Introduction

A New Kind of Problem-Solving

It's a lazy summer afternoon in 1450, and a tired monk is sitting at a desk, staring at a blank sheet of vellum. He's been given the task of making twenty copies (twenty!) of a fifty-page book. He sighs, then picks up a pen and begins to write, following the holes that have been carefully pricked into the sheet as guides. He wonders if he's being punished. This will take forever!

Figure 1: A monk copying a manuscript.
Source: Wikimedia Commons

As he works, his mind wanders into fantasies of being an Abbott or a King, capable of commanding monks to do all the menial work. He'd only have to command twenty copies of a book (or a thousand!) and it would be done. Even better to be a Wizard, and not have to deal with lazy monks! Swoosh! goes the magic wand, and a pile of books appears!

The monk doesn't know it, but his vision is becoming reality even as he works. A few years earlier, Johannes Gutenberg had invented a printing press that used moveable type. As it spread across Europe, this new technology was changing the way people thought about problem-solving.

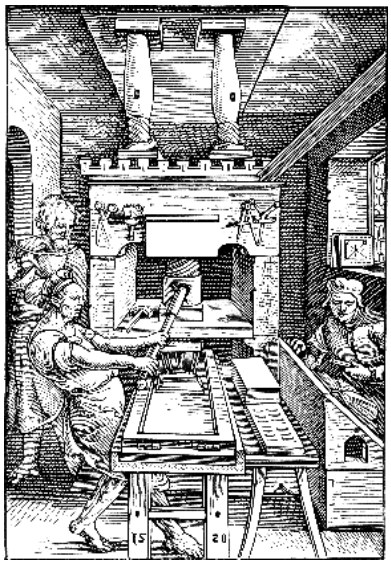

Figure 2: A printing press (1520).
Source: Wikimedia Commons

For the monk in his scriptorium, each new page is a new problem requiring an amount of time and effort similar to any previous page. To copy fifty pages takes him about fifty times as long as a single page. Even though he might begin the task by spending a little time thinking about the style of the writing and the layout of the pages, the vast majority of his time will be spent on the mindless, repetitive task of producing individual pages, one at a time. If his mind wanders into fantasies, a page could be ruined.

But consider the job of a printer a hundred years later. To him, the problem of printing a page consists of setting the type. Once he's done that, he can create as many copies of the page as he likes, with relatively little effort and in a short time.

* * *

Early 20^{th} Century particle physicists used "cloud chambers" and, later, "bubble chambers" to see the paths of subatomic particles. Collisions and decays within these chambers produced visible tracks that could be photographed. The chambers could take a new photograph every few seconds. Each photograph was then analyzed by people called "scanners", who measured the tracks as the photographs were projected onto a table. At their fastest these workers could analyze only about five photographs per hour. Photographs taken during a few days of running a bubble chamber could take years to analyze.

Bubble chambers have long been superseded by other kinds of detectors that can be read out electronically and analyzed by computers. Because of this, large experiments like the Compact Muon Solenoid at CERN can record and analyze thousands of electronic "snapshots" per second. There are no longer any "scanners", just as monks no longer copy manuscripts.

* * *

Since the earliest days of aeronautics, airplane designs have been tested in wind tunnels. The Wright brothers themselves used a simple wind tunnel in the development of the "Wright Flyer". Whole airplanes, parts of them, or models of them were placed into the wind tunnel to study their behavior. The lift generated by one type of wing or propeller might be measured and compared to measured values for other designs. Many models were made and tested in the process of designing an airplane.

Today, computer simulations have largely replaced wind tunnel tests. Modern computational fluid dynamics can accurately model the flow of air around complicated shapes, and we can change the shape by clicking and dragging a mouse or changing some parameters, rather than needing to manufacture a physical model, leaving the engineer free to test odd shapes and explore possibilities as they occur to her.

* * *

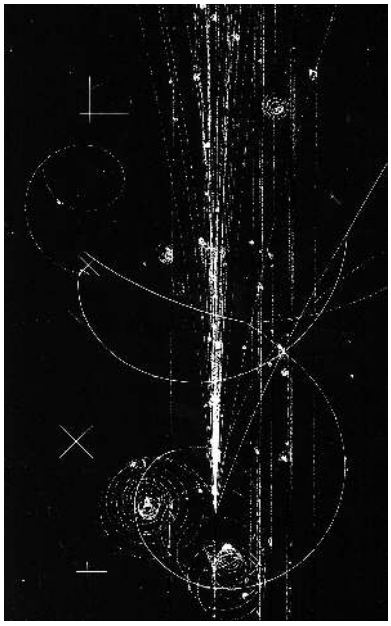

Figure 3: Traces of charged particles in a bubble chamber at Fermilab (1973).
Source: Wikimedia Commons

Figure 4: A "scanner" analyzes a bubble chamber photograph.
Source: CERN

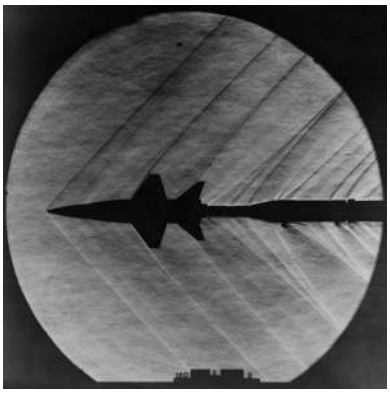

Figure 5: A model of the X-15 rocket plane in a wind tunnel (1962).
Source: Wikimedia Commons

In 1913 Henry Norris Russell documented a relationship between the color and brightness of stars. At that time, and indeed until the 1970s, most graphs used in publications were drawn by hand. On the left-hand side of the figure below you can see Russell's graph of brightness versus color (what we now call a Hertzsprung-Russell diagram). The graph shows data for about 300 stars, collected by observers using astronomical instruments and written down by hand. These data were then plotted, using pen and ink, to show the results.

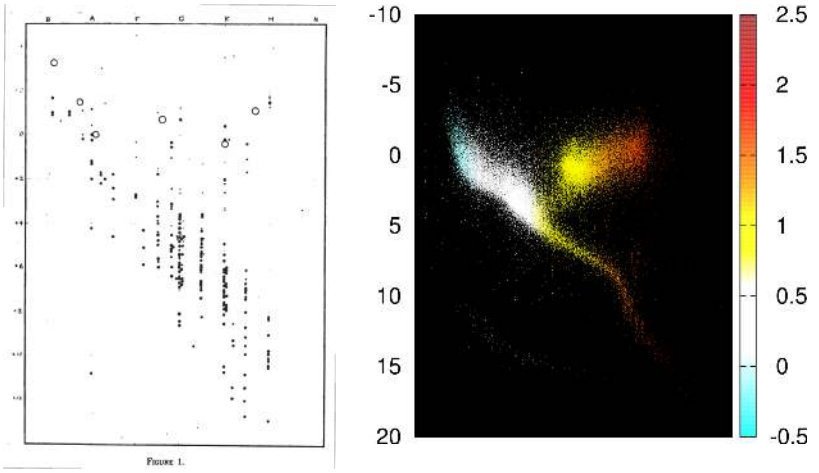

Figure 6: Russell's original diagram, and a modern Hertzsprung-Russell diagram produced with *gnuplot* using data from the Hipparcos satellite.

Source: Popular Astronomy. 22: 275-294, 1914

On the right-hand side of the figure above we see a modern-day Hertzsprung-Russell diagram. It was produced using data gathered by the Hipparcos satellite, downloaded over the Web, analyzed by a computer program, and plotted using *gnuplot*. It shows about 100,000 stars. It took the computer less than a second to produce this graph from the data.

* * *

The computer revolution of the late 20^{th} Century gave us a new kind of problem-solving. As in the aftermath of the Gutenberg revolution, we suddenly found that we no longer needed to focus on the mindless, repetitive components of many tasks. Computers could now make data analysis more-or-less effortless. Simulations done by computers were now capable of eliminating the need for many real-world tests. Visualizations that were once tedious to prepare could now be done instantly, by anybody. The ease, accuracy, and speed with which computers could perform repetitive tasks freed us up to explore in ways that would have been unfeasible earlier.

To a poor monk in a scriptorium every page is a new problem that needs

to be solved. To a printer, once the page is typeset the problem is solved forever. A well-written computer program does the same. It tackles a problem, and solves it *forever*. That's a new kind of problem-solving.

About this Book

Today, if you intend to pursue a career in science or engineering you'll need to know the basics of computation. This book aims to teach them to you.

It introduces three core skills: analyzing data, simulating data, and visualizing data. It assumes no prior programming experience or knowledge about the inner workings of computers. It will concentrate on using using computers to solve common problems you'll encounter in science and engineering.

A Note About Choices

Which is the best tool: a hammer or a screwdriver? Most people would say that the answer depends on the task. The same is true for computer languages. There is no "best" programming language, any more than there's a best tool.

When designing this book, I needed to choose a programming language that would suit its needs and yours. I settled on the C language for several reasons.

First of all, C and its cousins (C++, Objective-C, *etc cetera*) are very widely used. It's likely that any program you've ever used on a desktop computer was written in some variant of the C language. A 2016 study by IEEE[1] ranked C as the most popular programming language, based on its use in software repositories and appearance as a topic in various online forums.

[1] IEEE Spectrum: The Top Programming Languages 2016

C has been around a long time, and many newer programming languages have adopted features from it. This means that once you've learned C you'll find it easier to learn those languages, too. Some of these C-like languages include Java, PHP, Javascript, Perl, Go, and C#.

More than some languages, C lets you see the computer's internal workings. When learning C, you need to think about the way the computer uses memory to store information, and how data is stored in

files. An understanding of these concepts will help you later on, even if you move to higher-level programming languages that hide these details.

C has a reputation for being fast. Other languages sometimes rely on C to do their "heavy lifting". For example, Google recently released an artificial intelligence system named TensorFlow[2], which appears to be written in the Python programming language. If you download TensorFlow and look at the source code, though, you'll find that about 80% of it is written in C. The Google developers said they wrote the most compute-intensive parts of the code in C to make it run faster. If you go into research or engineering, you'll often be working at the cutting edge of technology. Having the skill to write C programs can help you squeeze the best performance out of your software.

[2] https://www.tensorflow.org/

Finally C is available on a wider range of computers than any other language, and the software needed to build C programs is available for free. No matter what kind of computer you're using, or how small your budget, It's almost certain that you'll be able to write and run C programs.

Figure 7: Dennis Ritchie, the inventor of the C language.
Source: Wikimedia Commons

Those are some of the reasons for choosing to use the C language in this book. Every language has its strengths and weaknesses. After you've learned C, I hope you go on to explore other languages too. When you're a researcher or an engineer, here are some other things you should think about when deciding which language to use for a project:

- What are your skills? Sometimes its better to use a language you already know.
- What are the skills of other programmers who are likely to work on this project in the future? When you're collaborating with other programmers, consider their skills, too.
- If there's an existing code base, what language(s) does it use? When adding features to existing software, it's often a good idea to stick to the same language the rest of the software uses, unless there's a compelling reason to introduce a new language.
- Are strengths of a given programming language a good match for the project's needs? Don't try to use a hammer to insert screws.