

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

© 2014 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed on acid-free paper
Version Date: 20140304

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

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>

and the CRC Press Web site at
<http://www.crcpress.com>

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

Preface.....	xiii
Acknowledgments	xv
Authors	xvii
1. Introduction to C.....	1
History of C	1
ANSI Standard	1
Advantages of C	2
Disadvantages of C	2
C Program Life Cycle	2
Write a C Program	3
First C Program.....	3
#include Directive	3
main() Function.....	3
Add Comments	4
Compilation	5
Common Errors.....	6
Linking.....	6
Run the Program.....	6
Reference	7
2. Data Types, Variables, and Data Output	9
Variables.....	9
Rules for Naming Variables	9
Variable Name Conventions.....	10
Declaring Variables	10
Assigning Values to Variables.....	12
Constants.....	13
#define Directive.....	13
printf() Function.....	14
Escape Sequences.....	15
Conversion Specifications	16
Return Value	17
Printing Variables	18
Optional Fields	19
Precision	19
Field Width.....	21
Prefix	22
Flags	22
Type Casting.....	23
Exercises	24
Unsolved Exercises.....	27

3. Getting Input with <code>scanf()</code>	31
<code>scanf()</code> Function.....	31
<code>scanf()</code> Examples.....	31
Use of Ordinary Characters	33
<code>scanf()</code> Return Value	33
Exercises	34
Unsolved Exercises	40
4. Operators	41
Assignment Operator.....	41
Arithmetic Operators	41
Increment and Decrement Operators	42
Relational Operators.....	44
Exercises	44
Not Operator !	45
Exercises	46
Compound Operators	46
Exercise.....	47
Logical Operators.....	47
Operator <code>&&</code>	47
Operator <code> </code>	48
Exercises	48
Comma Operator	50
Exercise.....	51
<code>sizeof</code> Operator	52
<code>enum</code> Type.....	52
Bitwise Operators	53
<code>&</code> Operator	54
<code> </code> Operator	54
<code>^</code> Operator	54
<code>~</code> Operator.....	54
Exercise.....	55
Shift Operators	55
<code>>></code> Operator	55
<code><<</code> Operator	56
Exercises	56
Operator Precedence	61
Unsolved Exercises	61
5. Program Control	63
<code>if</code> Statement	63
Common Errors.....	64
<code>if-else</code> Statement.....	64
Nested <code>if</code> Statements.....	65
Exercises	68
Conditional Operator <code>?:</code>	76
Exercises	77
<code>switch</code> Statement.....	79
<code>switch</code> versus <code>if</code>	82

Exercises	82
Unsolved Exercises	86
6. Loops	89
for Statement	89
Omitting Expressions	91
Exercises	92
break Statement	94
continue Statement	94
Exercises	95
Nested Loops	103
Exercises	105
while Statement	109
Exercises	110
do-while Statement	118
Exercises	119
goto Statement	121
Unsolved Exercises	122
7. Arrays	125
Declaring Arrays	125
Accessing Array Elements	126
Array Initialization	127
Exercises	128
Two-Dimensional Arrays	140
Two-Dimensional Array Declaration	140
Accessing the Elements of a Two-Dimensional Array	140
Two-Dimensional Array Initialization	141
Exercises	142
Unsolved Exercises	151
8. Pointers	153
Pointers and Memory	153
Declaring Pointers	153
Pointer Initialization	154
NULL Value	155
Use a Pointer	155
Exercises	157
void* Pointer	163
Use of const Keyword	163
Pointer Arithmetic	163
Pointers and Integers	164
Subtracting Pointers	164
Comparing Pointers	165
Exercises	165
Pointers and Arrays	168
Exercises	171
Arrays of Pointers	180
Exercises	181

Pointer to Pointer	182
Exercises	183
Pointers and Two-Dimensional Arrays	184
Exercises	187
Pointer to Function	187
Exercise.....	189
Array of Pointers to Functions.....	189
Unsolved Exercises	191
9. Characters.....	193
char Type.....	193
Exercises	194
getchar() Function.....	198
Exercises	198
10. Strings.....	203
String Literals	203
Storing Strings.....	203
Exercises	204
Writing Strings.....	205
Exercise.....	207
Pointers and String Literals	207
Exercises	209
Read Strings.....	210
For a Safe Reading of Strings	212
Exercises	213
String Functions.....	217
strlen() Function.....	217
Exercises	218
strcpy() Function	224
Exercises	225
strncpy() Function.....	227
strcat() Function.....	227
strcmp() and strncmp() Functions.....	228
Exercises	229
Two-Dimensional Arrays and Strings	236
Exercises	237
Unsolved Exercises	240
11. Functions	243
Function Declaration.....	243
Return Type	243
Function Parameters.....	244
Function Definition	244
return Statement	245
Function Call	246
Function Call without Parameters.....	246
Function Call with Parameters	248
Passing Values	249

- Exercises 251
- Variables Scope..... 258
 - Local Variables..... 258
 - Global Variables..... 260
 - Static Variables..... 261
- Arrays as Arguments 262
- Exercises 264
- Function Call with Parameter Two-Dimensional Array 284
- Exercises 285
- Passing Data in `main()` Function 289
- Exercises 290
- Functions with Variable Number of Parameters..... 291
- Recursive Functions 293
- Exercises 294
- Unsolved Exercises 298
- 12. Searching and Sorting Arrays 301**
 - Searching Arrays 301
 - Linear Search..... 301
 - Exercises 301
 - Binary Search..... 303
 - Exercises 304
 - Sorting Arrays 306
 - Selection Sort 306
 - Exercises 306
 - Insertion Sort 310
 - Exercise..... 311
 - Bubble Sort 312
 - Exercise..... 313
 - `bsearch()` and `qsort()` Library Functions 314
 - Exercise..... 315
- 13. Structures and Unions 317**
 - Structures 317
 - Defining a Structure 317
 - Declaring Structure Variables..... 318
 - Accessing the Structure Fields 320
 - Pointer to a Structure Field..... 321
 - Structure Operations 321
 - Structures Containing Arrays..... 322
 - Structures Containing Pointers..... 323
 - Structures Containing Structures..... 323
 - Exercise..... 324
 - Bit Fields 325
 - Pointer to Structure..... 326
 - Arrays of Structures 327
 - Exercises 329
 - Structures as Function Arguments 331
 - Exercises 332

Unions.....	342
Using Unions.....	342
Access Union Fields.....	343
Exercise.....	345
Unsolved Exercises.....	348
14. Memory Management and Data Structures.....	351
Memory Blocks.....	351
Static Memory Allocation.....	352
Dynamic Memory Allocation.....	353
malloc() Function.....	354
free() Function.....	355
memcpy() and memmove() Functions.....	356
memcmp() Function.....	358
Exercises.....	359
Dynamic Data Structures.....	371
Linked List.....	371
Insert a Node.....	371
Delete a Node.....	372
Examples.....	372
Implementing a Stack.....	373
Exercise.....	373
Implementing a Queue.....	377
Exercise.....	377
Implementing a Linked List.....	380
Exercises.....	380
Unsolved Exercises.....	386
15. Files.....	389
Files in C.....	389
Open a File.....	390
Close a File.....	392
Process a File.....	392
Write Data in a Text File.....	392
fputs() Function.....	392
fprintf() Function.....	393
Exercise.....	393
fputc() Function.....	394
Exercises.....	395
Read Data from a Text File.....	397
fscanf() Function.....	398
Exercises.....	400
fgets() Function.....	402
Exercise.....	404
fgetc() Function.....	405
Exercises.....	405
End of File.....	409
fseek() Function.....	410
ftell() Function.....	410

Write and Read Data from a Binary File	411
fwrite() Function.....	411
fread() Function.....	412
Exercises.....	412
Exercises.....	415
feof() Function	422
Exercise.....	423
Unsolved Exercises	423
16. Preprocessor Directives and Macros	425
Simple Macros	425
Macros with Parameters	426
# and ## Preprocessor Operators.....	428
Preprocessor Directives and Conditional Compilation	430
#if, #else, #elif, and #endif Directives.....	430
#ifdef, #ifndef, and #undef Directives	432
defined Operator.....	433
Exercises.....	433
Unsolved Exercises	438
17. Review Exercises.....	441
Appendix A	473
Appendix B.....	475
Appendix C	477
Appendix D.....	491
Bibliography.....	493
Index	495

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.