16. Exercise 10 in Chapter 4 introduced the doubly linked chain. The analogy for a binary search tree is to maintain parent pointers in each binary node in addition to the pointers to the node's children. That is, every node except the root will have a pointer to its parent in the tree. This type of binary tree is called a *doubly linked binary tree*, and such a tree is a part of Figure 16-22. Write addition and removal operations for this tree.

17. A node in a general tree can have an arbitrary number of children.

a. Describe a C++ implementation of a general tree in which every node contains an array of child pointers. Write a recursive preorder traversal method for this implementation. What are the advantages and disadvantages of this implementation?

b. Consider the implementation of a general tree that is illustrated in Figure 16-19. Each node has two pointers: The left pointer points to the node's oldest child and the right pointer points to the node's next sibling. Write a recursive preorder traversal method for this implementation.

c. Every node in a binary tree *T* has at most two children. Compare the oldest-child/next-sibling representation of *T*, as part *b* describes, to the left-child/right-child representation of a binary tree, as this chapter describes. Does one representation simplify the implementation of the ADT operations? Are the two representations ever the same?

18. Given an unbalanced binary search tree, use an inorder traversal to copy its data to an array. Then create a balanced binary search tree using the `readTree` algorithm given in Section 16.4, but use your array instead of a file.

***19.** Add an overloaded == operator to the class `BinaryNodeTree`.

Programming Problems

1. Complete the implementation of the class `BinaryNodeTree` that was begun in Section 16.2.2 of this chapter.

2. Implement the class `BinarySearchTree`, as given in Listing 16-4.

3. Write an array-based implementation of the ADT binary tree that uses dynamic memory allocation. Use a data structure like the one in Figure 16-1.

4. Repeat the previous problem, but define a binary search tree instead.

5. Write a program that maintains a database containing data, such as name and birthday, about your friends and relatives. You should be able to enter, remove, modify, or search this data. Initially, you can assume that the names are unique. The program should be able to save the data in a file for use later.

Design a class to represent the database and another class to represent the people. Use a binary search tree of people as a data member of the database class.

You can enhance this problem by adding an operation that lists everyone who satisfies a given criterion. For example, you could list people born in a given month. You should also be able to list everyone in the database.

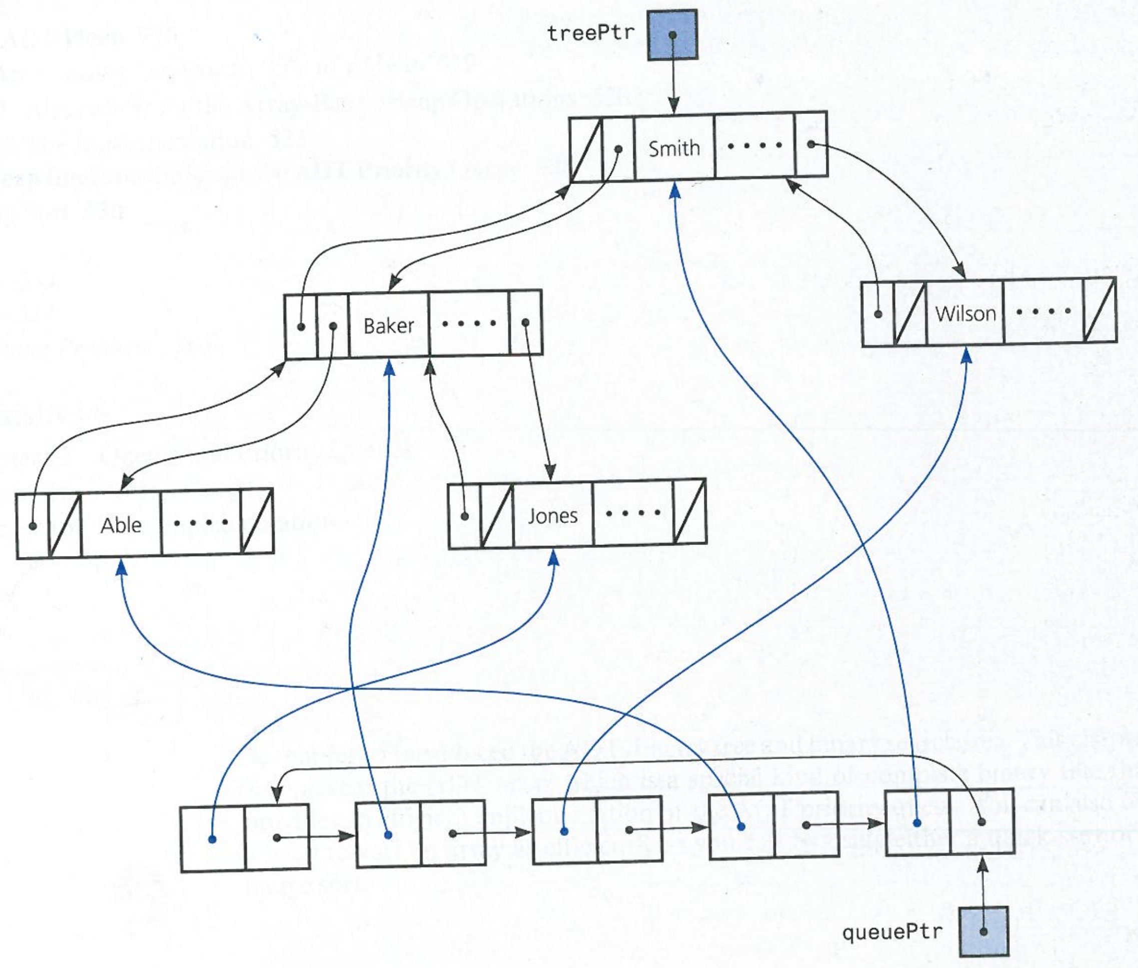
***6.** Many applications require a data organization that simultaneously supports several different data-management tasks. One simple example involves a waiting list of customers, that is, a queue of customer records. In addition to requiring the standard queue operations `isEmpty`, `enqueue`, `dequeue`, and `getFront`, suppose that the application frequently requires a listing of the customer records in the queue. This listing is more useful if the records appear sorted by customer name. You thus need a traverse operation that visits the customer records in sorted order.

This scenario presents an interesting problem. If you simply store the customer records in a queue, they will not, in general, be sorted by name. If, on the other hand, you just store the records in sorted order, you will be unable to process the customers on a first-come, first-served basis. Apparently, this problem requires you to organize the data in two different ways.

One solution is to maintain two independent data structures, one organized to support the sorted traversal and the other organized to support the queue operations. Figure 16-22 depicts a sorted linked list of customer records and a pointer-based implementation of the queue. The pointer-based data structures are a good choice because they do not require a good estimate of the maximum number of customer records that must be stored.

Implement the ADT queue operations as well as a sorted traversal operation for a queue that points into a doubly linked binary search tree, as shown in Figure 16-22. Doubly linked binary trees are explained in Exercise 16. You will need the addition and removal operations for a binary search tree that contains parent pointers.

FIGURE 16-22 A queue that points into a doubly linked binary search tree


treePtr
Smith
Baker
Wilson
Able
Jones
queuePtr