 - 7.6.1 Symbolic constants 232
 - 7.6.2 Use of functions 234
 - 7.6.3 Code layout 235
 - 7.6.4 Commenting 237
- 7.7 Recovering from errors 239
- 7.8 Variables 242
 - 7.8.1 Variables and definitions 242
 - 7.8.2 Introducing names 247
 - 7.8.3 Predefined names 250
 - 7.8.4 Are we there yet? 250

Chapter 8 Technicalities: Functions, etc. 255

- 8.1 Technicalities 256
- 8.2 Declarations and definitions 257
 - 8.2.1 Kinds of declarations 261
 - 8.2.2 Variable and constant declarations 262
 - 8.2.3 Default initialization 263

- 8.3 Header files 264
- 8.4 Scope 266
- 8.5 Function call and return 272
 - 8.5.1 Declaring arguments and return type 272
 - 8.5.2 Returning a value 274
 - 8.5.3 Pass-by-value 275
 - 8.5.4 Pass-by-**const**-reference 276
 - 8.5.5 Pass-by-reference 279
 - 8.5.6 Pass-by-value vs. pass-by-reference 281
 - 8.5.7 Argument checking and conversion 284
 - 8.5.8 Function call implementation 285
 - 8.5.9 **constexpr** functions 290
- 8.6 Order of evaluation 291
 - 8.6.1 Expression evaluation 292
 - 8.6.2 Global initialization 293
- 8.7 Namespaces 294
 - 8.7.1 **using** declarations and **using** directives 296

Chapter 9 Technicalities: Classes, etc. 303

- 9.1 User-defined types 304
- 9.2 Classes and members 305
- 9.3 Interface and implementation 306
- 9.4 Evolving a class 308
 - 9.4.1 **struct** and functions 308
 - 9.4.2 Member functions and constructors 310
 - 9.4.3 Keep details private 312
 - 9.4.4 Defining member functions 314
 - 9.4.5 Referring to the current object 317
 - 9.4.6 Reporting errors 317
- 9.5 Enumerations 318
 - 9.5.1 “Plain” enumerations 320
- 9.6 Operator overloading 321
- 9.7 Class interfaces 323
 - 9.7.1 Argument types 324
 - 9.7.2 Copying 326
 - 9.7.3 Default constructors 327
 - 9.7.4 **const** member functions 330
 - 9.7.5 Members and “helper functions” 332
- 9.8 The **Date** class 334

Part II Input and Output 343

Chapter 10 Input and Output Streams 345

- 10.1 Input and output 346
- 10.2 The I/O stream model 347
- 10.3 Files 349
- 10.4 Opening a file 350
- 10.5 Reading and writing a file 352
- 10.6 I/O error handling 354
- 10.7 Reading a single value 358
 - 10.7.1 Breaking the problem into manageable parts 359
 - 10.7.2 Separating dialog from function 362
- 10.8 User-defined output operators 363
- 10.9 User-defined input operators 365
- 10.10 A standard input loop 365
- 10.11 Reading a structured file 367
 - 10.11.1 In-memory representation 368
 - 10.11.2 Reading structured values 370
 - 10.11.3 Changing representations 374

Chapter 11 Customizing Input and Output 379

- 11.1 Regularity and irregularity 380
- 11.2 Output formatting 380
 - 11.2.1 Integer output 381
 - 11.2.2 Integer input 383
 - 11.2.3 Floating-point output 384
 - 11.2.4 Precision 385
 - 11.2.5 Fields 387
- 11.3 File opening and positioning 388
 - 11.3.1 File open modes 388
 - 11.3.2 Binary files 390
 - 11.3.3 Positioning in files 393
- 11.4 String streams 394
- 11.5 Line-oriented input 395
- 11.6 Character classification 396
- 11.7 Using nonstandard separators 398
- 11.8 And there is so much more 406

Chapter 12 A Display Model 411

- 12.1 Why graphics? 412
- 12.2 A display model 413
- 12.3 A first example 414

- 12.4 Using a GUI library 418
- 12.5 Coordinates 419
- 12.6 **Shapes** 420
- 12.7 Using **Shape** primitives 421
 - 12.7.1 Graphics headers and **main** 421
 - 12.7.2 An almost blank window 422
 - 12.7.3 **Axis** 424
 - 12.7.4 Graphing a function 426
 - 12.7.5 **Polygons** 427
 - 12.7.6 **Rectangles** 428
 - 12.7.7 **Fill** 431
 - 12.7.8 **Text** 431
 - 12.7.9 **Images** 433
 - 12.7.10 And much more 434
- 12.8 Getting this to run 435
 - 12.8.1 Source files 437

Chapter 13 Graphics Classes 441

- 13.1 Overview of graphics classes 442
- 13.2 **Point** and **Line** 444
- 13.3 **Lines** 447
- 13.4 **Color** 450
- 13.5 **Line_style** 452
- 13.6 **Open_polyline** 455
- 13.7 **Closed_polyline** 456
- 13.8 **Polygon** 458
- 13.9 **Rectangle** 460
- 13.10 Managing unnamed objects 465
- 13.11 **Text** 467
- 13.12 **Circle** 470
- 13.13 **Ellipse** 472
- 13.14 **Marked_polyline** 474
- 13.15 **Marks** 476
- 13.16 **Mark** 478
- 13.17 **Images** 479

Chapter 14 Graphics Class Design 487

- 14.1 Design principles 488
 - 14.1.1 Types 488
 - 14.1.2 Operations 490
 - 14.1.3 Naming 491
 - 14.1.4 Mutability 492