Data Structures and Algorithms with Object-Oriented Design Patterns in C#

Bruno R. Preiss
B.A.Sc., M.A.Sc., Ph.D., P.Eng.
Software Engineer and Architect
SOMA Networks, Inc.
Toronto, Canada

- Colophon
- Dedication
- Preface
- Contents
- Introduction
- Algorithm Analysis
- Asymptotic Notation
- Foundational Data Structures
- Data Types and Abstraction
- Stacks, Queues, and Deques
- Ordered Lists and Sorted Lists
- Hashing, Hash Tables, and Scatter Tables
- Trees
- Search Trees
- Heaps and Priority Queues
- Sets, Multisets, and Partitions
- Garbage Collection and the Other Kind of Heap
- Algorithmic Patterns and Problem Solvers
Colophon

Copyright © 19101 by Bruno R. Preiss.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author.

This book was prepared with LaTeX and reproduced from camera-ready copy supplied by the author. The book is typeset using the Computer Modern fonts designed by Donald E. Knuth with various additional glyphs designed by the author and implemented using METAFONT.

METAFONT is a trademark of Addison Wesley Publishing Company.

Java is a registered trademark of Sun Microsystems.

TeX is a trademark of the American Mathematical Society.

UNIX is a registered trademark of AT&T Bell Laboratories.

Microsoft is a registered trademark of Microsoft Corporation.
Dedication

To Patty

Copyright © 2001 by Bruno R. Preiss, P.Eng. All rights reserved.
Preface

This book was motivated by my experience in teaching the course E&CE 250: Algorithms and Data Structures in the Computer Engineering program at the University of Waterloo. I have observed that the advent of object-oriented methods and the emergence of object-oriented design patterns has lead to a profound change in the pedagogy of data structures and algorithms. The successful application of these techniques gives rise to a kind of cognitive unification: Ideas that are disparate and apparently unrelated seem to come together when the appropriate design patterns and abstractions are used.

This paradigm shift is both evolutionary and revolutionary. On the one hand, the knowledge base grows incrementally as programmers and researchers invent new algorithms and data structures. On the other hand, the proper use of object-oriented techniques requires a fundamental change in the way the programs are designed and implemented. Programmers who are well schooled in the procedural ways often find the leap to objects to be a difficult one.

• Goals
• Approach
• Outline
• Suggested Course Outline
• Online Course Materials

Copyright © 2001 by Bruno R. Preiss, P.Eng. All rights reserved.
Goals

The primary goal of this book is to promote object-oriented design using C# and to illustrate the use of the emerging object-oriented design patterns. Experienced object-oriented programmers find that certain ways of doing things work best and that these ways occur over and over again. The book shows how these patterns are used to create good software designs. In particular, the following design patterns are used throughout the text: singleton, container, enumeration, adapter and visitor.

Virtually all of the data structures are presented in the context of a single, unified, polymorphic class hierarchy. This framework clearly shows the relationships between data structures and it illustrates how polymorphism and inheritance can be used effectively. In addition, algorithmic abstraction is used extensively when presenting classes of algorithms. By using algorithmic abstraction, it is possible to describe a generic algorithm without having to worry about the details of a particular concrete realization of that algorithm.

A secondary goal of the book is to present mathematical tools just in time. Analysis techniques and proofs are presented as needed and in the proper context. In the past when the topics in this book were taught at the graduate level, an author could rely on students having the needed background in mathematics. However, because the book is targeted for second- and third-year students, it is necessary to fill in the background as needed. To the extent possible without compromising correctness, the presentation fosters intuitive understanding of the concepts rather than mathematical rigor.

Copyright © 2001 by Bruno R. Preiss, P.Eng. All rights reserved.
Approach

One cannot learn to program just by reading a book. It is a skill that must be developed by practice. Nevertheless, the best practitioners study the works of others and incorporate their observations into their own practice. I firmly believe that after learning the rudiments of program writing, students should be exposed to examples of complex, yet well-designed program artifacts so that they can learn about the designing good software.

Consequently, this book presents the various data structures and algorithms as complete C# program fragments. All the program fragments presented in this book have been extracted automatically from the source code files of working and tested programs. It has been my experience that by developing the proper abstractions, it is possible to present the concepts as fully functional programs without resorting to pseudo-code or to hand-waving.
Contents

- Colophon
- Dedication
- Preface
  - Goals
  - Approach
  - Outline
  - Suggested Course Outline
  - Online Course Materials
- Introduction
  - What This Book Is About
  - Object-Oriented Design
    - Abstraction
    - Encapsulation
  - Object Hierarchies and Design Patterns
    - Containers
    - Enumerators
    - Visitors
    - Cursors
    - Adapters
    - Singletons
  - The Features of C# You Need to Know
    - Variables
    - Value Types and Reference Types
    - Parameter Passing
    - Classes and Objects
    - Inheritance
    - Interfaces and Polymorphism
    - Other Features
  - How This Book Is Organized
    - Models and Asymptotic Analysis
    - Foundational Data Structures
    - Abstract Data Types and the Class Hierarchy
Contents

- Data Structures
- Algorithms

**Algorithm Analysis**
- A Detailed Model of the Computer
  - The Basic Axioms
  - A Simple Example-Arithmetic Series Summation
  - Array Subscripting Operations
  - Another Example-Horner's Rule
  - Analyzing Recursive Methods
    - Solving Recurrence Relations-Repeated Substitution
  - Yet Another Example-Finding the Largest Element of an Array
  - Average Running Times
  - About Harmonic Numbers
  - Best-Case and Worst-Case Running Times
  - The Last Axiom
- A Simplified Model of the Computer
  - An Example-Geometric Series Summation
  - About Arithmetic Series Summation
  - Example-Geometric Series Summation Again
  - About Geometric Series Summation
  - Example-Computing Powers
  - Example-Geometric Series Summation Yet Again
- Exercises
- Projects

**Asymptotic Notation**
- An Asymptotic Upper Bound-Big Oh
  - A Simple Example
  - Big Oh Fallacies and Pitfalls
  - Properties of Big Oh
  - About Polynomials
  - About Logarithms
  - Tight Big Oh Bounds
  - More Big Oh Fallacies and Pitfalls
  - Conventions for Writing Big Oh Expressions
- An Asymptotic Lower Bound-Omega
  - A Simple Example
  - About Polynomials Again
- More Notation-Theta and Little Oh
Contents

- Implementation
  - Queues
    - Array Implementation
      - Fields
      - Constructor and Purge Methods
      - Enqueue and Dequeue Methods, Head Property
    - Linked-List Implementation
      - Fields
      - Constructor and Purge Methods
      - Enqueue and Dequeue Methods, Head Property
  - Applications
    - Implementation
  - Deques
    - Array Implementation
      - The "Head" Operations
      - The "Tail" Operations
    - Linked List Implementation
      - The "Head" Operations
      - The "Tail" Operations
    - Doubly-Linked and Circular Lists
  - Exercises
  - Projects
- Ordered Lists and Sorted Lists
  - Ordered Lists
    - Array Implementation
      - Fields
      - Creating a List and Inserting Items
      - Finding Items in a List
      - Removing Items from a List
      - Positions of Items in a List
      - Finding the Position of an Item and Accessing by Position
      - Inserting an Item at an Arbitrary Position
      - Removing Arbitrary Items by Position
    - Linked-List Implementation
      - Fields
      - Inserting and Accessing Items in a List
      - Finding Items in a List
      - Removing Items from a List
Positions of Items in a List
Finding the Position of an Item and Accessing by Position
Inserting an Item at an Arbitrary Position
Removing Arbitrary Items by Position
Performance Comparison: OrderedListAsArray vs. ListAsLinkedList
Applications

Sorted Lists
Array Implementation
Inserting Items in a Sorted List
Locating Items in an Array-Binary Search
Finding Items in a Sorted List
Removing Items from a List
Linked-List Implementation
Inserting Items in a Sorted List
Other Operations on Sorted Lists
Performance Comparison: SortedListAsArray vs. SortedListAsList
Applications
Implementation
Analysis

Exercises
Projects

Hashing, Hash Tables, and Scatter Tables
Hashing-The Basic Idea
Example
Keys and Hash Functions
Avoiding Collisions
Spreading Keys Evenly
Ease of Computation

Hashing Methods
Division Method
Middle Square Method
Multiplication Method
Fibonacci Hashing

Hash Function Implementations
Integral Keys
Floating-Point Keys
Character String Keys
Hashing Containers
- **Search Trees**
  - **Basics**
    - *M*-Way Search Trees
    - Binary Search Trees
  - **Searching a Search Tree**
    - Searching an *M*-way Tree
    - Searching a Binary Tree
  - **Average Case Analysis**
    - Successful Search
    - Solving The Recurrence-Telescoping
    - Unsuccessful Search
    - Traversing a Search Tree
  - **Implementing Search Trees**
    - Binary Search Trees
      - Fields
      - Find Method
      - Min Property
    - Inserting Items in a Binary Search Tree
      - Insert and AttachKey Methods
    - Removing Items from a Binary Search Tree
      - Withdraw Method
  - **AVL Search Trees**
    - Implementing AVL Trees
      - Constructor
      - AdjustHeight Method, Height and BalanceFactor Properties
    - Inserting Items into an AVL Tree
      - Balancing AVL Trees
      - Single Rotations
      - Double Rotations
      - Implementation
    - Removing Items from an AVL Tree
  - ***M*-Way Search Trees**
    - Implementing *M*-Way Search Trees
      - Implementation
      - Constructor and *M* Property
      - Inorder Traversal
    - Finding Items in an *M*-Way Search Tree
      - Linear Search
Contents

- Fields
  - AddTree and RemoveTree
  - MinTree and Min Properties
- Merging Binomial Queues
- Putting Items into a Binomial Queue
- Removing an Item from a Binomial Queue

- Applications
  - Discrete Event Simulation
  - Implementation

- Exercises

- Projects

- Sets, Multisets, and Partitions
  - Basics
    - Implementing Sets
  - Array and Bit-Vector Sets
    - Basic Operations
    - Union, Intersection, and Difference
    - Comparing Sets
    - Bit-Vector Sets
      - Basic Operations
      - Union, Intersection, and Difference
  - Multisets
    - Array Implementation
      - Basic Operations
      - Union, Intersection, and Difference
    - Linked-List Implementation
      - Union
      - Intersection
  - Partitions
    - Representing Partitions
    - Implementing a Partition using a Forest
      - Implementation
      - Constructor
      - Find and Join Methods
    - Collapsing Find
    - Union by Size
    - Union by Height or Rank

- Applications
Garbage Collection and the Other Kind of Heap

What is Garbage?
- Reduce, Reuse, Recycle
  - Reduce
  - Reuse
  - Recycle
- Helping the Garbage Collector

Reference Counting Garbage Collection
- When Objects Refer to Other Objects
- Why Reference Counting Does Not Work

Mark-and-Sweep Garbage Collection
- The Fragmentation Problem

Stop-and-Copy Garbage Collection
- The Copy Algorithm

Mark-and-Compact Garbage Collection
- Handles

Algorithmic Patterns and Problem Solvers

Brute-Force and Greedy Algorithms
- Example-Counting Change
  - Brute-Force Algorithm
  - Greedy Algorithm
- Example-0/1 Knapsack Problem

Backtracking Algorithms
- Example-Balancing Scales
- Representing the Solution Space
- Abstract Backtracking Solvers
  - Abstract Solvers
  - Depth-First Solver
  - Breadth-First Solver
- Branch-and-Bound Solvers
  - Depth-First, Branch-and-Bound Solver
- Example-0/1 Knapsack Problem Again

Top-Down Algorithms: Divide-and-Conquer
- Example-Binary Search
Examples

- Computing Fibonacci Numbers
- Merge Sorting
- Running Time of Divide-and-Conquer Algorithms
  - Case 1 (\(a > b\))
  - Case 2 (\(a = b\))
  - Case 3 (\(a < b\))
- Matrix Multiplication

Summary

Bottom-Up Algorithms: Dynamic Programming

- Generalized Fibonacci Numbers
- Computing Binomial Coefficients
- Application: Typesetting Problem
  - Example
  - Implementation

Randomized Algorithms

- Generating Random Numbers
  - Minimal Standard Random Number Generator
  - Implementation
- Random Variables
  - A Simple Random Variable
  - Uniformly Distributed Random Variables
  - Exponentially Distributed Random Variables
- Monte Carlo Methods
  - Computing Simulated Annealing
- Balancing Scales

Exercises

Projects

Sorting Algorithms and Sorters

- Basics
- Sorting and Sorters
  - Abstract Sorters
  - Sorter Class Hierarchy
- Insertion Sorting
  - Straight Insertion Sort
  - Implementation
Contents

- Directed Acyclic Graphs
- Undirected Graphs
- Terminology
- Labeled Graphs
- Representing Graphs
  - Adjacency Matrices
  - Sparse vs. Dense Graphs
  - Adjacency Lists
- Implementing Graphs
  - Vertices
    - Enumerators
  - Edges
  - Graphs and Digraphs
    - Accessors and Mutators
    - Enumerators
    - Graph Traversals
- Directed Graphs
- Abstract Graphs
- Implementing Undirected Graphs
  - Using Adjacency Matrices
  - Using Adjacency Lists
- Comparison of Graph Representations
  - Space Comparison
  - Time Comparison
- Graph Traversals
  - Depth-First Traversal
    - Implementation
    - Running Time Analysis
  - Breadth-First Traversal
    - Implementation
    - Running Time Analysis
  - Topological Sort
    - Implementation
    - Running Time Analysis
- Graph Traversal Applications:
  - Testing for Cycles and Connectedness
    - Connectedness of an Undirected Graph
    - Connectedness of a Directed Graph
Contents

- **Testing Strong Connectedness**
- **Testing for Cycles in a Directed Graph**

### Shortest-Path Algorithms
- **Single-Source Shortest Path**
  - **Dijkstra's Algorithm**
  - **Data Structures for Dijkstra's Algorithm**
  - **Implementation**
  - **Running Time Analysis**
- **All-Pairs Source Shortest Path**
  - **Floyd's Algorithm**
  - **Implementation**
  - **Running Time Analysis**

### Minimum-Cost Spanning Trees
- **Constructing Spanning Trees**
- **Minimum-Cost Spanning Trees**
- **Prim's Algorithm**
  - **Implementation**
- **Kruskal's Algorithm**
  - **Implementation**
  - **Running Time Analysis**

### Application: Critical Path Analysis
- **Implementation**

### Exercises

### Projects

### C# and Object-Oriented Programming

#### Variables
- **Value Types**
- **References Types**
  - **Null References**

#### Parameter Passing
- **Pass By Value**
- **Passing By Reference**
  - **The Trade-off**
- **In and Out Parameters**
- **Passing Reference Types**

#### Objects and Classes
- **Class Members: Fields and Methods**
- **Constructors**
Contents

- The No-Arg Constructor
- Properties and Accessors
- Member Access Control
- Operator Overloading
- Nested Classes
- Inheritance and Polymorphism
  - Derivation and Inheritance
    - Derivation and Access Control
  - Polymorphism
    - Interfaces
    - Abstract Methods and Abstract Classes
    - Method Resolution
    - Abstract Classes and Concrete Classes
    - Algorithmic Abstraction
- Multiple Inheritance
- Run-Time Type Information and Casts
- Exceptions
- Class Hierarchy Diagrams
- Character Codes
- References
- Index

Copyright © 2001 by Bruno R. Preiss, P.Eng. All rights reserved.