Arliz

A Journey Through Arrays and Computer Fundamentals

From Bits to Data Structures

Mahdi

May 27, 2025

Contents

0.1 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 Iofficiallycall 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 wordencountered 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-ees, 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 alongside 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 structureI incorporate it into Arliz.

This means Arliz is a living project. As long as I keep learning, Arliz will remain alive.

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.

Part I

The Birth of Computing: From Mechanical to Electronic

Introduction

Long before a single line of code was ever written, long before the transistor or even electricity became central to our world, the desire to computeto quantify, to measure, to simulatewas embedded in the human spirit. Computing is not a modern invention. It is an ancient pursuit, born out of necessity and nurtured by imagination. Before we attempt to understand complex abstractions like arrays or the intricate orchestration of modern architectures, it is worth taking a step back and asking a more fundamental question: What does it mean to compute?

This part of the book invites you on a journeynot just through the technical milestones that led to the digital world, but through the evolution of ideas. Ideas about information. About logic. About representation. And above all, about control. Arrays, as we will explore later, are not simply ways to store numbers in memory. They are symbolic containers for how we humans conceive order, access, structure, and repetition. They are computational reflections of patterns we've observed and manipulated for millennia.

To truly understand arrays and all data structures that follow, we must understand their philosophical and physical roots. We begin our journey with the abacus, a device that predates recorded history yet encodes principles of state, transformation, and positionconcepts which remain foundational in modern computation. From there, we traverse the age of mechanical innovation, when inventors like Pascal and Leibniz attempted to mechanize arithmetic. Then comes Charles Babbage, whose vision of the Analytical Engine outlined the blueprint for a programmable machine nearly a century before electricity revolutionized the field. And Ada Lovelace, who saw beyond machinery to glimpse the essence of algorithmic thought. She realized that machines could go beyond numbersthat they could model logic, creativity, and potentially even cognition.

As we progress, we enter the era of electromechanical and electronic machines: hulking contraptions powered by relays, vacuum tubes, and eventually transistors. These developments were not just advances in speed or reliability; they marked a conceptual leap. They transformed computing from a mechanical operation into a symbolic one. Logic was no longer just gears and leversit became voltages and switches, and eventually, binary abstraction.

One of the most critical conceptual shifts came with the notion of a "stored program." Before this idea took hold, machines were hardwired for a single task. But in the stored-program model, the machine could modify its behavior based on the data it processedblurring the line between hardware and software, between machine and

meaning. This shift underpins all modern computing. Without it, arrays wouldn't exist as dynamic, mutable objects. Memory would not be a programmable canvas. Programs themselves would not be flexible instruments of logic, but static constructs.
Our journey will then transition into the heart of computer hardware: gates, circuits, boolean algebra, and transistors. Not merely as physical components, but as ideasways of controlling flow, enforcing conditions, and making decisions. The logic gate is not just a transistor trick; it is the embodiment of conditionality. It is if, and, or, notthe very syntax of thought, encoded in silicon.
We then observe the progression from individual gates to integrated circuits and finally microprocessors: the functional cores of computers that now reside on chips smaller than your fingernail but capable of billions of operations per second. As abstraction increases, so does complexityand yet, the foundational ideas remain remarkably stable. Truth tables, combinational logic, flip-flops, and memory cells still govern our most advanced machines.
Before concluding this part, we explore the representation of data itself: number systems, binary encoding, character sets, and floating-point arithmetic. These are not merely technical encodings; they are worldviews. They define what a computer can know, can store, and can express. You will encounter how a machine sees numbers, letters, and even concepts like time and positionnot with intuition, but with representation, ranges, and encodings. You will see how bits become meaning.
And finally, we arrive at memorythe substrate of state, the ground on which all computation stands. From early punch cards and magnetic drums to modern RAM hierarchies and virtual address spaces, memory has always been more than storage. It is the record of thought in progress, the only place where logic can become consequence. Arrays, in particular, are born in memory. Their efficiency, limitations, and power derive from how memory is structured, addressed, and accessed.
If you are someone eager to jump into code, to build systems, to write loops and manipulate structures, you may be tempted to skip ahead. And you are welcome to do so. But by understanding where computation comes fromnot just the how, but the whyyou will gain something more profound. You will see programming not as instruction-giving, but as idea-expression. Not as control over machines, but as alignment with centuries of thought.
This is not merely history. It is orientation. It is the intellectual soil from which your code will grow.
Let us now begin at the beginningwith machines made of wood and brass, and with minds that dared to make them 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

Part II

From Bits to Structures

Chapter 8

Data Organization Principles

8.1 The Philosophy of Structured Storage

8.2 Primitive Data Types

8.2.1 Bit Patterns and Value Interpretation

8.3 Composite Data Types

8.3.1 Packed vs. Unpacked Formats

Chapter 9

Memory Access Patterns

9.1 Big/Little Endian Architectures

9.2 Memory Alignment Considerations

9.3 Pointer Arithmetic Deep Dive

9.3.1 Type-Safe Addressing

Part III

The Array Odyssey

Chapter 10

Historical Emergence of Arrays

10.1 Early Array Concepts in Mathematics

10.2 Arrays in Assembly Language

10.2.1 IBM 704 Index Registers

10.3 Array Adoption in High-Level Languages

Chapter 11

Array Anatomy

11.1 Formal Mathematical Definition

11.2 Machine Representation

11.2.1 Contiguous Memory Layout

11.2.2 Stride and Cache Considerations

11.3 Dimensionality Perspectives

11.3.1 Physical vs. Logical Dimensions

Chapter 12

Memory Layout Engineering

12.1 Static Allocation Strategies

12.1.1 BSS vs. DATA Segments

12.2 Dynamic Allocation Mechanics

12.2.1 Heap Management Strategies

12.3 Multidimensional Mapping

12.3.1 Row-Major vs. Column-Major

12.3.2 Blocked Memory Layouts

Chapter 13

Array Indexing Evolution

13.1 Address Calculation Mathematics

13.1.1 Generalized Dimensional Formula

13.2 Bounds Checking Implementations

13.2.1 Hardware vs. Software Approaches

13.3 Pointer/Array Duality in C

Part IV

Advanced Array Concepts

Chapter 14

Low-Level Optimization Techniques

14.1 Cache-Aware Array Traversal

14.2 SIMD Vectorization Strategies

14.3 False Sharing Prevention

Chapter 15

Theoretical Foundations

15.1 Arrays in Automata Theory

15.2 Turing Machines with Array Tapes

15.3 Chomsky Hierarchy Relationships

Chapter 16

Specialized Array Architectures

16.1 Sparse Array Storage

16.1.1 Compressed Sparse Row Format

16.2 Jagged Array Implementations

16.3 Associative Array Designs

Chapter 17

Computer Architecture Supplement

17.1 From Vacuum Tubes to VLSI

17.2 Pipeline Architectures Deep Dive

Chapter 18

Number System Reference

18.1 Positional Number Proofs

18.2 Endianness Conversion Algorithms

Chapter 19

Introduction to Arrays

19.1 Overview

19.2 Why Use Arrays?

19.3 History

Chapter 20

Basics of Array Operations

20.1 Traversal Operation

20.2 Insertion Operation

20.3 Deletion Operation

20.4 Search Operation

20.5 Sorting Operation

20.6 Access Operation

Chapter 21

Types and Representations of Arrays

21.1 Chomsky

21.2 Types

21.3 Abstract Arrays

Chapter 22

Memory Layout and Storage

22.1 Memory Layout of Arrays

22.2 Memory Segmentation and Bounds Checking

22.2.1 Memory Segmentation

Hardware Implementation

Segmentation without Paging

Segmentation with Paging

Historical Implementations

x86 Architecture

22.2.2 Index-Bounds Checking

Range Checking

Index Checking

Hardware Bounds Checking

Support in High-Level Programming Languages

Buffer Overflow

Integer Overflow

Chapter 23

Development of Array Indexing

Chapter 24

Array Algorithms

24.1 Sorting Algorithms

24.2 Searching Algorithms

24.3 Array Manipulation Algorithms

24.4 Dynamic Programming and Arrays

Chapter 25

Practical and Advanced Topics

25.1 Self-Modifying Code in Early Computers

25.2 Common Array Algorithms

25.3 Performance Considerations

25.4 Practical Applications of Arrays

25.5 Future Trends in Array Handling

Chapter 26

Implementing Arrays in Low-Level Languages

Chapter 27

Static Arrays

27.1 Single-Dimensional Arrays

27.1.1 Declaration and Initialization

27.1.2 Accessing Elements

27.1.3 Iterating Through an Array

27.1.4 Common Operations

Insertion

Deletion

Searching

27.1.5 Memory Considerations

27.2 Multi-Dimensional Arrays

27.2.1 2D Arrays

Declaration and Initialization

Accessing Elements

Iterating Through a 2D Array

27.2.2 3D Arrays and Higher Dimensions

Declaration and Initialization

Accessing Elements

Use Cases and Applications

Chapter 28

Dynamic Arrays

28.1 Introduction to Dynamic Arrays

28.1.1 Definition and Overview

28.1.2 Comparison with Static Arrays

28.2 Single-Dimensional Dynamic Arrays

28.2.1 Using `malloc` and `calloc` in C

28.2.2 Resizing Arrays with `realloc`

28.2.3 Using `ArrayList` in Java

28.2.4 Using `Vector` in C++

28.2.5 Using `List` in Python

28.3 Multi-Dimensional Dynamic Arrays

28.3.1 2D Dynamic Arrays

Creating and Resizing 2D Arrays

28.3.2 3D and Higher Dimensions

Memory Allocation Techniques

Use Cases and Applications

Chapter 29

Advanced Topics in Arrays

29.1 Array Algorithms

29.1.1 Sorting Algorithms

Bubble Sort

Merge Sort

29.1.2 Searching Algorithms

Linear Search

Binary Search

29.2 Memory Management in Arrays

29.2.1 Static vs. Dynamic Memory

29.2.2 Optimizing Memory Usage

29.3 Handling Large Data Sets

29.3.1 Efficient Storage Techniques

29.3.2 Using Arrays in Big Data Applications

29.4 Parallel Processing with Arrays

29.4.1 Introduction to Parallel Arrays

29.4.2 Applications in GPU Programming

29.5 Sparse Arrays

Chapter 30

Arrays in Theoretical Computing Paradigms

Chapter 31

Specialized Arrays and Applications

31.1 Circular Buffers

31.2 Circular Arrays

31.2.1 Implementation and Use Cases

31.2.2 Applications in Buffer Management

31.3 Dynamic Buffering and Arrays

31.3.1 Dynamic Circular Buffers

31.3.2 Handling Streaming Data

31.4 Jagged Arrays

31.4.1 Definition and Usage

31.4.2 Applications in Database Management

31.5 Bit Arrays (Bitsets)

31.5.1 Introduction and Representation

31.5.2 Applications in Cryptography

31.6 Circular Buffers

31.7 Priority Queues

31.8 Hash Tables

Chapter 32

Linked Lists

32.1 Overview

32.2 Singly Linked Lists

32.3 Doubly Linked Lists

32.4 Circular Linked Lists

32.5 Comparison with Arrays

Chapter 33

Array-Based Algorithms

33.1 Sorting Algorithms

33.2 Searching Algorithms

33.3 Array Manipulation Algorithms

33.4 Dynamic Programming and Arrays

Chapter 34

Performance Analysis

34.1 Time Complexity of Array Operations

34.2 Space Complexity Considerations

34.3 Cache Performance and Optimization

Chapter 35

Memory Management

35.1 Memory Allocation Strategies

35.2 Garbage Collection

35.3 Manual Memory Management in Low-Level Languages

Chapter 36

Error Handling and Debugging

36.1 Common Errors with Arrays

36.2 Bounds Checking Techniques

36.3 Debugging Tools and Strategies

Chapter 37

Optimization Techniques for Arrays

37.1 Optimizing Array Traversal

37.2 Minimizing Cache Misses

37.3 Loop Unrolling

37.4 Vectorization

37.5 Memory Access Patterns

37.6 Reducing Memory Fragmentation

Chapter 38

Concurrency and Parallelism

38.1 Concurrent Array Access

38.2 Parallel Array Processing

38.3 Synchronization Techniques

Chapter 39

Applications in Modern Software Development

39.1 Arrays in Graphics and Game Development

39.2 Arrays in Scientific Computing

39.3 Arrays in Data Analysis and Machine Learning

39.4 Arrays in Embedded Systems

Chapter 40

Arrays in High-Performance Computing (HPC)

Chapter 41

Arrays in Functional Programming

41.1 Immutable Arrays

41.2 Persistent Arrays

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

41.4 Functional Array Operations

Chapter 42

Arrays in Machine Learning and Data Science

42.1 Numerical Arrays

42.2 Handling Large Datasets with Arrays

42.3 Arrays in Tensor Operations

42.4 Arrays in Dataframes

42.5 Optimization of Array-Based Algorithms in ML

Chapter 43

Advanced Memory Management in Arrays

43.1 Memory Pools

43.2 Dynamic Memory Allocation Strategies

Chapter 44

Data Structures Derived from Arrays

44.1 Stacks

44.2 Queues

44.3 Heaps

44.4 Hash Tables

44.5 Trees Implemented Using Arrays

44.6 Graphs Implemented Using Arrays

44.7 Dynamic Arrays as Building Blocks

Chapter 45

Best Practices and Common Pitfalls in Array Usage

45.1 Avoiding Out-of-Bounds Errors

45.2 Efficient Initialization

45.3 Choosing the Right Array Type

45.4 Debugging and Testing Arrays

45.5 Avoiding Memory Leaks

45.6 Ensuring Portability Across Platforms

Chapter 46

Historical Perspectives and Evolution

46.1 Custom Memory Allocators

46.2 Early Implementations

46.3 Array Storage on Disk

46.4 Evolution of Array Data Structures

46.5 Impact on Programming Languages and Paradigms

Chapter 47

Future Trends in Array Handling

47.1 Emerging Data Structures

47.2 Quantum Computing and Arrays

47.3 Bioinformatics Applications

47.4 Big Data and Arrays

47.5 Arrays in Emerging Programming Paradigms

Chapter 48

Appendices

48.1 Glossary of Terms

48.2 Bibliography

48.3 Index