ARLIZ

A JOURNEY THROUGH ARRAYS

Mahdi

In Praise of Arliz

MAHDI

COMPUTER ENGINEERING STUDENT

This book evolves. Every insight gainedwhether a circuit, a structure, or a simple ideais absorbed and integrated. Arliz is never complete. Because understanding never is.

FIRST EDITION

May 27, 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.

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.

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 capac-ities 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.*

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.

Chapter 1

Mechanical Roots of Computing

1.1 From the Abacus to the Analytical Engine

1.1.1 The Abacus: The First Data Structure

1.1.2 Pascalin and Leibniz's Wheel

1.1.3 Babbie's Analytical Engine

1.1.4 Ada Lovelace and the First Algorithm

1.2 Electromechanical Computers and Early Concepts

Chapter 2

Introduction to Computers and Data Storage

2.1 A Brief History of Computing

2.1.1 From the Abacus to the Analytical Engine

2.1.2 The Electronic Computer Revolution

2.1.3 The Birth of Stored Programs

Chapter 3

The Birth of the Modern Computer and Its Architecture

3.1 The Transition to Electronic Computing

3.1.1 The Age of the Vacuum Tube

3.1.2 ENIAC and Early Electronic Computers

3.1.3 Von Neumann Architecture

3.1.4 The Concept of a Program Saved

Chapter 4

Hardware Foundations

4.1 Hardware Fundamentals

4.1.1 Logic Circuits and Gates

4.1.2 Von Neumann Architecture

4.2 Logic Gates and Boolean Algebra

4.3 Transistors: Building Blocks

4.4 Integrated Circuits and Microprocessors

4.5 Evolution of Computer Architecture

4.6 The Birth of Modern Computer Architecture

Chapter 5

Digital Logic and Boolean Foundations

5.1 Transistors: The Atomic Units of Computation

5.2 Logic Gates and Circuit Design

5.3 From NAND to NOR: Building Computational Primitives

Chapter 6

Number Systems and Data Representation

6.1 Historic Counting Systems

6.2 Binary: The Language of Machines

6.2.1 Unsigned Integer Representation

6.2.2 Two's Complement System

6.3 Floating Point: Representing the Continuous

6.4 Character Encoding Evolution

6.4.1 From EBCDIC to Unicode

Chapter 7

Memory: The Computer's Canvas

7.1 Historic Storage Media

7.1.1 Punch Cards to Core Memory

7.2 Modern Memory Hierarchy

7.2.1 Registers and Cache Architecture

7.2.2 RAM Geometries and Bank Organization

7.3 Address Space Concepts

7.3.1 Physical vs. Virtual Addressing

7.3.2 Memory Mapping and Address Translation

7.4 Memory Layout in Programs

7.4.1 Code Segment (Text Segment)

7.4.2 Data Segment (Initialized Data)

7.4.3 BSS Segment (Uninitialized Data)

7.4.4 Heap and Stack Segments

7.5 Memory Allocation Strategies

7.5.1 Static Memory Allocation

7.5.2 Stack-Based Allocation

Part II

The Array Odyssey

Chapter 8

Historical Emergence of Arrays

8.1 Early Array Concepts in Mathematics

8.2 Arrays in Assembly Language

8.2.1 IBM 704 Index Registers

8.3 Array Adoption in High-Level Languages

Chapter 9

Array Anatomy

9.1 Formal Mathematical Definition

9.2 Machine Representation

9.2.1 Contiguous Memory Layout

9.2.2 Stride and Cache Considerations

9.3 Dimensionality Perspectives

9.3.1 Physical vs. Logical Dimensions

Chapter 10

Memory Layout Engineering

10.1 Static Allocation Strategies

10.1.1 BSS vs. DATA Segments

10.2 Dynamic Allocation Mechanics

10.2.1 Heap Management Strategies

10.3 Multidimensional Mapping

10.3.1 Row-Major vs. Column-Major

10.3.2 Blocked Memory Layouts

Chapter 11

Array Indexing Evolution

11.1 Address Calculation Mathematics

11.1.1 Generalized Dimensional Formula

11.2 Bounds Checking Implementations

11.2.1 Hardware vs. Software Approaches

11.3 Pointer/Array Duality in C

Part III

Advanced Array Concepts

Chapter 12

Low-Level Optimization Techniques

12.1 Cache-Aware Array Traversal

12.2 SIMD Vectorization Strategies

12.3 False Sharing Prevention

Chapter 13

Theoretical Foundations

13.1 Arrays in Automata Theory

13.2 Turing Machines with Array Tapes

13.3 Chomsky Hierarchy Relationships

Chapter 14

Specialized Array Architectures

14.1 Sparse Array Storage

14.1.1 Compressed Sparse Row Format

14.2 Jagged Array Implementations

14.3 Associative Array Designs

Chapter 15

Computer Architecture Supplement

15.1 From Vacuum Tubes to VLSI

15.2 Pipeline Architectures Deep Dive

Chapter 16

Number System Reference

16.1 Positional Number Proofs

16.2 Endianness Conversion Algorithms

Chapter 17

Introduction to Arrays

17.1 Overview

17.2 Why Use Arrays?

17.3 History

Chapter 18

Basics of Array Operations

18.1 Traversal Operation

18.2 Insertion Operation

18.3 Deletion Operation

18.4 Search Operation

18.5 Sorting Operation

18.6 Access Operation

Chapter 19

Types and Representations of Arrays

19.1 Chomsky

19.2 Types

19.3 Abstract Arrays

Chapter 20

Memory Layout and Storage

20.1 Memory Layout of Arrays

20.2 Memory Segmentation and Bounds Checking

20.2.1 Memory Segmentation

Hardware Implementation

Segmentation without Paging

Segmentation with Paging

Historical Implementations

x86 Architecture

20.2.2 Index-Bounds Checking

Range Checking

Index Checking

Hardware Bounds Checking

Support in High-Level Programming Languages

Buffer Overflow

Integer Overflow

Chapter 21

Development of Array Indexing

Address Calculation for Multi-dimensional Arrays

One-Dimensional Array

Two-Dimensional Array

Three-Dimensional Array

Generalizing to a k-Dimensional Array

Examples

Chapter 22

Array Algorithms

22.1 Sorting Algorithms

22.2 Searching Algorithms

22.3 Array Manipulation Algorithms

22.4 Dynamic Programming and Arrays

Chapter 23

Practical and Advanced Topics

23.1 Self-Modifying Code in Early Computers

23.2 Common Array Algorithms

23.3 Performance Considerations

23.4 Practical Applications of Arrays

23.5 Future Trends in Array Handling

Chapter 24

Implementing Arrays in Low-Level Languages

Chapter 25

Static Arrays

25.1 Single-Dimensional Arrays

25.1.1 Declaration and Initialization

25.1.2 Accessing Elements

25.1.3 Iterating Through an Array

25.1.4 Common Operations

Insertion

Deletion

Searching

25.1.5 Memory Considerations

25.2 Multi-Dimensional Arrays

25.2.1 2D Arrays

Declaration and Initialization

Accessing Elements

Iterating Through a 2D Array

25.2.2 3D Arrays and Higher Dimensions

Declaration and Initialization

Accessing Elements

Use Cases and Applications

Chapter 26

Dynamic Arrays

26.1 Introduction to Dynamic Arrays

26.1.1 Definition and Overview

26.1.2 Comparison with Static Arrays

26.2 Single-Dimensional Dynamic Arrays

26.2.1 Using `malloc` and `calloc` in C

26.2.2 Resizing Arrays with `realloc`

26.2.3 Using `ArrayList` in Java

26.2.4 Using `Vector` in C++

26.2.5 Using `List` in Python

26.3 Multi-Dimensional Dynamic Arrays

26.3.1 2D Dynamic Arrays

Creating and Resizing 2D Arrays

26.3.2 3D and Higher Dimensions

Memory Allocation Techniques

Use Cases and Applications

Chapter 27

Advanced Topics in Arrays

27.1 Array Algorithms

27.1.1 Sorting Algorithms

Bubble Sort

Merge Sort

27.1.2 Searching Algorithms

Linear Search

Binary Search

27.2 Memory Management in Arrays

27.2.1 Static vs. Dynamic Memory

27.2.2 Optimizing Memory Usage

27.3 Handling Large Data Sets

27.3.1 Efficient Storage Techniques

27.3.2 Using Arrays in Big Data Applications

27.4 Parallel Processing with Arrays

27.4.1 Introduction to Parallel Arrays

27.4.2 Applications in GPU Programming

27.5 Sparse Arrays

Chapter 28

Arrays in Theoretical Computing Paradigms

28.1 Introduction to Theoretical Computing Paradigms

28.2 Arrays in Turing Machines

28.3 Arrays in Cellular Automata

28.4 Arrays in Cellular Automata

28.5 Arrays in Quantum Computing

28.6 Arrays in Neural Network Simulations

28.7 Arrays in Automata Theory

28.8 Arrays in Hypercomputation Models

28.9 The Lambda Calculus Perspective on Arrays

28.10 Arrays in Novel Computational Models

Chapter 29

Specialized Arrays and Applications

29.1 Circular Buffers

29.2 Circular Arrays

29.2.1 Implementation and Use Cases

29.2.2 Applications in Buffer Management

29.3 Dynamic Buffering and Arrays

29.3.1 Dynamic Circular Buffers

29.3.2 Handling Streaming Data

29.4 Jagged Arrays

29.4.1 Definition and Usage

29.4.2 Applications in Database Management

29.5 Bit Arrays (Bitsets)

29.5.1 Introduction and Representation

29.5.2 Applications in Cryptography

29.6 Circular Buffers

29.7 Priority Queues

29.8 Hash Tables

Chapter 30

Linked Lists

30.1 Overview

30.2 Singly Linked Lists

30.3 Doubly Linked Lists

30.4 Circular Linked Lists

30.5 Comparison with Arrays

Chapter 31

Array-Based Algorithms

31.1 Sorting Algorithms

31.2 Searching Algorithms

31.3 Array Manipulation Algorithms

31.4 Dynamic Programming and Arrays

Chapter 32

Performance Analysis

32.1 Time Complexity of Array Operations

32.2 Space Complexity Considerations

32.3 Cache Performance and Optimization

Chapter 33

Memory Management

33.1 Memory Allocation Strategies

33.2 Garbage Collection

33.3 Manual Memory Management in Low-Level Languages

Chapter 34

Error Handling and Debugging

34.1 Common Errors with Arrays

34.2 Bounds Checking Techniques

34.3 Debugging Tools and Strategies

Chapter 35

Optimization Techniques for Arrays

35.1 Optimizing Array Traversal

35.2 Minimizing Cache Misses

35.3 Loop Unrolling

35.4 Vectorization

35.5 Memory Access Patterns

35.6 Reducing Memory Fragmentation

Chapter 36

Concurrency and Parallelism

36.1 Concurrent Array Access

36.2 Parallel Array Processing

36.3 Synchronization Techniques

Chapter 37

Applications in Modern Software Development

37.1 Arrays in Graphics and Game Development

37.2 Arrays in Scientific Computing

37.3 Arrays in Data Analysis and Machine Learning

37.4 Arrays in Embedded Systems

Chapter 38

Arrays in High-Performance Computing (HPC)

38.1 Introduction to HPC Arrays

38.2 Distributed Arrays

38.3 Parallel Processing with Arrays

38.4 Arrays in GPU Computing

38.5 Multi-threaded Array Operations

38.6 Handling Arrays in Cloud Computing

Chapter 39

Arrays in Functional Programming

39.1 Immutable Arrays

39.2 Persistent Arrays

39.3 Arrays in Functional Languages (Haskell, Erlang, etc.)

39.4 Functional Array Operations

Chapter 40

Arrays in Machine Learning and Data Science

40.1 Numerical Arrays

40.2 Handling Large Datasets with Arrays

40.3 Arrays in Tensor Operations

40.4 Arrays in Dataframes

40.5 Optimization of Array-Based Algorithms in ML

Chapter 41

Advanced Memory Management in Arrays

41.1 Memory Pools

41.2 Dynamic Memory Allocation Strategies

Chapter 42

Data Structures Derived from Arrays

42.1 Stacks

42.2 Queues

42.3 Heaps

42.4 Hash Tables

42.5 Trees Implemented Using Arrays

42.6 Graphs Implemented Using Arrays

42.7 Dynamic Arrays as Building Blocks

Chapter 43

Best Practices and Common Pitfalls in Array Usage

43.1 Avoiding Out-of-Bounds Errors

43.2 Efficient Initialization

43.3 Choosing the Right Array Type

43.4 Debugging and Testing Arrays

43.5 Avoiding Memory Leaks

43.6 Ensuring Portability Across Platforms

Chapter 44

Historical Perspectives and Evolution

44.1 Custom Memory Allocators

44.2 Early Implementations

44.3 Array Storage on Disk

44.4 Evolution of Array Data Structures

44.5 Impact on Programming Languages and Paradigms

Chapter 45

Future Trends in Array Handling

45.1 Emerging Data Structures

45.2 Quantum Computing and Arrays

45.3 Bioinformatics Applications

45.4 Big Data and Arrays

45.5 Arrays in Emerging Programming Paradigms

Chapter 46

Appendices

46.1 Glossary of Terms

46.2 Bibliography

46.3 Index