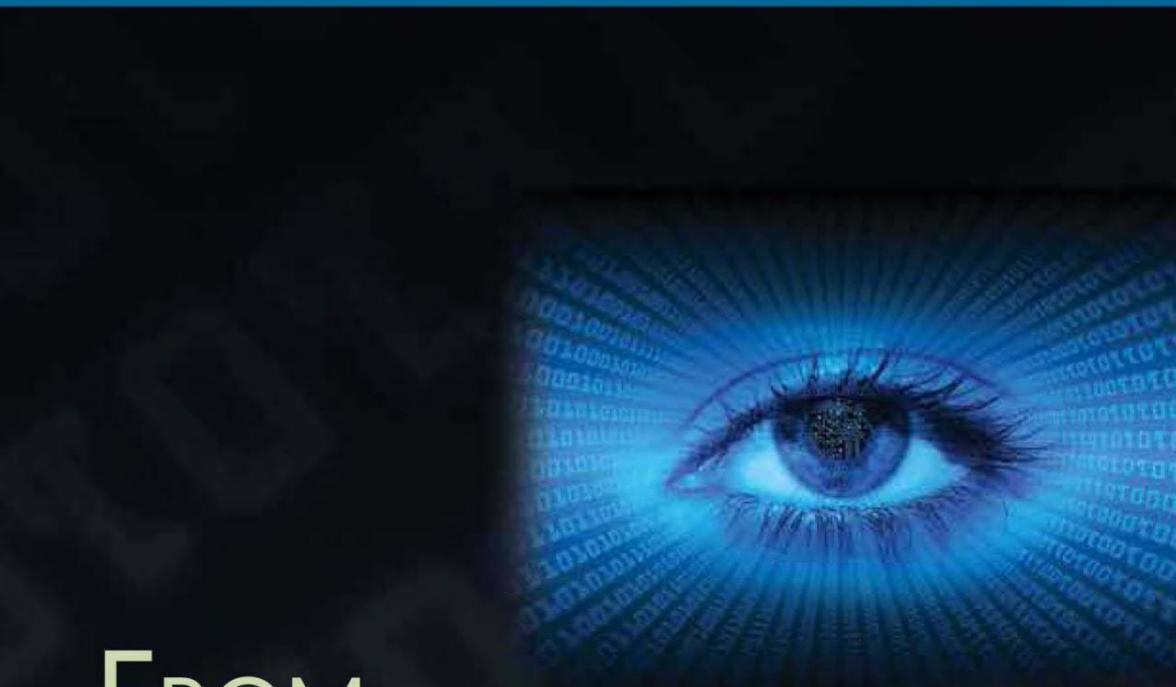
Paul S. Wang



From Computing to Computational Thinking

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

FROM COMPUTING TO COMPUTATIONAL THINKING

Paul S. Wang
Kent State University
Ohio, USA

CRC Press
Taylor & Francis Group
Boca Raton London New York

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

A CHAPMAN & HALL BOOK

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

Version Date: 20151204

International Standard Book Number-13: 978-1-4822-1766-7 (eBook - PDF)

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

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

Contents

Preface

It has been widely recognized that concepts, techniques, and analytical abilities from the field of computing can be powerful mental tools in general for solving problems, performing tasks, planning, working with others, anticipating problems, troubleshooting, and more. We refer to this mental tool set as *computational thinking* (CT).

This textbook will help readers acquire computational thinking through an understanding of modern computer technologies. Neither programming background nor learning how to program is required. Students just need to bring their curiosity and an open mind to class.

Reading this book can be an excellent way to prepare someone to pursue a rewarding career in computing or information technology. The materials are as much about computing as about sharpening the mind.

Topics and Presentation

The book has an end-user viewpoint. Topics are presented in an interesting and thought-provoking way, keeping the reader engaged and motivated to continue.

Unconventional chapter titles, CT callout boxes, relation to daily living, and connection to well-known events combine to encourage computational thinking and instill agile mental skills. In addition, we introduce a new verb in English, *computize*. To computize is to apply CT. With a little bit of help, everyone can do it.

The CT callout boxes highlight nuggets of computational thinking wisdom worth revisiting from time to time. They can be found easily in the Table of Contents and in the Index.

The user is guided through a well-selected set of topics covering the type of material appropriate for a one-semester course at the college freshman level for students from all different majors. Advanced programs in high schools, and the public in general, may find this book useful and rewarding as well.

Computing and CT

Understanding computing and acquiring CT are two sides of the same coin. By learning about hardware, software, networking, the operating system, security

measures, the Web, digital data, apps, and programming paradigms, we gain valuable knowledge to better take advantage of information technologies.

At the same time, concepts and methods from computing form elements of CT that are applicable outside of computing. CT can make us wiser and more effective in countless ways. CT can help us avoid accidents and mishaps. It can even be life saving.

Chapter-end exercises reinforce topics in each chapter and challenge students to apply CT (to computize) in various situations. Group discussions are encouraged as well.

The CT Website

Throughout the book, concepts, techniques, and technologies are explained with many interesting examples. Hands-on demos for experimentation are online at the book's companion website `http://computize.org`

The site is mobile-enabled and works on both regular and mobile devices. In the text, we refer to it as *the CT website*. The live demos are cross-referenced to in-text descriptions with a notation such as **Ex:**`UpCounter` that also appears in the book's index.

The CT website offers additional resources, allows you and others to share insights on CT, and provides information updates.

Acknowledgments

I thank my dear wife, Jennifer Wang (葛孝薇), who encouraged me to embark on this project, read all drafts, and provided great feedback, sometimes with specific ideas and wording changes. I am very grateful to her.

In fact, she was the one who asked me to listen to a broadcast of the *Kojo Nnamdi Show* (WAMU/NPR) on November 18, 2008. The show started with an interview of Dr. Jeannette Wing on the topic

"*Thinking like a computer scientist.*"

Jennifer was so excited and told me over the phone, "You need to listen to this right now, it is what you always talked about."

Encouraged by Dr. Wing's advocacy, I soon contacted her at the National Science Foundation and invited her to visit our Computer Science Department

at Kent State University to give a talk on CT. The face-to-face interactions with Dr. Wing further convinced me to make a contribution in this important direction. The influence of NPR and Dr. Wing on me cannot be overstated.

Thanks must also go to my children Laura, Deborah, and David, who took an active interest in this book; especially David who read first chapters of an early draft and made comments that influenced the choice of the book title.

I also thank Randi Cohen, editor at CRC Press, who supported this project with energy and enthusiasm from the very beginning. She also helped to choose the book title.

Deep thanks also go to production coordinaters, Kathryn Everett and Amber Donley as well as project editor Robin Lloyd-Starkes, and others at CRC Press for their professionalism and dedication.

During the planning and writing of this book, several reviews have been conducted. Much appreciated are the input and suggestions from the reviewers:

- Iyad A. Ajwa, Ashland University, Ohio, USA
- Lian Li (李廉教授) and his team, HeFei University of Technology, Anhui, China
- Alex Melton, Benjamin Logan High School, Ohio, USA
- Anonymous reviewers

Paul S. Wang
王士弘
Kent, Ohio
pwang@cs.kent.edu

Introduction

Digital computers brought us the information revolution. Citizens in the information age must deal with computers, smartphones, and the Internet. In addition, they also need to gain *computational thinking*.

Computational thinking (CT) is the mental skill to apply fundamental concepts and reasoning, derived from modern digital computers and computer science, in all areas, including day-to-day activities. CT is thinking inspired by an understanding of computers and information technologies, and the advantages, limitations, and problems they bring. CT also encourages us to keep asking questions such as "*What if we automate this?*" "*What instructions and precautions would we need if we were asking young children to do this?*" "*How efficient is this?*" and "*What can go wrong with this?*"

CT can expand your mind, help you solve problems, increase efficiency, avoid mistakes, and anticipate pitfalls, as well as interact and communicate better with others, people or machines. CT can make you more successful and even save lives!

It is not necessary to become a computer scientist or engineer for you to acquire CT. From a user point of view, we present a well-organized sequence of topics to introduce computers and computing in simple and easy ways, assuming little prior knowledge of computer science or programming. While presenting the hardware, software, data representation, algorithm, systems, security, networking, the Web, and other aspects of computing, we will highlight widely applicable concepts and mental skills in *CT call-out boxes* and explain how/where they can be applied in real life.

Background

Back in March 2006, Dr. Jeannette M. Wing published an article on computational thinking in *The Communications of ACM* and boldly advocated it as a skill for everyone:

> "Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. ... Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading,

> writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking."

Within the academic research community, there have been significant discussions on computational thinking, what it encompasses, and its role inside the education system.

In educational circles, there is an increasing realization of the potential importance of learning to think computationally. According to a recent report on computational thinking by the National Research Council of The National Academies (NRC):

> "... Computational thinking is a fundamental analytical skill that everyone, not just computer scientists, can use to help solve problems, design systems, and understand human behavior. ... Computational thinking is likely to benefit not only other scientists but also everyone else. ..."

The ACM/IEEE-CS Joint Task Force on Curriculum recently (2013) stated

> "Computational Thinking—While there has been a great deal of discussion in regard to computational thinking, its direct impact on curriculum is still unclear. While we believe there is no 'right answer' here, CS 2013 seeks to gain more clarity regarding models by which CS curricula can promote computational thinking for broader audiences."

Discovering the Secrets of CT

Computers are dumb. They deal only with *bits*. Each bit represents either a zero or a one. They blindly follow program instructions and operate on data, both being represented by sequences of `0`s and `1`s. Yet, they are universal machines that can perform any tasks when given instructions. The ways they are programmed, controlled, and made to work are fascinating to learn by themselves. But such understanding has more to give us, namely, computational thinking.

Important aspects of CT include

- Simplification through abstraction—Abstraction is a technique to reduce complexity by ignoring unimportant details and focusing on what matters. For example, a driver views a car in terms of how to drive it and ignores how it works or is built. A user cares only about which mouse

button to click and keys to press and generally overlooks how computers work internally.

- Power of automation—Arranging matters so they become routine and easy to automate. Working out a systematic procedure, an algorithm, for carrying out recurring tasks can significantly increase efficiency and productivity.

- Iteration and recursion—Ingeniously reapplying the same successful techniques and repeatedly executing the same set of steps to solve problems.

- An eye and a mind for details—Changing a 0 to a 1, or an upper-case `O`, can mess up the whole program. You need eyes of an eagle, mind of a detective, and a careful and meticulous approach. Overlooking anything can and will lead to failure.

- Precision in communication—Try telling the computer to do what you mean and not what you say `;-)`. You need to spell it out precisely and completely. Don't spare any details. Vagueness is not tolerated. And contexts must be made explicit.

- Logical deductions—"Cold logic" rules. Causes will result in consequences, whether you like it or not. There is no room for wishful or emotional thinking.

- Breaking out of the box—A computer program executes code to achieve any task. Unlike humans, especially experts, it does not bring experience or expertise to bear. Coding a solution forces us to think at a dumb computer's level (as if talking to a one-year-old) and get down to basics. This way, we will naturally need to think outside any "boxes."

- Anticipating problems—Automation relies on preset conditions. All possible exceptions must be met with prearranged contingencies. Ever said "I'll take care of that later"? Because there is a chance you might forget, according to CT, you should have a contingency plan ready in case you do forget. Otherwise, you have set a trap for yourself.

These are just some of the main ideas. CT offers you many more concepts and ways to think that can be just as, if not more, important. With increasing understanding of computing, one begins a process of gaining CT insights from many angles and viewpoints. Such CT takeaways can vary from person to person, in terms of what they are and their significance.

Here is a chicken and egg question. Which comes first, computing or computational thinking? Surely, ideas and techniques, from other disciplines as well as the long history of human civilization, have contributed to the development, breakthroughs, and refinements in computing. Yet, computer science has also generated many unique concepts, techniques, and problem-solving

ideas. Computing has given rise to a digital ecosystem, called *cyberspace*, that includes us all.

Understanding the digital computer and computation is beneficial in itself. Plus, it gives us a very efficient way to discover/rediscover a set of powerful ideas, collectively known as CT, that can be applied widely. As we gain more understanding of computing and its various aspects, we will raise CT ideas along the way. Repeat visits of CT concepts from different aspects and contexts of computing provide different viewpoints and help instill the concepts, their usefulness and generality, in our minds.

This textbook provides an interesting and thought-provoking way to gain general knowledge about modern computing. You'll be exposed to new notions and perspectives that not only enrich your thinking but also make you more successful. For example, without the notion of *germs*, people won't achieve proper hygiene practices or effectively prevent disease transmission. Similarly, without a general understanding of computing concepts, it is hard to become a full-fledged citizen of the digital age.

Taking ideas from one field and applying them in another is not new. In fact, many breakthroughs came from such interdisciplinary endeavors. For example, the new biomimicry science studies nature's models and then applies these designs, processes, and inspirations to solve our own problems.

Readers are encouraged to share their own views, insights, and inspirations on the CT website. How wonderful that we can use computing technology to join forces and help advance CT.

Computize

Definition: **computize**, verb. To apply computational thinking. To view, consider, analyze, design, plan, work, and solve problems from a computational perspective.

When considering, analyzing, designing, formulating, or devising a solution/answer to some specific problem, computizing becomes an important additional dimension of deliberation.

People say "hindsight is 20/20." But, since computer automation must deal with all possible applications in the future, we must ask "what if" questions

sequences of steps, to refine the solution logic, and to indicate how to handle different possibilities. Figure 1.9 shows a simple flowchart for the task of "getting up in the morning." We begin at the `Start` and follow the arrows to each

FIGURE 1.9 A Simple Flow Chart

next step. In programming, the flow from step to step is called the *control flow*. A diamond shape is used to indicate a fork in the path. Which way to turn depends on the conditions indicated. Obviously, we use diamond shapes to anticipate possibilities. The snooze option leads to a branch that repeats some steps. In programming, such a group of repeating steps is called a *loop*. The procedure ends when the person finally climbs out of bed.

As another example, let's look at a flowchart for troubleshooting a lamp (Figure 1.10). The very first step after `start` is significant. Although the

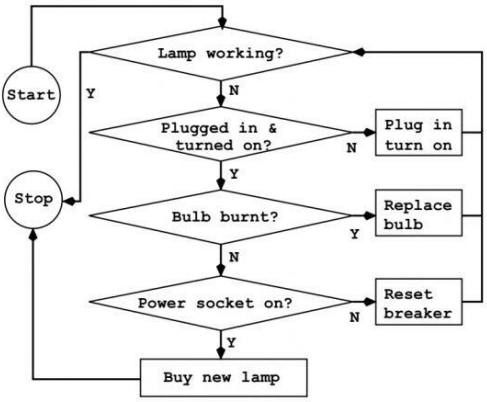


FIGURE 1.10 Lamp Fixing

purpose of the procedure is to troubleshoot a lamp, we, nonetheless, make no implicit assumption that the lamp is not working. Without this step at

the beginning, the procedure would potentially troubleshoot a perfectly good lamp, and, worse yet, would decide to replace it with a new lamp!

CT: Ready for All Contingencies

When executing a task, be prepared for all contingencies every step of the way.

Each of the next three steps tests for a particular problem and makes a fix. Then the same procedure is reiterated by going back to step one to determine if the lamp is now working. This further demonstrates the importance of step one.

Steps 2, 3, and 4 are ordered according to their probabilities. That is, we check the most common problem first. It makes the procedure more efficient. For this procedure, it still works if we take these three steps in a different order. However, in general, step sequencing is important, and changing the order may break the procedure.

CT: First Things First

Perform tasks in the correct order. Avoid putting the cart in front of the horse.

Our third example flowchart (Figure 1.11) outlines a plan for a computer program that receives a series of words on a line and echos back the given words in reverse order. The word count `n` is a variable that is set to the number of words given. If no words are given, a contingency not to be ignored, or remain (i.e., `n` is zero), the program ends by displaying a NEWLINE character. Otherwise, it displays the `n`th word, sets the new word count to `n-1`, and loops back to test the value of `n`. Because `n` is reduced by 1 with each iteration, the procedure eventually will end.

CT: Check before Proceeding

Always check before proceeding or repeating a task or a sequence of steps.

Flowcharting is very useful in activity planing and working out systematic procedures for tasks. Try it yourself at the CT site (**Demo:** `DoFlowchart`).

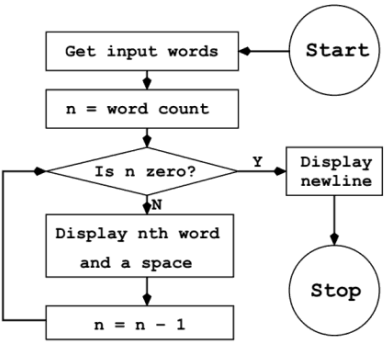


FIGURE 1.11 Reverse Echo

1.8 Algorithms

The origin of the word "algorithm" traces back to the surname Al-Khwārizmī of a Persian mathematician (780–850 CE), who was well-known for his work on algebra and arithmetic with Indian numbers (now known as Arabic numbers). The modern-day meaning of algorithm in mathematics and computer science relates to an effective step-by-step procedure to solve a given class of problems or to perform certain tasks or computations.

It is generally agreed that a procedure is an algorithm if it can be carried out successfully by a Turing machine (Section 1.2). Specifically, a procedure becomes an algorithm if it satisfies all of the following criteria[3]:

- Finiteness: The procedure consists of a finite number of steps and will always terminate in finite time.
- Definiteness: Each step is precisely, rigorously, and unambiguously specified.
- Input: The procedure receives a set of data, possibly empty, provided to the procedure before it starts. Possible values for the data may vary within limitations.
- Output: The procedure produces a nonempty set of results.
- Effectiveness: Each operation in the procedure is basic and clearly doable.

[3]Following criteria given by D. Knuth in his book, *The Art of Computer Programming, Vol. I.*

9.8.2 Huffman Code

In a particular message or type of messages, the frequencies of different symbols are certainly not equal. Some symbols happen more often than others and certain symbols hardly at all. For example, in a textual document, you would expect the letters `r`, `s`, `t`, as well as the vowels, to be more frequent than, say, `x`, `y`, and `z`; and characters such as `^` hardly at all. We can take advantage of such frequency differences to save space in representing characters. The basic idea of Huffman code is simple. Instead of using the same number of bits to represent each character (as in ASCII and UNICODE), we can use fewer bits for frequent characters and more bits for other characters.

The same goes for numbers. Instead of using, say, a 32-bit integer representation, we can use just a few bits for high-frequency numbers. In practice, you'll find small numbers much more often.

To Huffman encode a message, we first find the frequencies of symbols to be encoded. Based on the frequencies, we can build a *Huffman binary tree* (CT: Form Tree Structures, Section 8.8), which defines how these symbols will be encoded into bit strings of various lengths. The Huffman tree is stored as part of the deflated message because it is needed for inflation.

As an example, let's apply Huffman coding to deflate a love poem by Nima Akbari (**Demo:** `HuffCode`), which begins

> You're my man, my mighty king,
> And I'm the jewel in your crown,
> You're the sun so hot and bright,
> I'm your light-rays shining down,
> . . .

The plaintext file for the whole poem contains 545 bytes, including spaces and line breaks. The character frequencies are shown here in increasing frequency.

```
N 1     - 1     . 1     W 1     p 1     j 2
v 3     A 5     f 5     k 5     c 6     b 7
Y 7     I 8     w 8     l 11    g 15    ' 15
s 15    d 17    y 17    , 17    m 18    t 20
u 20    CR 21   a 22    h 24    i 26    o 30
r 30    n 38    e 39    SP 89
```

Note characters include uppercase and lowercase letters and punctuation marks, including SPACE (`SP`), and RETURN (`CR`).

A list, `fq`, of character-frequency pairs, ordered in increasing frequency, such as the above, is used in a simple recursive algorithm to build a *Huffman tree*. Basically, the algorithm builds a binary tree from the bottom up, by creating tree nodes with characters having the least frequencies first. Each recursive call `huffmanTree(fq)` passes a list `fq` that is smaller in size.