1. Record Nr. UNINA9910150213303321

Autore	Carrano Frank M.
Titolo	Data abstraction & problem solving with C++ : walls and mirrors / / Frank M. Carrano and Timothy Henry
Pubbl/distr/stampa	Boston : , : Pearson, , [2013] ©2013
ISBN	0-273-77827-7
Edizione	[Sixth edition.]
Descrizione fisica	1 online resource (842 pages) : illustrations
Collana	Always Learning
Disciplina	511.8
Soggetti	Problem solving - Data processing
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Note generali	Includes index.
Nota di contenuto	Cover -- Table of Contents -- Chapter 1 Data Abstraction: The Walls -- 1.1 Object-Oriented Concepts -- 1.1.1 Object-Oriented Analysis and Design -- 1.1.2 Aspects of an Object-Oriented Solution -- 1.2 Achieving a Better Solution -- 1.2.1 Cohesion -- 1.2.2 Coupling -- 1.3 Specifications -- 1.3.1 Operation Contracts -- 1.3.2 Unusual Conditions -- 1.3.3 Abstraction -- 1.3.4 Information Hiding -- 1.3.5 Minimal and Complete Interfaces -- 1.4 Abstract Data Types -- 1.4.1 Designing an ADT -- 1.4.2 ADTs That Suggest Other ADTs -- 1.5 The ADT Bag -- 1.5.1 Identifying Behaviors -- 1.5.2 Specifying Data and Operations -- 1.5.3 An Interface Template for the ADT -- 1.5.4 Using the ADT Bag -- C++ Interlude 1 C++ Classes -- C1.1 A Problem to Solve -- C1.1.1 Private Data Fields -- C1.1.2 Constructors and Destructors -- C1.1.3 Methods -- C1.1.4 Preventing Compiler Errors -- C1.2 Implementing a Solution -- C1.3 Templates -- C1.4 Inheritance -- C1.4.1 Base Classes and Derived Classes -- C1.4.2 Overriding Base-Class Methods -- C1.5 Virtual Methods and Abstract Classes -- C1.5.1 Virtual Methods -- C1.5.2 Abstract Classes -- Chapter 2 Recursion: The Mirrors -- 2.1 Recursive Solutions -- 2.2 Recursion That Returns a Value -- 2.2.1 A Recursive Valued Function: The Factorial of n -- 2.2.2 The Box Trace -- 2.3 Recursion That Performs an Action -- 2.3.1 A Recursive Void Function: Writing a String Backward -- 2.4 Recursion with Arrays -- 2.4.1 Writing an Array's Entries in Backward Order -- 2.4.2 The Binary Search -- 2.4.3 Finding the Largest Value in an Array

-- 2.4.4 Finding the k th Smallest Value of an Array -- 2.5 Organizing Data -- 2.5.1 The Towers of Hanoi -- 2.6 More Examples -- 2.6.1 The Fibonacci Sequence (Multiplying Rabbits) -- 2.6.2 Organizing a Parade -- 2.6.3 Choosing k Out of n Things -- 2.7 Recursion and Efficiency -- Chapter 3 Array-Based Implementations.
3.1 The Approach -- 3.1.1 Core Methods -- 3.1.2 Using Fixed-Size Arrays -- 3.2 An Array-Based Implementation of the ADT Bag -- 3.2.1 The Header File -- 3.2.2 Defining the Core Methods -- 3.2.3 Testing the Core Methods -- 3.2.4 Implementing More Methods -- 3.2.5 Methods That Remove Entries -- 3.2.6 Testing -- 3.3 Using Recursion in the Implementation -- 3.3.1 The Method getIndexOf -- 3.3.2 The Method getFrequencyOf -- C++ Interlude 2 Pointers, Polymorphism, and Memory Allocation -- C2.1 Memory Allocation for Variables and Early Binding of Methods -- C2.2 A Problem to Solve -- C2.3 Pointers and the Program's Free Store -- C2.3.1 Deallocating Memory -- C2.3.2 Avoiding Memory Leaks -- C2.3.3 Avoiding Dangling Pointers -- C2.4 Virtual Methods and Polymorphism -- C2.5 Dynamic Allocation of Arrays -- C2.5.1 A Resizable Array-Based Bag -- Chapter 4 Link-Based Implementations -- 4.1 Preliminaries -- 4.1.1 The Class Node -- 4.2 A Link-Based Implementation of the ADT Bag -- 4.2.1 The Header File -- 4.2.2 Defining the Core Methods -- 4.2.3 Implementing More Methods -- 4.3 Using Recursion in Link-Based Implementations -- 4.3.1 Recursive Definitions of Methods in LinkedBag -- 4.4 Testing Multiple ADT Implementations -- 4.5 Comparing Array-Based and Link-Based Implementations -- Chapter 5 Recursion as a Problem-Solving Technique -- 5.1 Defi ning Languages -- 5.1.1 The Basics of Grammars -- 5.1.2 Two Simple Languages -- 5.2 Algebraic Expressions -- 5.2.1 Kinds of Algebraic Expressions -- 5.2.2 Prefix Expressions -- 5.2.3 Postfix Expressions -- 5.2.4 Fully Parenthesized Expressions -- 5.3 Backtracking -- 5.3.1 Searching for an Airline Route -- 5.3.2 The Eight Queens Problem -- 5.4 The Relationship Between Recursion and Mathematical Induction -- 5.4.1 The Correctness of the Recursive Factorial Function -- 5.4.2 The Cost of Towers of Hanoi -- Chapter 6 Stacks.
6.1 The Abstract Data Type Stack -- 6.1.1 Developing an ADT During the Design of a Solution -- 6.1.2 Specifications for the ADT Stack -- 6.2 Simple Uses of a Stack -- 6.2.1 Checking for Balanced Braces -- 6.2.2 Recognizing Strings in a Language -- 6.3 Using Stacks with Algebraic Expressions -- 6.3.1 Evaluating Postfix Expressions -- 6.3.2 Converting Infix Expressions to Equivalent Postfix Expressions -- 6.4 Using a Stack to Search a Flight Map -- 6.5 The Relationship Between Stacks and Recursion -- C++ Interlude 3 Exceptions -- C3.1 Background -- C3.1.1 A Problem to Solve -- C3.2 Assertions -- C3.3 Throwing Exceptions -- C3.4 Handling Exceptions -- C3.4.1 Multiple catch Blocks -- C3.4.2 Uncaught Exceptions -- C3.5 Programmer-Defined Exception Classes -- Chapter 7 Implementations of the ADT Stack -- 7.1 An Array-Based Implementation -- 7.2 A Link-Based implementation -- 7.3 Implementations That Use Exceptions -- Chapter 8 Lists -- 8.1 Specifying the ADT List -- 8.2 Using the List Operations -- 8.3 An Interface Template for the ADT List -- Chapter 9 List Implementations -- 9.1 An Array-Based Implementation of the ADT List -- 9.1.1 The Header File -- 9.1.2 The Implementation File -- 9.2 A Link-Based Implementation of the ADT List -- 9.2.1 The Header File -- 9.2.2 The Implementation File -- 9.2.3 Using Recursion in LinkedList Methods -- 9.3 Comparing Implementations -- Chapter 10 Algorithm Efficiency -- 10.1 What Is a Good Solution? -- 10.2 Measuring the Efficiency of Algorithms -- 10.2.1 The Execution Time of Algorithms -- 10.2.2 Algorithm Growth Rates -- 10.2.3 Analysis and Big O Notation

-- 10.2.4 Keeping Your Perspective -- 10.2.5 The Efficiency of Searching Algorithms -- Chapter 11 Sorting Algorithms and Their Efficiency -- 11.1 Basic Sorting Algorithms -- 11.1.1 The Selection Sort -- 11.1.2 The Bubble Sort -- 11.1.3 The Insertion Sort.
11.2 Faster Sorting Algorithms -- 11.2.1 The Merge Sort -- 11.2.2 The Quick Sort -- 11.2.3 The Radix Sort -- 11.3 A Comparison of Sorting Algorithms -- C++ Interlude 4 Class Relationships and Reuse -- C4.1 Inheritance Revisited -- C4.1.1 Public, Private, and Protected Sections of a Class -- C4.1.2 Public, Private, and Protected Inheritance -- C4.1.3 Is-a and As-a Relationships -- C4.2 Containment: Has-a Relationships -- C4.3 Abstract Base Classes Revisited -- Chapter 12 Sorted Lists and Their Implementations -- 12.1 Specifying the ADT Sorted List -- 12.1.1 An Interface Template for the ADT Sorted List -- 12.1.2 Using the Sorted List Operations -- 12.2 A Link-Based Implementation -- 12.2.1 The Header File -- 12.2.2 The Implementation File -- 12.2.3 The Efficiency of the Link-Based Implementation -- 12.3 Implementations That Use the ADT List -- 12.3.1 Containment -- 12.3.2 Public Inheritance -- 12.3.3 Private Inheritance -- Chapter 13 Queues and Priority Queues -- 13.1 The ADT Queue -- 13.2 Simple Applications of the ADT Queue -- 13.2.1 Reading a String of Characters -- 13.2.2 Recognizing Palindromes -- 13.3 The ADT Priority Queue -- 13.3.1 Tracking Your Assignments -- 13.4 Application: Simulation -- 13.5 Position-Oriented and Value-Oriented ADTs -- Chapter 14 Queue and Priority Queue Implementations -- 14.1 Implementations of the ADT Queue -- 14.1.1 An Implementation That Uses the ADT List -- 14.1.2 A Link-Based Implementation -- 14.1.3 An Array-Based Implementation -- 14.1.4 Comparing Implementations -- 14.2 An Implementation of the ADT Priority Queue -- C++ Interlude 5 Overloaded Operators and Friend Access -- C5.1 Overloaded Operators -- C5.1.1 Overloading = for Assignment -- C5.1.2 Overloading + for Concatenation -- C5.2 Friend Access and Overloading &lt -- &lt -- -- Chapter 15 Trees -- 15.1 Terminology -- 15.1.1 Kinds of Trees.
15.1.2 The Height of Trees -- 15.1.3 Full, Complete, and Balanced Binary Trees -- 15.1.4 The Maximum and Minimum Heights of a Binary Tree -- 15.2 The ADT Binary Tree -- 15.2.1 Traversals of a Binary Tree -- 15.2.2 Binary Tree Operations -- 15.2.3 An Interface Template for the ADT Binary Tree -- 15.3 The ADT Binary Search Tree -- 15.3.1 Binary Search Tree Operations -- 15.3.2 Searching a Binary Search Tree -- 15.3.3 Creating a Binary Search Tree -- 15.3.4 Traversals of a Binary Search Tree -- 15.3.5 The Efficiency of Binary Search Tree Operations -- Chapter 16 Tree Implementations -- 16.1 The Nodes in a Binary Tree -- 16.1.1 An Array-Based Representation -- 16.1.2 A Link-Based Representation -- 16.2 A Link-Based Implementation of the ADT Binary Tree -- 16.2.1 The Header File -- 16.2.2 The Implementation -- 16.3 A Link-Based Implementation of the ADT Binary Search Tree -- 16.3.1 Algorithms for the ADT Binary Search Tree Operations -- 16.3.2 The Class BinarySearch Tree -- 16.4 Saving a Binary Search Tree in a File -- 16.5 Tree Sort -- 16.6 General Trees -- C++ Interlude 6 Iterators -- C6.1 Iterators -- C6.1.1 Common Iterator Operations -- C6.1.2 Using Iterator Operations -- C6.1.3 Implementing an Iterator -- C6.2 Advanced Iterator Functionality -- Chapter 17 Heaps -- 17.1 The ADT Heap -- 17.2 An Array-Based Implementation of a Heap -- 17.2.1 Algorithms for the Array-Based Heap Operations -- 17.2.2 The Implementation -- 17.3 A Heap Implementation of the ADT Priority Queue -- 17.4 Heap Sort -- Chapter 18 Dictionaries and Their Implementations -- 18.1 The ADT Dictionary -- 18.1.1 An Interface for the ADT Dictionary -- 18.2 Possible Implementations -- 18.2.1 A Sorted Array-Based Implementation of the ADT Dictionary -- 18.2.2 A

	Binary Search Tree Implementation of the ADT Dictionary -- 18.3 Selecting an Implementation -- 18.3.1 Four Scenarios -- 18.4 Hashing. 18.4.1 Hash Functions.
Sommario/riassunto	This classic, best selling data structures text provides a firm foundation in data abstraction that emphasizes the distinction between specifications and implementation as the basis for an object-oriented approach. Software engineering principles and concepts as well as UML diagrams are used to enhance student understanding.