MEMORY AS A PROGRAMMING CONCEPT IN C AND C++

FRANTISEK FRANEK
McMaster University

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2004

Printed in the United States of America

Typefaces Utopia 9.5/13.5 pt. and ITC Kabel *System* AMS-TEX [FH]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging in Publication data
Franek, F. (Frantisek)
Memory as a programming concept in C and C++ / Frantisek Franek.
p. cm.
Includes bibliographical references and index.
ISBN 0-521-81720-X – ISBN 0-521-52043-6 (pb.)
1. Memory management (Computer science) 2. C (Computer program language)
3. C++ (Computer program language) I. Title.

QA76.9.M45F73 2003
005.4'35 – dc21 2003051543

ISBN 0 521 81720 X hardback
ISBN 0 521 52043 6 paperback

CONTENTS

CHAPTER ONE

INTRODUCTION

The motivation for this book came from years of observing computer science students at universities as well as professional programmers working in software development. I had come to the conclusion that there seemed to be a gap in their understanding of programming. They usually understood the syntax of the programming language they were using and had a reasonable grasp of such topics as algorithms and data structures. However, a program is not executed in a vacuum; it is executed in computer memory. This simple fact exerts a powerful influence on the actual behavior of the program – or, expressed more precisely, a subtle yet powerful influence on the semantics of the particular programming language. I had observed that many students and programmers did not fully understand how memory affected the behavior of the C and C++ programs they were designing. This book is an attempt to fill this gap and provide students and programmers alike with a text that is focused on this topic.

In a typical computer science curriculum, it is expected that students take courses in computer architecture, operating systems, compilers, and principles of programming languages – courses that should provide them with a "model" of how memory matters in the behavior of programs.

However, not all students end up taking all these courses, and even if they do, they may not take them in the right order. Often the courses are presented in a disjointed way, making it difficult for students to forge a unified view of how memory affects the execution of programs. Additionally, not all programmers are graduates of university or college programs that feature a typical computer science curriculum. Whatever the reasons, there seems to be a significant number of computer science students and professional programmers who lack a full understanding of the intricate relationship between programs and memory. In this book we will try to pull together the various pieces of knowledge related to the topic from all the fields involved (operating systems, computer architecture, compilers, principles of programming languages, and C and C++ programming) into a coherent picture. This should free the reader from searching various texts for relevant information. However, in no way should this book be viewed as a programming text, for it assumes that the reader has at least an intermediate level of programming skills in C or C++ and hence simple programming concepts are not explained. Nor should this book be viewed as an advanced C/C++ programming text, for it leaves too many topics – the ones not directly related to memory – uncovered (e.g., virtual methods and dynamic binding in C++). Moreover, it should not be seen as an operating system book, for it does not delve into the general issues of the discipline and only refers to facts that are relevant to C and C++ programmers.

Unfortunately, there seems to be no curriculum at any university or college covering this topic on its own. As a result, students usually end up with three or four disjointed views: programming syntax and (an incomplete) C/C++ semantics; algorithms and data structures, with their emphasis on the mathematical treatment of the subject; operating systems; and possibly compilers. Although my ambition is to fill the gaps among these various views – at least from the perspective of C/C++ programming – I hope that the book proves to be a valuable supplement to any of the topics mentioned.

My own experience with software development in the real world shows that an overwhelming number of computer program bugs and problems are related to memory in some way. This is not so surprising, since there are in fact few ways to "crash" a program and most involve memory. For instance, a common problem in C/C++ is accessing an array item with an index that is out of range (see Chapter 6). A program with such a simple bug can exhibit totally erratic behavior during different executions,

behavior that ranges from perfect to incorrect, to crashing at the execution of an unrelated instruction with an unrelated message from the operating system, to crashing at the execution of the offending instruction with a message from the operating system that signals an invalid memory access.

With the advent of object oriented programming and the design and development of more complex software systems, a peculiar problem has started to manifest itself more frequently: so-called memory leaks (see Chapter 10). In simple terms, this is a failure to design adequate housecleaning facilities for a program, with the result that unneeded earlier allocated memory is not deallocated. Such undeallocated and ultimately unused memory keeps accumulating to the point of paralyzing the execution of the program or the performance of the whole computer system. It sounds almost mystical when a programmer's explanation of why the system performs so badly is "we are dealing with memory leaks", as if it were some kind of deficiency of the memory. A more concrete (and accurate) explanation would be "we did not design the system properly, so the unneeded but undeallocated memory accumulates to the point of severely degrading the performance of the system". The troubles that I have witnessed in detecting and rectifying memory leaks strongly indicate that many students and programmers lack a fundamental appreciation of the role and function of memory in programming and program behavior.

We are not really interested in technical, physical, or engineering characteristics of memory as such (how it is organized, what the machine word is, how the access is organized, how it is implemented on the physical level, etc.); rather, we are interested in memory as a concept and the role it plays in programming and behavior of C/C++ programs. After finishing this book, the reader should – in addition to recognizing superficial differences in syntax and use – be able to understand (for example) the deeper differences between the "compile-time index range checking" philosophy used in C/C++ and the "run-time index range checking" philosophy used in Pascal (Chapter 6) or between the "recursive procedure calls" philosophy used in C/C++ and the "nonrecursive procedure calls" philosophy used in FORTRAN (Chapter 5). As another example, the reader of this book should come to appreciate why Java requires garbage collection whereas C/C++ does not (and in general cannot); why C/C++ cannot be interpreted in a manner similar to Java; and why Java does not (and cannot) have pointers whereas C/C++ does (Chapter 3) – because

all these aspects are related in some way to memory and its use. The reader should understand the issues concerning memory during object construction and destruction (Chapter 8); learn how to compact or serialize linked data structures so they can be recorded to a disk or transmitted across a network (Chapter 9); and learn how to design programs that allow monitoring of memory allocation/deallocation to detect memory leaks (Chapter 10). The reader will also be exposed to important concepts not exclusively related to C/C++, concepts that are usually covered in courses on operating systems but included here by virtue of being related to memory: for example, concepts of process and thread and interprocess communication (Chapter 11) facilitated by memory (shared memory segments, pipes, messages). Of course, as always, our interest will be on the memory issues concerning both the processes and the threads.

The book is divided into eleven chapters. Chapter 2 deals with the process of compilation, linking, and loading in order to explain how the behavior of programs can be discussed and examined as if they were executing in the source form, how the static and the dynamic parts of memory are assigned to a program, and how the abstract address space of the program is mapped to the physical memory. Most of the topics in Chapter 2 are drawn from the field of the principles of operating systems. We cover the topics without referring to any particular operating system or any low-level technical details. Otherwise, the text would become cumbersome and difficult to read and would distract the reader from focusing on memory and its role in C/C++ programming. However, knowledge of the topics covered in Chapter 2 is essential to almost all discussions of the role of memory in the subsequent chapters.

Chapter 3 deals with variables as memory segments (data containers) and the related notions of addresses and pointers, with a particular emphasis on various interpretations of the contents of memory segments and possible memory access errors. In Chapter 4, dynamic memory allocation and deallocation are discussed and illustrated using the C allocators `malloc()`, `calloc()`, and `realloc()` and the C deallocator `free()`. In Chapter 5, function calls are explained with a detailed look at activation frames, the system stack, and the related notion of recursion. In Chapter 6, one-dimensional arrays and strings, both static and dynamic, are discussed. Chapter 7 extends that discussion to multi-dimensional arrays.

Chapter 8 examines in detail the construction and destruction of C++ objects together with the C++ allocators (the operators `new` and `new[]`) and the C++ deallocators (the operators `delete` and `delete[]`) in their

global and class-specific forms. The focus of the chapter is not the object orientation of C++ classes but rather the aspects of object creation and destruction related to memory. Similarly, in Chapter 9 we discuss linked data structures but not from the usual point of view (i.e., their definition, behavior, implementation, and applications); instead, our point of view is related to memory (i.e., how to move linked data structures in memory, to or from a disk, or across a communication channel). Chapter 10 is devoted to a classification of the most frequent problems leading to memory leaks and their detection and remedy for both C and C++ programs.

We started our discussion with operating system topics related to programs – compilation, linking, and loading – in Chapter 2, and in Chapter 11 we finish our book by again discussing operating system topics related to programs in execution: processes and threads, and how they relate to memory. Of course, this chapter must be more operating system–specific, so some notions (e.g., the system call `fork()` and the sample code) are specific to UNIX.

Finally, in the appendices we present some complete code and discuss it briefly. In Appendix A we describe the Hanoi towers puzzle and provide a simple C program solving it (for completeness, as the puzzle is mentioned in Chapter 5 in relation to recursion). In Appendix B we present a simple C++ program on which we illustrate object tracing: how to keep track of objects and of when and where they were allocated (this includes localization tracing as well). We go through various combinations of turning the features on and off. In Appendix C, a similar C++ program is used and object tracing, localization tracing, and memory allocation tracing are all demonstrated. Appendix B and Appendix C both illustrate debugging of memory leaks as discussed in Chapter 10. Finally, Appendix D contains process-safe and thread-safe UNIX logging functions (used in examples throughout the book) that serve to illustrate some of the topics related to processes and threads discussed in Chapter 11.

Every chapter includes a Review section that contains a brief and condensed description of the topics covered, followed by an Exercises section that tests whether the reader has fully grasped the issues discussed. This is followed by a References section, pointing the reader to sources for examining the issues in more depth. All special terms used in the book are defined and/or explained in the Glossary, which follows Appendix D.

I have tried to limit the sample computer code to the minimum needed to comprehend the issues being illustrated, leaving out any code not relevant to the topic under discussion. Hence some of the fragments of code

within a given chapter are not complete, though all were tested within larger programs for their correctness.

I wish you, dear reader, happy reading, and I hope that if somebody asks you about it later you can reply: "if my *memory* serves, it was a rather useful book".