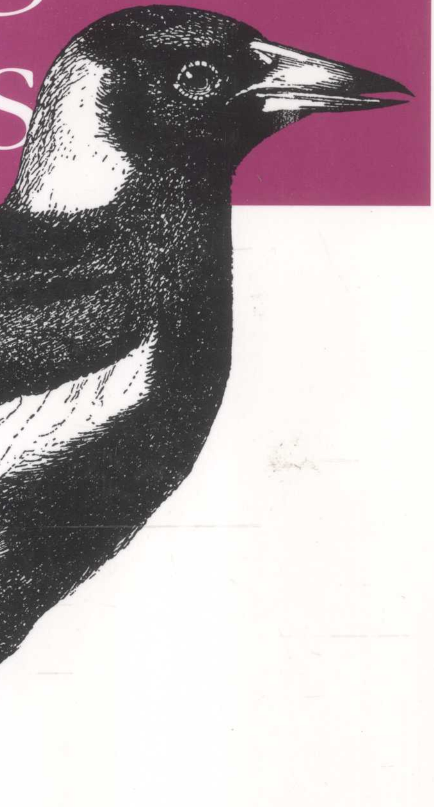
学习使用C指针（影印版）

Understanding and Using C Pointers



O'REILLY®

东南大學出版社

Richard Reese 著

学习使用C指针（影印版）

Understanding and Using C Pointers

Richard Reese 著

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

南京 东南大学出版社

图书在版编目（CIP）数据

学习使用C指针：英文/(美)雷斯（Reese, R.)著．—影印本．—南京：东南大学出版社，2014.1

书名原文：Understanding and Using C Pinters

ISBN 978-7-5641-4603-0

I. ① 学…　II. ①雷…　III. ① C语言－程序设计－英文 IV. ① TP312

中国版本图书馆CIP数据核字（2013）第246096号

江苏省版权局著作权合同登记
图字：10-2013-381号

学习使用C指针（影印版）

出版发行：东南大学出版社
地　　址：南京四牌楼2号　　邮编：210096
出 版 人：江建中
网　　址：http://www.seupress.com
电子邮件：press@seupress.com
印　　刷：扬中市印刷有限公司
开　　本：787毫米×980毫米　16开本
印　　张：14.25
字　　数：279千字
版　　次：2014年1月第1版
印　　次：2014年1月第1次印刷
书　　号：ISBN 978-7-5641-4603-0
定　　价：46.00元（册）

本社图书若有印装质量问题，请直接与营销部联系。电话（传真）：025-83791830

Preface

C is an important language and has had extensive treatment over the years. Central to the language are pointers that provide much of the flexibility and power found in the language. It provides the mechanism to dynamically manipulate memory, enhances support for data structures, and enables access to hardware. This power and flexibility comes with a price: pointers can be difficult to master.

Why This Book Is Different

Numerous books have been written about C. They usually offer a broad coverage of the language while addressing pointers only to the extent necessary for the topic at hand. Rarely do they venture beyond a basic treatment of pointers and most give only cursory coverage of the important memory management technology involving the stack and the heap. Yet without this discussion, only an incomplete understanding of pointers can be obtained. The stack and heap are areas of memory used to support functions and dynamic memory allocation, respectively.

Pointers are complex enough to deserve more in-depth treatment. This book provides that treatment by focusing on pointers to convey a deeper understanding of C. Part of this understanding requires a working knowledge of the program stack and heap along with the use of pointers in this context. Any area of knowledge can be understood at varying degrees, ranging from a cursory overview to an in-depth, intuitive understanding. That higher level of understanding for C can only be achieved with a solid understanding of pointers and the management of memory.

The Approach

Programming is concerned with manipulating data that is normally located in memory. It follows that a better understanding of how C manages memory will provide insight that translates to better programming. While it is one thing to know that the `malloc` function allocates memory from the heap, it is another thing to understand the implications of this allocation. If we allocate a structure whose logical size is 45, we may be surprised to learn that more than 45 bytes are typically allocated and the memory allocated may be fragmented.

When a function is called, a stack frame is created and pushed onto the program stack. Understanding stack frames and the program stack will clarify the concepts of passing by value and passing by pointer. While not necessarily directly related to pointers, the understanding of stack frames also explains how recursion works.

To facilitate the understanding of pointers and memory management techniques, various memory models will be presented. These range from a simple linear representation of memory to more complex diagrams that illustrate the state of the program stack and heap for a specific example. Code displayed on a screen or in a book is a static representation of a dynamic program. The abstract nature of this representation is a major stumbling block to understanding a program's behavior. Memory models go a long way to helping bridge this gap.

Audience

The C language is a block structured language whose procedural aspects are shared with most modern languages such as C++ and Java. They all use a program stack and heap. They all use pointers, which are often disguised as references. We assume that you have a minimal understanding of C. If you are learning C, then this book will provide you with a more comprehensive treatment of pointers and memory than is found in other books. It will expand your knowledge base regarding C and highlight unfamiliar aspects of C. If you are a more experienced C or C++ programmer, this book will help you fill in possible gaps regarding C and will enhance your understanding of how they work "under the hood," thus making you a better programmer. If you are a C# or Java developer, this book will help you better understand C and provide you with insight into how object-oriented languages deal with the stack and the heap.

Organization

The book is organized along traditional topics such as arrays, structures, and functions. However, each chapter focuses on the use of pointers and how memory is managed. For example, passing and returning pointers to and from functions are covered, and we also depict their use as part of stack frames and how they reference memory in the heap.

Chapter 1, Introduction

This chapter covers pointer basics for those who are not necessarily proficient or are new to pointers. This includes pointer operators and the declaration of different types of pointers such as constant pointers, function pointers, and the use of `NULL` and its closely related variations. This can have a significant impact on how memory is allocated and used.

Chapter 2, Dynamic Memory Management in C

Dynamic memory allocation is the subject of Chapter 2. The standard memory allocation functions are covered along with techniques for dealing with the deallocation of memory. Effective memory deallocation is critical to most applications, and failure to adequately address this activity can result in memory leaks and dangling pointers. Alternative deallocation techniques, including garbage collection and exception handlers, are presented.

Chapter 3, Pointers and Functions

Functions provide the building blocks for an application's code. However, passing or returning data to and from functions can be confusing to new developers. This chapter covers techniques for passing data, along with common pitfalls that occur when returning information by pointers. This is followed by extensive treatment of function pointers. These types of pointers provide yet another level of control and flexibility that can be used to enhance a program.

Chapter 4, Pointers and Arrays

While array notation and pointer notation are not completely interchangeable, they are closely related. This chapter covers single and multidimensional arrays and how pointers are used with them. In particular, passing arrays and the various nuisances involved in dynamically allocating arrays in both a contiguous and a noncontiguous manner are explained and illustrated with different memory models.

Chapter 5, Pointers and Strings

Strings are an important component of many applications. This chapter addresses the fundamentals of strings and their manipulation with pointers. The literal pool and its impact on pointers is another often neglected feature of C. Illustrations are provided to explain and illuminate this topic.

Chapter 6, Pointers and Structures

Structures provide a very useful way of ordering and manipulating data. Pointers enhance the utility of structures by providing more flexibility in how they can be constructed. This chapter presents the basics of structures as they relate to memory allocation and pointers, followed by examples of how they can be used with various data structures.

Chapter 7, Security Issues and the Improper Use of Pointers

As powerful and useful as pointers can be, they are also the source of many security problems. In this chapter, we examine the fundamental problems surrounding buffer overflow and related pointer issues. Techniques for mitigating many of these problems are presented.

Chapter 8, Odds and Ends

The last chapter addresses other pointer techniques and issues. While C is not an object-oriented language, many aspects of object-oriented programming can be incorporated into a C program, including polymorphic behavior. The essential elements of using pointers with threads are illustrated. The meaning and use of the `restrict` keyword are covered.

Summary

This book is intended to provide a more in-depth discussion of the use of pointers than is found in other books. It presents examples ranging from the core use of pointers to obscure uses of pointers and identifies common pointer problems.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, if this book includes code examples, you may use the code in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Understanding and Using C Pointers* by Richard Reese (O'Reilly). Copyright 2013 Richard Reese, Ph.D. 978-1-449-34418-4."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online (*www.safaribooksonline.com*) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://oreil.ly/Understand_Use_CPointers*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Table of Contents

CHAPTER 1

Introduction

A solid understanding of pointers and the ability to effectively use them separates a novice C programmer from a more experienced one. Pointers pervade the language and provide much of its flexibility. They provide important support for dynamic memory allocation, are closely tied to array notation, and, when used to point to functions, add another dimension to flow control in a program.

Pointers have long been a stumbling block in learning C. The basic concept of a pointer is simple: it is a variable that stores the address of a memory location. The concept, however, quickly becomes complicated when we start applying pointer operators and try to discern their often cryptic notations. But this does not have to be the case. If we start simple and establish a firm foundation, then the advanced uses of pointers are not hard to follow and apply.

The key to comprehending pointers is understanding how memory is managed in a C program. After all, pointers contain addresses in memory. If we don't understand how memory is organized and managed, it is difficult to understand how pointers work. To address this concern, the organization of memory is illustrated whenever it is useful to explain a pointer concept. Once you have a firm grasp of memory and the ways it can be organized, understanding pointers becomes a lot easier.

This chapter presents an introduction to pointers, their operators, and how they interact with memory. The first section examines how they are declared, the basic pointer operators, and the concept of null. There are various types of "nulls" supported by C so a careful examination of them can be enlightening.

The second section looks more closely at the various memory models you will undoubtedly encounter when working with C. The model used with a given compiler and operating system environment affects how pointers are used. In addition, we closely examine various predefined types related to pointers and the memory models.

Pointer operators are covered in more depth in the next section, including pointer arithmetic and pointer comparisons. The last section examines constants and pointers. The numerous declaration combinations offer many interesting and often very useful possibilities.

Whether you are a novice C programmer or an experienced programmer, this book will provide you with a solid understanding of pointers and fill the gaps in your education. The experienced programmer will want to pick and choose the topics of interest. The beginning programmer should probably take a more deliberate approach.

Pointers and Memory

When a C program is compiled, it works with three types of memory:

Static/Global

Statically declared variables are allocated to this type of memory. Global variables also use this region of memory. They are allocated when the program starts and remain in existence until the program terminates. While all functions have access to global variables, the scope of static variables is restricted to their defining function.

Automatic

These variables are declared within a function and are created when a function is called. Their scope is restricted to the function, and their lifetime is limited to the time the function is executing.

Dynamic

Memory is allocated from the heap and can be released as necessary. A pointer references the allocated memory. The scope is limited to the pointer or pointers that reference the memory. It exists until it is released. This is the focus of Chapter 2.

Table 1-1 summarizes the scope of and lifetime of variables used in these memory regions.

Table 1-1. Scope and lifetime

	Scope	Lifetime
Global	The entire file	The lifetime of the application
Static	The function it is declared within	The lifetime of the application
Automatic (local)	The function it is declared within	While the function is executing
Dynamic	Determined by the pointers that reference this memory	Until the memory is freed

Understanding these types of memory will enable you to better understand how pointers work. Most pointers are used to manipulate data in memory. Understanding how memory is partitioned and organized will clarify how pointers manipulate memory.

A pointer variable contains the address in memory of another variable, object, or function. An object is considered to be memory allocated using one of the memory allocation functions, such as the `malloc` function. A pointer is normally declared to be of a specific type depending on what it points to, such as a pointer to a `char`. The object may be any C data type such as integer, character, string, or structure. However, nothing inherent in a pointer indicates what type of data the pointer is referencing. A pointer only contains an address.

Why You Should Become Proficient with Pointers

Pointers have several uses, including:

- Creating fast and efficient code
- Providing a convenient means for addressing many types of problems
- Supporting dynamic memory allocation
- Making expressions compact and succinct
- Providing the ability to pass data structures by pointer without incurring a large overhead
- Protecting data passed as a parameter to a function

Faster and more efficient code can be written because pointers are closer to the hardware. That is, the compiler can more easily translate the operation into machine code. There is not as much overhead associated with pointers as might be present with other operators.

Many data structures are more easily implemented using pointers. For example, a linked list could be supported using either arrays or pointers. However, pointers are easier to use and map directly to a next or previous link. An array implementation requires array indexes that are not as intuitive or as flexible as pointers.

Figure 1-1 illustrates how this can be visualized using arrays and pointers for a linked list of employees. The lefthand side of the figure uses an array. The head variable indicates that the linked list's first element is at index 10 of the array. Each array's element contains a structure that represents an employee. The structure's `next` field holds the index in the array of the next employee. The shaded elements represent unused array elements.

The righthand side shows the equivalent representation using pointers. The head variable holds a pointer to the first employee's node. Each node holds employee data as well as a pointer to the next node in the linked list.

The pointer representation is not only clearer but also more flexible. The size of an array typically needs to be known when it is created. This will impose a restriction on the