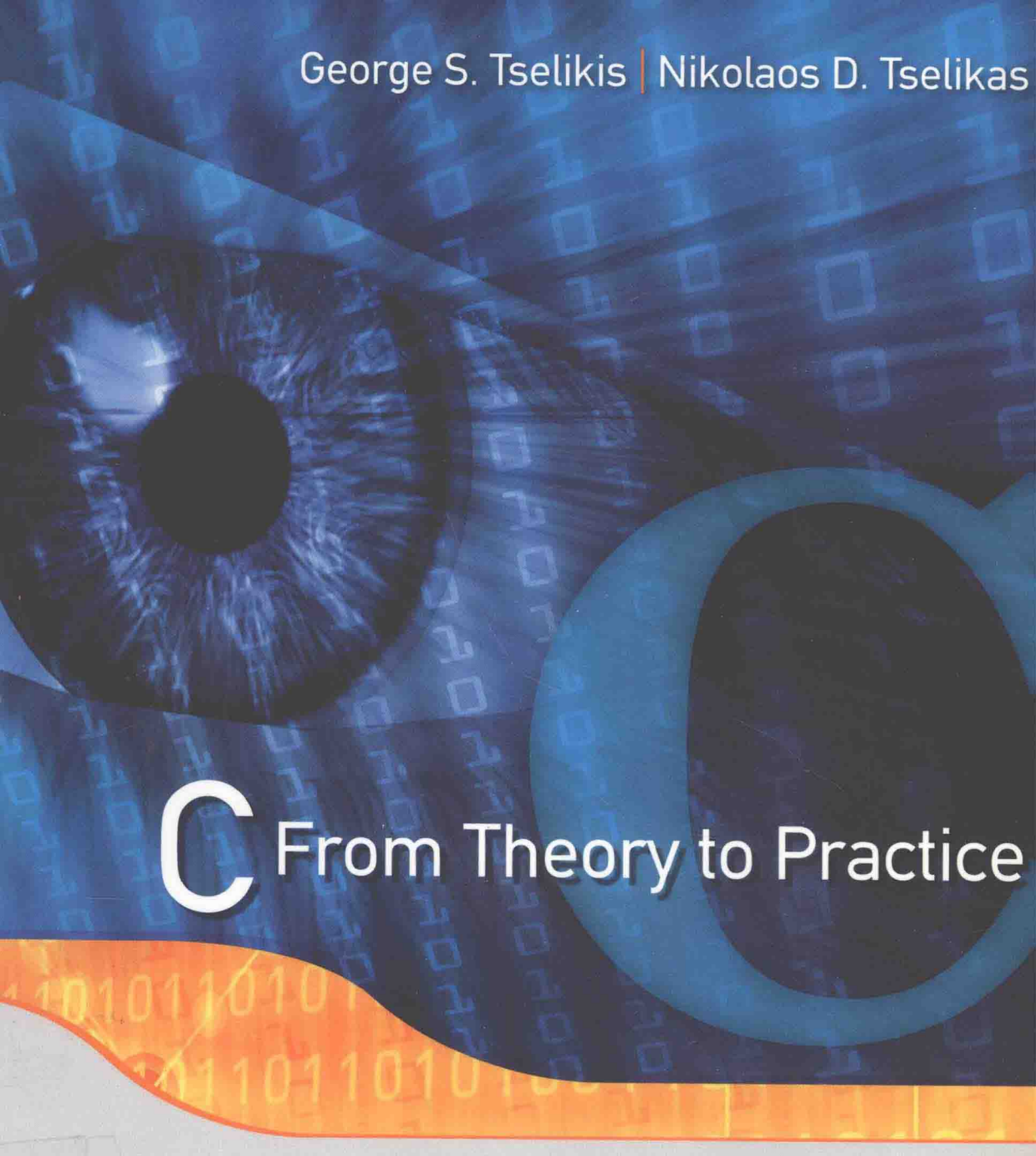
George S. Tselikis | Nikolaos D. Tselikas



C From Theory to Practice

- **500+** difficulty-scaled solved programming exercises
- Substantial, non-tiring, and easy-to-read presentation of the C theory
- Emphasis on the complex aspects of the C language
- Tips to avoid programming bugs and implement efficient and clear C programs

CRC Press
Taylor & Francis Group

C From Theory to Practice

George S. Tselikis | Nikolaos D. Tselikas

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20140304

International Standard Book Number-13: 978-1-4822-1450-5 (Paperback)

Preface

This book is primarily addressed to students who are taking a course on the C language, to those who desire to pursue self-study of the C language, as well as to experienced C programmers who want to test their skills. It could also prove useful to instructors of a C course, who are looking for explanatory programming examples to add in their lectures.

So what, exactly, differentiates this book from the others in the field? This book tests the skills of both beginners and advanced developers by providing an easy-to-read compilation of the C theory enriched with tips and advice as well as difficulty-scaled solved programming exercises.

When we first encountered the C language as students, we needed a book that would introduce us quickly to the secrets of the C language as well as provide in-depth knowledge on the subject—a book with a focus on providing inside information and programming knowledge through practical examples and meaningful advice. That is the spirit that this book aims to capture.

The programming examples are short but concrete, providing programming know-how in a substantial manner. Rest assured that if you are able to understand the examples and solve the exercises, you can safely go on to edit longer programs and start your programming career successfully.

For all of you who intend to deal with computer programming, a little bit of advice coming from our long experience may come in handy:

- Programming is a difficult task that requires a calm mind, a clear development plan, a methodical approach, patience, and luck as well.
- When coding, try to write in a simple and comprehensive way for your own benefit and for those who are going to read your code. Always remember that the debug, support, and upgrade of a code written in a complex way is a painful process.
- Hands-on! To learn the features of a programming language, you must write your own programs and experiment with them.
- Programming is definitely a creative activity, but do not forget that there are plenty of creative, pleasant, and less stressful activities in life. Do not waste your life in front of a computer screen, searching for *losing* pointers, buffer overflows, memory overruns, and buggy conditions. Program for some time and then have some fun. Keep living and do not keep programming.

Enjoy the C flight; it would be safe, with some turbulence, though.

Acknowledgments

We would like to thank all our colleagues and professors, namely, C. Antonopoulos, G. Bardis, V. Doufexi, G. Kapitsaki, E. Kofidis, T. Kotaras, G. Koulouras, L. Lazos, N. Sagias, K. Slavakis, P. Stamatopoulos, E. Valamonte, and K. Yiannopoulos, whose comments, fruitful suggestions, and criticisms contributed to the improvement of this book. Many thanks also to Ruijun He, Laurie Schlags, and Jennifer Stair, associate editor, project coordinator, and project editor, respectively, at CRC Press/Taylor & Francis Group, and to Remya Divakaran, project manager at SPi Global, India, for their advice and assistance during the production of this book.

Authors

Dr. George S. Tselikis received his Dipl–Ing and his PhD from the School of Electrical and Computer Engineering of the National Technical University of Athens (NTUA) in 1993 and 1997, respectively. In 1998, he joined the COMET group in the Center for Telecommunications Research at Columbia University, New York and worked as a postdoc research associate. He was a founding member of 4Plus S.A. (1999–2013), where he worked in the development of network protocols and services. He has a long working experience in the telecom area, and his research interests focus on software specification, development, and testing of network protocols and services in wired and wireless networks. During his professional career, he has collaborated with big players in the telecom industry like Siemens, Nokia, and Alcatel. Since 2004, he has been a visiting lecturer in several universities and technical institutes. He teaches courses related to network technologies, protocols, and communications, as well as the C language.

Dr. Nikolaos D. Tselikas received his Dipl–Ing and his PhD from the School of Electrical and Computer Engineering of the National Technical University of Athens (NTUA) in 1999 and 2004, respectively. His research interests focus on the specification and implementation of network services and applications. He has participated in several European and national research projects in this field. During his involvement in the research projects, he has collaborated in software design and development topics with big players in the telecom industry like Ericsson, Siemens, Alcatel, and Vodafone. He has served as a visiting lecturer in the Department of Informatics and Telecommunications, University of Peloponnese, Greece, since 2006 and was appointed lecturer in 2009. He currently serves as an assistant professor at the same university, where he teaches courses related to applications' programming and network services, as well as the C language.

Contents

1

Introduction to C

Before getting into the details of C language, this chapter presents, in brief, its history, evolution, strengths, and weaknesses. Then, we'll discuss some basic concepts that we'll need in order to write our first program.

History of C

The C language was developed at Bell Laboratories in the early 1970s by Dennis Richie and others. At that time, the UNIX operating system, also developed at Bell Labs, was written in assembly language. Programs written in assembly are usually hard to debug, maintain, and enhance, and UNIX was no exception. Richie decided to rewrite UNIX's code in another language that would make the execution of these tasks easier. He named the language C because it was the evolution of an earlier language written by Ken Thompson, called B.

The C language continued to evolve during the 1970s, and since then it is widely used by thousands of programmers for the development of various software applications.

ANSI Standard

The rapid expansion of the C language and its increased popularity led many companies to develop their own C compilers. Due to the absence of an official standard, their development relied on the bible of C programmers, the legendary K&R book, written by Brian Kernighan and Dennis Ritchie in 1978.

However, the K&R book was not written in the precise way that a standard requires. Features that were not clearly described could be implemented in different ways. As a result, the same program could be compiled with one C compiler and not with another. In parallel, the C language continued to evolve with the addition of new features and the replacement or obsolescence of existing ones.

The need for the standardization of C language became apparent. In 1983, the American National Standard Institute (ANSI) began the development of C standard that was completed and formally approved, in 1989, as ANSI C or Standard C. This standard describes precisely the features, characteristics, and properties of the C language, and every C compiler must support it. The addition of some new features to the ANSI standard during the late 1990s led to a new standard called C99.

This book describes the C language based on the ANSI/ISO C standard [1].

Advantages of C

Despite the emergence of many programming languages, C still remains competitive and popular in the programming world for several reasons, such as the following:

1. It is a flexible language, which can be used for the development of different kinds of applications, from embedded systems and operating systems to industrial applications. For example, we've used C in the area of communication networks for the development of network protocols and the support of network services.
2. A C program is executed very fast.
3. It is a small language. Its vocabulary consists of a few words with special meanings.
4. It is portable, meaning that a C program may run under different operating systems.
5. It supports structural programming, meaning that a C program may contain functions to perform several tasks.
6. It is a language very close to the hardware.
7. Every C compiler comes with a set of ready-to-use functions, called C standard library. The use of these library functions saves considerable programming effort.
8. Thanks to the popularity of the C language, there are many C compilers available, some of them free of charge.
9. Learning C is the first step toward object-oriented programming. Most of the C features are supported in several object-oriented languages, like C++, Java, and C#.

Disadvantages of C

1. Because the C language does not impose many restrictions on the use of its features, it is an error-prone language. When writing a C program, be cautious because you may insert bugs that won't be detected by the compiler.
2. Although C is a small language, it is not an "easy" to use language. C code can be very hard to understand even if it consists of a small number of lines. After reading this book, check out the International Obfuscated C Code Contest (http://www.ioccc.org) to get a feeling.
3. It is not an object-oriented language.

C Program Life Cycle

The life cycle of a C program involves several steps: writing the source code, its compilation, linking the code produced by the compiler with the code of the used library functions, and executing the program.

Usually, a C compiler provides an integrated development environment that allows us to perform this set of operations without leaving the environment.

Write a C Program

To write a C program, you can use any available text editor. The source code must be saved in a file with extension .c.

When the size of the code is very large, it is a common practice to divide the code into several files in order to facilitate tasks like debugging and maintenance. In such cases, each file is compiled separately.

First C Program

Our first program will be a "rock" version of the program that most programmers begin with. Instead of the classical K&R `"Hello world"`, our program displays `"Ramones: Hey Ho, Let's Go"`.

```
#include <stdio.h>
int main()
{
  printf("Ramones: Hey Ho, Let's Go\n");
  return 0;
}
```

The following sections explain the significance of each program line.

`#include` Directive

The `#include` directive instructs the compiler to include the contents of the specified file in the program before it is compiled. The C standard library contains a number of header files; each contains information about a specific part of the library. For example, the `stdio.h` (standard input output) file contains information about data input and output functions. When you get familiar with C language, you may edit your own header files and include them in your programs.

When the program is compiled, the compiler searches for the included files. The brackets < > instruct the compiler to look into predefined folders. If a file is not found, the compiler will raise an error message and the compilation fails.

Regarding syntax, notice that a directive starts with an # and does not end with a semicolon (;).

`main()` Function

Every C program must contain a function named `main()`. In C, a function is a series of statements that have been grouped together and given a name. The statements of the

program must be enclosed in braces {}. A statement is a command that will be executed when the program runs. Unlike the directives, a statement ends almost always with a semicolon.

The `main()` function is called *automatically* when the program runs. It ends when its last statement is executed, unless an exit function (like the `return`) is called earlier. The word `int` indicates that `main()` must return a status code to the operating system when it terminates. This value is returned with the `return` statement; the value `0` indicates normal termination.

The declaration `int main()` is fairly common. However, you may see other declarations like

```
void main()
{
  ...
}
```

The word `void` indicates that the `main()` function doesn't return any value. Although a C compiler may accept this declaration, it is illegal according to the C standard because `main()` must return a value. On the other hand, the declaration

```
main()
{
  ...
}
```

is acceptable because the return type is `int` by default.

This declaration is also acceptable:

```
int main(void)
{
  ...
}
```

The word `void` inside the parentheses indicates that `main()` has no arguments.

In Chapter 11, we'll see another declaration of `main()`, where it accepts arguments.

As discussed, we may use functions of the standard library in our programs. For example, `printf()` is a standard library function that is used for data output. The reason to include the `stdio.h` file is that it contains information about `printf()`. The new line character `'\n'` instructs `printf()` to advance to the next line. We'll discuss more about `printf()` in the next chapter.

Until Chapter 11, where you'll learn how to write other functions, `main()` will be the only function in our programs.

Add Comments

A well-documented program should contain comments to explain its complicated parts and make it easier to understand. A comment begins with the `/*` symbol and ends with `*/`. Comments can extend in more than one line.