

# Programming

Principles and Practice  
using C++

Bjarne Stroustrup

Copyright

Library of congress card

printing

# Contents

Preface

Table of contents

Chapter 0: Notes to the reader

I. The Basics

1 Computers, People, and Programming

2 Hello world!

3 Objects, types, and values

4 Computation

5 Errors

6 Writing a program

7 Completing a program

8 Technicalities: functions, etc.

9 Technicalities: classes, etc.

II. Input and Output

10 Input and output streams

11 File streams and string streams

12 Graphs

13 Graph classes

14 Graph class design

15 Graphing functions

16 Graphical user interfaces

III. Data and algorithms

17 Vectors: memory management

18 Vectors: arrays

19 Vectors: exceptions and templates

20 STL: containers, iterators, and algorithms

21 STL: maps and algorithms

IV. Broadening the view

22 Ideals and history

23 Text manipulation

24 Numerics

25 Embedded systems programming

26 Testing

27 The C programming language

Appendices

A Language summary

B Standard-library summary

C Glossary

D Getting started with Visual Studio

E Installing FLTK

F GUI Implementation

Index

“Damn the torpedoes!  
Full speed ahead”  
–Admiral Farragut

## Preface

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 acquire the principles and 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 system 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 – hopefully 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 non-proprietary 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'll 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 if 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 and well as its value. I would hate for you to acquire a dislike for programming because of a mismatch between what you needed and the part of the software reality we 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 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, we refrain from guessing how long it will take you to read this book, but we do encourage you to do many of our 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 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 lasts: 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 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 acquire the principles and practical skills to create such code. Good luck with programming!

## **A note to students**

Of the about 1,000 first-year students we have taught so far using drafts of this book at Texas A&M University, about 60% had programmed before and about 40% had never seen a line of code in their life. Most succeeded, so you can do it too.

You don't have to read this book as part of a course. I assume that the book will be 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 become a pointless theoretical exercise.

Most students – especially thoughtful good students – face times where they wonder whether their hard work is worthwhile. When (not if) this happens to you, take a break, re-read the foreword, 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 part of the reason is that the last third of the book is “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, we 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.

## **Acknowledgements**

I’d especially like to thank my 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 at Texas A&M University who directly and indirectly helped us construct this book, and to Walter Daugherty who has also taught the course. Also thanks to Damian Dechev, Tracy

Hammond, Arne Tolstrup Madsen, Gabriel Dos Reis, Nicholas Stroustrup, J. C. van Winkel, and Greg Versoonder for constructive comments on drafts of this book.

Thanks to the reviewers that Addison Wesley found for me. Their comments, mostly based on teaching either C++ or Computer Science 101 at 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 patience.

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.