ARLIZ

A JOURNEY THROUGH ARRAYS

Mahdi

In Praise of Arliz

MAHDI

This book evolves. Every insight gainedwhether a circuit, a structure, or a simple ideais absorbed and integrated.

FIRST EDITION

May 29, 2025

To those who build from first principles.
To the silent thinkers who design before they speak.
To the ones who see in systems
not just machines, but metaphors.
This is for you.

Preface

Every book has its own story, and this book is no exception. If I were to summarize the process of creating this book in one word, that word would be improvised. Yet the truth is that Arliz is the result of pure, persistent curiosity that has grown in my mind for years. What you are reading now could be called a technical book, a collection of personal notes, or even a journal of unanswered questions and curiosities. But Ioffi-ciallycall it a *book*, because it is written not only for others but for myself, as a record of my learning journey and an effort to understand more precisely the concepts that once seemed obscure and, at times, frustrating.

The story of Arliz began with a simple feeling: **curiosity**. Curiosity about what an array truly is. Perhaps for many this question seems trivial, but for me this worden-countered again and again in algorithm and data structure discussionsalways raised a persistent question.

Every time I saw terms like `array`, `stack`, `queue`, `linked list`, `hash table`, or `heap`, I not only felt confused but sensed that something fundamental was missing. It was as if a key piece of the puzzle had been left out. The first brief, straightforward explanations I found in various sources never sufficed; they assumed you already knew exactly what an array is and why you should use it. But I was looking for the *roots*. I wanted to understand from zero what an array means, how it was born, and what hidden capacities it holds.

That realization led me to decide: *If I truly want to understand, I must start from zero.*

There is no deeper story behind the name Arliz. There is no hidden philosophy or special inspirationjust a random choice. I simply declared: *This book is called Arliz.* You may pronounce it "Ar-liz," "Array-Liz," or any way you like. I personally say "ar-liz." That is allsimple and arbitrary.

But Arliz is not merely a technical book on data structures. In fact, **Arliz grows along-side me**.

Whenever I learn something I deem worth writing, I add it to this book. Whenever I feel a section could be explained better or more precisely, I revise it. Whenever a new idea strikes mean algorithm, an exercise, or even a simple diagram to clarify a struc-

tureI incorporate it into Arliz.
This means Arliz is a living project. As long as I keep learning, Arliz will remain alive.
The structure of this book has evolved around a simple belief: true understanding begins with context. Thats why Arliz doesnt start with code or syntax, but with the origins of computation itself. We begin with the earliest tools and ideascounting stones, the abacus, mechanical gears, and early notions of logiclong before transistors or binary digits came into play. From there, we follow the evolution of computing: from ancient methods of calculation to vacuum tubes and silicon chips, from Babbages Analytical Engine to the modern microprocessor. Along this journey, we discover that concepts like arrays arent recent inventionsthey are the culmination of centuries of thought about how to structure, store, and process information.
In writing this book, I have always tried to follow three principles:

- **Simplicity of Expression:** I strive to present concepts in the simplest form possible, so they are accessible to beginners and not superficial or tedious for experienced readers.
- **Concept Visualization:** I use diagrams, figures, and visual examples to explain ideas that are hard to imagine, because I believe visual understanding has great staying power.
- **Clear Code and Pseudocode:** Nearly every topic is accompanied by code that can be easily translated into major languages like C++, Java, or C#, aiming for both clarity and practicality.

An important note: many of the algorithms in Arliz are implemented by myself. I did not copy them from elsewhere, nor are they necessarily the most optimized versions. My goal has been to understand and build them from scratch rather than memorize ready-made solutions. Therefore, some may run slower than standard implementationsor sometimes even faster. For me, the process of understanding and constructing has been more important than simply reaching the fastest result.
Finally, let me tell you a bit about myself: I am **Mehdi**. If you prefer, you can call me by my alias: *Genix*. I am a student of Computer Engineering (at least at the time of writing this). I grew up with computersfrom simple games to typing commands in the terminaland I have always wondered what lies behind this screen of black and green text. There is not much you need to know about me, just that I am someone who works with computers, sometimes gives them commands, and sometimes learns from them.
I hope this book will be useful for understanding concepts, beginning your learning

journey, or diving deeper into data structures.
Arliz is freely available. You can access the PDF, LaTeX source, and related code at:

https://github.com/m-mdy-m/Arliz

In each chapter, I have included exercises and projects to aid your understanding. Please do not move on until you have completed these exercises, because true learning happens only by solving problems.
I hope this book serves you wellwhether for starting out, reviewing, or simply satisfying your curiosity. And if you learn something, find an error, or have a suggestion, please let me know. As I said: *This book grows with me.*

Acknowledgments

I would like to express my gratitude to everyone who supported me during the creation of this book. Special thanks to the open-source community for their invaluable resources and to all those who reviewed early drafts and provided feedback.

Contents

Part I

The Birth of Computing: From Mechanical to Electronic

Introduction

Long before a single line of code was ever writtenlong before electricity, transistors, or even the concept of modern logic circuitshumans felt an innate drive to calculate, record, and model the world around them. Computing is not a recent invention. It is one of humanitys oldest intellectual pursuits, rooted in necessity and evolved through creativity. Before we dive into complex abstractions like arrays or data structures, we must ask a deeper, almost philosophical question: **What does it mean to compute?**

This part of the book invites you on a journeynot just through the machinery and breakthroughs that brought us the modern computer, but through the evolution of human thought about numbers, representation, and control. Arrays, as we will later explore in depth, are not merely structures to store data. They are reflections of how weve ordered information for thousands of years. Their logic is built upon ancient insightson sets, sequences, and patternsand they embody the fundamental human need to represent, repeat, and manipulate structured information.

Our journey begins in ancient times, long before Christ, with devices like the abacus, first appearing over 2,500 years ago in Mesopotamia and later refined by Chinese, Roman, and Japanese cultures. The abacus was not just a calculatorit was an embodiment of the concepts of **state**, **position**, and **transformation**, principles that continue to underpin all modern computation. It allowed people to model quantities, track multiple values in parallel (an early echo of array indexing), and perform operations based on positional representation.

From these early tools, we progress into the classical mathematical age, where the Greeks formalized logic, and concepts like **sets** and **ordered lists** began to take philosophical shape. While not arrays in the modern sense, these ideas laid the intellectual groundwork for thinking about groups of datagrouped, related, or sequentialthat could be acted upon as a whole. The set, in particular, became a foundational concept in mathematics and later in programming: an abstract container for elements that obey rules and enable operations. The leap from abstract sets to concrete arrays reflects one of the key transitions in computational historyfrom idea to implementation.

In the 17th century, visionaries like Blaise Pascal and Gottfried Wilhelm Leibniz attempted to automate arithmetic with mechanical devices. These werent just clever toolsthey were the first signs of a dream to make thinking itself mechanical. Charles Babbage expanded this dream with his Analytical Engine in the 19th century, envisioning a machine that could be programmed and reprogrammeda concept that wouldnt become reality until a century later. Ada Lovelace, who worked with Babbage, went even further. She grasped that machines could go beyond numbers: they could pro-

cess symbolic logic, follow instructions, and even imitate aspects of reasoning. She anticipated the algorithm as a mental construct, not just a set of steps.
As we move forward into the 20th century, the invention of electromechanical and electronic machinesusing relays, vacuum tubes, and later transistorsmarked a revolution. No longer limited by gears and levers, computers became faster, more reliable, and more abstract. The idea of a **stored program** emerged, allowing machines to modify their behavior dynamically. This wasnt just a technical innovationit was a conceptual transformation. Programs became data, and data became active. Arrays, now implemented in memory, could be changed, traversed, and manipulated at runtimeopening the door to software as we know it today.
Eventually, we arrive at logic gates, boolean algebra, and the transistorthe atomic units of modern computation. These are more than circuits; they are the physical embodiment of logical thought: conditions, branching, repetition. From gates we build circuits, from circuits microprocessors, and from those, machines that can simulate anything we can formalize.
Before concluding this part, we will look closely at how data is represented: binary numbers, encoding schemes, floating-point formats, and character representations. These are not just technical tools; they are perspectives. They define the limits of what a machine can know, express, and manipulate. And finally, we arrive at memorywhere arrays live, grow, and function. Memory is not just storage; it is the canvas of computation. It is where change happens and where order emerges.
If you are excited to write code, build systems, and jump into implementation, you are free to skip ahead. But if you stay with us for this brief but essential historical and conceptual journey, you will see programming not just as control over a machine, but as part of a much older story: the story of how humans learned to structure thought, encode logic, and make abstract ideas come alive.
Let us beginat the beginning. With sand, stone, wood, and brass. And with minds bold enough to imagine machines that think.

4

How to Read

Part II

Philosophical & Historical Foundations

Introduction

How to Read

Part III

Mathematical Fundamentals

Introduction

How to Read

Part IV

Data Representation

Introduction

How to Read

Part V

Computer Architecture & Logic

Introduction

How to Read

Part VI

Array Odyssey

Introduction

How to Read

Part VII

Data Structures & Algorithms

Introduction

How to Read

Part VIII

Parallelism & Systems

Introduction

How to Read

Part IX

Synthesis & Frontiers

Introduction

How to Read