22 Custom Generic Data Structures: Solutions

Much that I bound, I could not free;
Much that I freed returned to me.
—Lee Wilson Dodd

'Will you walk a little faster?' said a whiting to a snail,
'There's a porpoise close behind us, and he's treading on my tail.'
—Lewis Carroll

There is always room at the top.
—Daniel Webster

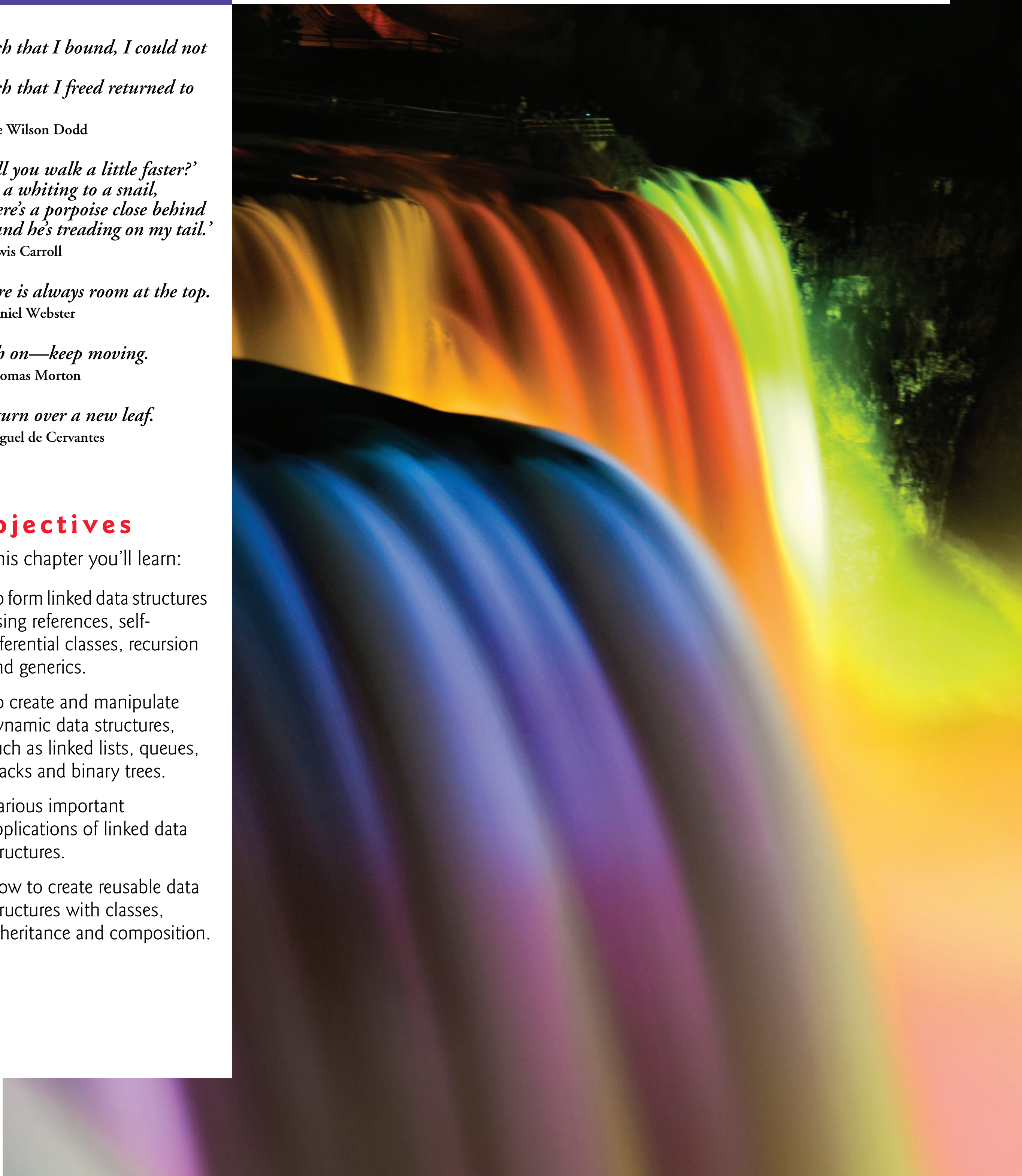
Push on—keep moving.
—Thomas Morton

I'll turn over a new leaf.
—Miguel de Cervantes

Objectives

In this chapter you'll learn:

- To form linked data structures using references, self-referential classes, recursion and generics.
- To create and manipulate dynamic data structures, such as linked lists, queues, stacks and binary trees.
- Various important applications of linked data structures.
- How to create reusable data structures with classes, inheritance and composition.



Self-Review Exercises

22.1 Fill in the blanks in each of the following statements:

a) A self-________ class is used to form dynamic data structures that can grow and shrink at execution time.

ANS: referential.

b) A(n) ________ is a constrained version of a linked list in which nodes can be inserted and deleted only from the start of the list.

ANS: stack.

c) A method that does not alter a linked list, but simply looks at it to determine whether it's empty, is referred to as a(n) ________ method.

ANS: predicate.

d) A queue is referred to as a(n) ________ data structure because the first nodes inserted are the first ones removed.

ANS: first-in, first-out (FIFO).

e) The reference to the next node in a linked list is referred to as a(n) ________.

ANS: link.

f) Automatically reclaiming dynamically allocated memory in Java is called ________.

ANS: garbage collection.

g) A(n) ________ is a constrained version of a linked list in which nodes can be inserted only at the end of the list and deleted only from the start of the list.

ANS: queue.

h) A(n) ________ is a nonlinear, two-dimensional data structure that contains nodes with two or more links.

ANS: tree.

i) A stack is referred to as a(n) ________ data structure because the last node inserted is the first node removed.

ANS: last-in, first-out (LIFO).

j) The nodes of a(n) ________ tree contain two link members.

ANS: binary.

k) The first node of a tree is the ________ node.

ANS: root.

l) Each link in a tree node refers to a(n) ________ or ________ of that node.

ANS: child or subtree.

m) A tree node that has no children is called a(n) ________ node.

ANS: leaf.

n) The three traversal algorithms we mentioned in the text for binary search trees are ________, ________ and ________.

ANS: inorder, preorder, postorder.

22.2 What are the differences between a linked list and a stack?

ANS: It's possible to insert a node anywhere in a linked list and remove a node from anywhere in a linked list. Nodes in a stack may be inserted only at the top of the stack and removed only from the top.

22.3 What are the differences between a stack and a queue?

ANS: A queue is a FIFO data structure that has references to both its head and its tail, so that nodes may be inserted at the tail and deleted from the head. A stack is a LIFO data structure that has a single reference to the stack's top, where both insertion and deletion of nodes are performed.

22.4 Perhaps a more appropriate title for this chapter would have been Reusable Data Structures. Comment on how each of the following entities or concepts contributes to the reusability of data structures:

a) classes

ANS: Classes allow us to instantiate as many data structure objects of a certain type (i.e., class) as we wish.

b) inheritance

ANS: Inheritance enables a subclass to reuse the functionality from a superclass. Public and protected methods of a superclass can be accessed through a subclass to eliminate duplicate logic.

c) composition

ANS: Composition enables a class to reuse code by storing a reference to an instance of another class in a field. Public methods of the instance can be called by methods in the class that contains the reference.

22.5 Manually provide the inorder, preorder and postorder traversals of the binary search tree of Fig. 22.20.

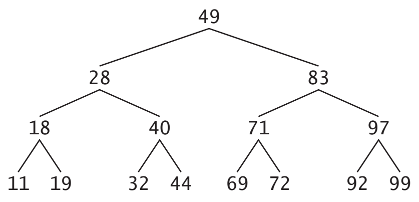


Fig. 22.20 | Binary search tree with 15 nodes.

ANS: The inorder traversal is

```
11 18 19 28 32 40 44 49 69 71 72 83 92 97 99
```

The preorder traversal is

```
49 28 18 11 19 40 32 44 83 71 69 72 97 92 99
```

The postorder traversal is

```
11 19 18 32 44 40 28 69 72 71 92 99 97 83 49
```

Exercises

NOTE: Solutions to the programming exercises are located in the `ch22solutions` folder. Each exercise has its own folder named `ex22_##` where ## is a two-digit number representing the exercise number. For example, exercise 22.17's solution is located in the folder `ex22_17`.

22.18 *(Duplicate Elimination)* In this chapter, we saw that duplicate elimination is straightforward when creating a binary search tree. Describe how you'd perform duplicate elimination when using only a one-dimensional array. Compare the performance of array-based duplicate elimination with the performance of binary-search-tree-based duplicate elimination.

ANS: First, sort the array—this makes all the duplicates be adjacent. Then, walk through the array with a source and destination index, copying only the first of a sequence of duplicates from the source to the destination. Once the source index reaches the end of the array, remove the elements after the destination index from the array. This yields the same efficiency as inserting them into a balanced binary search tree.

22.27 ***(Lists and Queues without Tail References)*** Our implementation of a linked list (Fig. 22.3) used both a `firstNode` and a `lastNode`. The `lastNode` was useful for the `insertAtBack` and `removeFromBack` methods of the `List` class. The `insertAtBack` method corresponds to the `enqueue` method of the `Queue` class.

Rewrite the `List` class so that it does not use a `lastNode`. Thus, any operations on the tail of a list must begin searching the list from the front. Does this affect our implementation of the `Queue` class (Fig. 22.13)?

ANS: The interface of the `List` class was not changed—just its implementation. Therefore, the `Queue` class does not need to be changed other than changing the `import` statements to point to the different `List` class. The performance of the `Queue` will be adversely affected because adding to the end of the `List` is now linear in the number of elements instead of constant time.

22.28 ***(Performance of Binary Tree Sorting and Searching)*** One problem with the binary tree sort is that the order in which the data is inserted affects the shape of the tree—for the same collection of data, different orderings can yield binary trees of dramatically different shapes. The performance of the binary tree sorting and searching algorithms is sensitive to the shape of the binary tree. What shape would a binary tree have if its data were inserted in increasing order? in decreasing order? What shape should the tree have to achieve maximal searching performance?

ANS: If the data were inserted in increasing or decreasing order, the tree would consist of only right children or only left children, respectively. Essentially, the tree would be a sorted linked list. To achieve maximum performance, the depth of the tree should be minimized—in other words, all non-leaf nodes except for the last row should have two children.