7. The specifications of the ADTs list and sorted list do not mention the case in which two or more items have the same value. Are these specifications sufficient to cover this case, or must they be revised?

8. Specify operations that are a part of the ADT character string. Include typical operations such as length computation and concatenation (appending one string to another).

## Exercises

1. Consider an ADT list of integers. Write a method that computes the maximum of the integers in the list `aList`. The definition of your method should be independent of the list's implementation.

2. Implement the method `swap`, as described in Self-Test Exercise 2, but remove the assumption that the $i^{th}$ and $j^{th}$ items in the list exist. Throw an exception `ListIndexOutOfBoundsException` if $i$ or $j$ is out of range.

3. Use the method `swap` that you wrote in Exercise 2 to write a method that reverses the order of the items in a list `aList`.

4. The section "The ADT List" describes the methods `displayList` and `replace`. As given in this chapter, these operations exist outside of the ADT; that is, they are not operations of the ADT list. Instead, their implementations are written in terms of the ADT list's operations.

   a. What is an advantage and a disadvantage of the way that `displayList` and `replace` are implemented?

   b. What is an advantage and a disadvantage of adding the operations `displayList` and `replace` to the ADT list?

5. The ADT `Bag` is a group of items, much like what you might have with a bag of groceries. Note that the items in the bag are in no particular order and that the bag may contain duplicate items. Specify operations to put an item in the bag, remove the last item put in the bag, remove a random item from the bag, check how many items are in the bag, check to see if the bag is full or empty, and completely empty the bag.

6. Design and implement an ADT that represents a credit card. The data of the ADT should include the customer name, the account number, the next due date, the reward points, and the account balance. The initialization operation should set the data to client-supplied values. Include operations for a credit card charge, a cash advance, a payment, the addition of interest to the balance, and the display of the statistics of the account.

7. Specify operations that are a part of the ADT fraction. Include typical operations such as addition, subtraction, and reduce (reduce fraction to lowest terms).

8. Suppose you want to write a program to play the card game War. Create an ADT for a card, a second ADT for a deck of cards, and a third ADT for a hand. What operations will you need on each of these ADTs to play the game of War? Note that in the game of War, you must be able to determine the higher card (Ace is high), and the winner wins all of the cards in that round and places those cards at the bottom of his or her hand. When there is a tie, a "war" is dealt with three cards face down, then the fourth face up, and again the winner wins all the cards. If there is a tie again, "war" is played again until the

tie is broken. If a player runs out of cards in his or her hand, that last card is always played face up, even if it is in the middle of a "war."

9. Write pseudocode implementations of the operations of an ADT that represents a trapezoid. Include typical operations, such as setting and retrieving the dimensions of the trapezoid, finding the area and the perimeter of the trapezoid, and displaying the statistics of the trapezoid.

10. Implement in Java the pseudocode for the ADT trapezoid that you created in Exercise 9.

11. Write a pseudocode method in terms of the ADT appointment book, described in the section "Designing an ADT," for each of the following tasks:

    a. Change the purpose of the appointment at a given date and time.

    b. Display all the appointments for a given date.

    Do you need to add operations to the ADT to perform these tasks?

12. Consider the ADT polynomial—in a single variable $x$—whose operations include the following:

```
+degree():integer {query}
// Returns the degree of a polynomial.
+getCoefficient(in power:integer):integer
// Returns the coefficient of the x^power term.
+changeCoefficient(in newCoef:integer,
                   in power:integer)
// Replaces the coefficient of the x^power term
// with newCoef.
```

For this problem, consider only polynomials whose exponents are nonnegative integers. For example,

$p = 4x^5 + 7x^3 - x^2 + 9$

The following examples demonstrate the ADT operations on this polynomial.

`p.degree()` is 5 (the highest power of a term with a nonzero coefficient)

`p.getCoefficient(3)` is 7 (the coefficient of the $x^3$ term)

`p.getCoefficient(4)` is 0 (the coefficient of a missing term is implicitly 0)

`p.changeCoefficient(-3, 7)` produces the polynomial

$p = -3x^7 + 4x^5 + 7x^3 - x^2 + 9$

Using only the ADT operations provided, write statements to perform the following tasks:

a. Display the constant term (the coefficient for the $x^0$ term).

b. Change each coefficient in the polynomial by multiplying them by 5.

c. For a given polynomial such as $p = -3x^7 + 4x^5 + 7x^3 - x^2 + 9$, display the expression in the form `- 3x^7 + 4x^5 + 7x^3 - 1x^2 + 9`.

d. Change the polynomial to its derivative—for example, $p = -3x^7 + 4x^5 + 7x^3 - x^2 + 9$ becomes $p = -21x^6 + 20x^4 + 21x^2 - 2x^1$.

13. Design and implement an ADT for a *Playing Card* and a *Deck* of playing cards. Each *Playing Card* must keep track of its suit (heart, diamond, club, spade), rank (2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace), and value (2 through 10 are face value, Jack is 11, Queen is 12, King is 13, and Ace is 14). The *Deck* of cards is all 52 cards—assume no Jokers will be used. The *Deck* ADT should initialize the 52 cards by suit (hearts, diamonds, clubs, spades) and within each suit as 2 through Ace. Operations for the Deck ADT should minimally include shuffling the deck and dealing a card from the deck.

14. Imagine an unknown implementation of an ADT sorted list of integers. This ADT organizes its items into ascending order. Suppose that you have just read $N$ integers into a one-dimensional array of integers called `data`. Write some Java statements that use the operations of the ADT sorted list to sort the array into ascending order.

15. Use the axioms for the ADT list, as given in this chapter in the section "Axioms," to prove that the sequence of operations

```
Insert A into position 2
Insert B into position 3
Insert C into position 2
```

has the same effect on a nonempty list of characters as the sequence

```
Insert B into position 2
Insert A into position 2
Insert C into position 2
```

16. Define a set of axioms for the ADT sorted list and use them to prove that the sorted list of characters, which is defined by the sequence of operations

```
Create an empty sorted list
Insert S
Insert T
Insert R
Delete T
```

is exactly the same as the sorted list defined by the sequence

```
Create an empty sorted list
Insert T
Insert R
Delete T
Insert S
```

17. Repeat Exercise 20 in Chapter 3, using a variation of the ADT list to implement a nonrecursive version of the method $f(n)$.

18. Write pseudocode that merges two sorted lists into a new third sorted list by using only operations of the ADT sorted list.

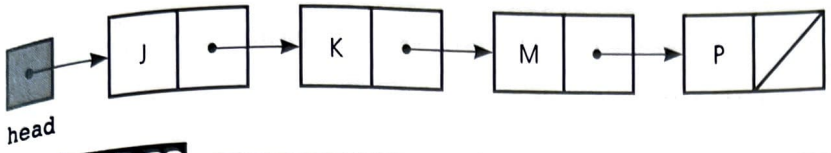


**FIGURE 5-32**

Linked list for Exercise 1

5. There were many types of linked lists discussed in the chapter. What was common in the nodes used in all of these classes?

6. How many assignment operations does the method that you wrote for Self-Test Exercise 5 require?

7. Do a box trace of `writeBackwards2(head)`, where `head` references the linked list of characters pictured in Figure 5-31. Show which node `head` points to in each recursive call. The method `writebackwards2` appears on page 273 of this chapter.

## Exercises

1. For each of the following, write the Java statements that perform the requested operation on the list shown in Figure 5-32. Also draw a picture of the status of the list after each operation is complete. When you delete a node from the list, make sure it will eventually be returned to the system. All insertions into the list should maintain the list's sorted order. Do not use any of the methods that were presented in this chapter.

   a. Assume that `prev` references the first node and `curr` references the second node. Insert L into the list.

   b. Assume that `prev` references the second node and that `curr` references the third node of the list after you revised it in Part *a*. Delete the last node of the list.

   c. Assume that `prev` references the last node of the list after you revised it in Part *b*, and assume that `curr` is `null`. Insert Q into the list.

2. Consider a linked list of items that are in no particular order.

   a. Write a method that inserts a node at the beginning of the linked list and a method that deletes the first node of the linked list.

   b. Repeat Part *a*, but this time perform the insertion and deletion at the end of the list instead of at the beginning. Assume the list has only a head reference.

   c. Repeat Part *b*, but this time assume that the list has a tail reference as well as a head reference.

3. Write a method that randomly removes and returns an item from a linked list. Write the method such that

   a. the method uses only the ADT List operations; that is, it is independent of the list's implementation.

b. the method assumes and uses the reference-based implementation of the ADT List.

c. the method assumes and uses the array-based implementation of the ADT List.

4. Given the following `Student` class:

```
class Student {
  private String name;
  private int age;

  public Student(String n, int a) {
    name = n;
    age = a;
  } // end constructor

  public String getName() {
    return name;
  } // end getName

  public int getAge() {
    return age;
  } // end getAge
} // end Student
```

Write a Java method that displays only the name of the $i^{th}$ student in a linked list of students. Assume that $i \geq 0$ and that the linked list contains at least i nodes.

5. Using the `Student` class in Exercise 4, write a recursive version of the method that displays only the name of the $i^{th}$ student in a linked list of students. Assume that $i \geq 1$ and that the linked list contains at least $i$ nodes. (Hint: If $i = 0$, print the name of the first student in the list; otherwise, print the $(i-1)^{th}$ student name from the rest of the list.)

6. The section "Processing Linked Lists Recursively" discussed the traversal of a linked list.

   a. Compare the efficiencies of an iterative method that displays a linked list with the method `writeList`.

   b. Write an iterative method that displays a linked list backward. Compare the efficiencies of your method with the method `writeListBackward2`.

7. Write a method to merge two linked lists of integers that are sorted into descending order. The result should be a third linked list that is the sorted combination of the original lists. Do not destroy the original lists.

8. The `Node` class presented in this chapter assumed that it would be declared package-private; hence, the data fields were declared for package access only.

   a. Suppose that the data fields were declared `private`. Write accessor and mutator methods for both the `item` and `next` fields.

   b. Give at least three different examples of how the code in the `ListReferenceBased` implementation would have to be changed.

9. Assume that the reference `list` references the last node of a circular linked list like the one in Figure 5-24. Write a loop that searches for an item in the list and if

found, returns its position. If it is not found, return –1. Assume that the node referenced by `list.next` is the node in the first position.

10. Write the pseudocode for a method that inserts a new node to the end of a doubly linked list.

11. Write the method `add(int index, Object item)` from the interface `ListInterface` for a doubly linked list with a head pointer as depicted in Figure 5-26.

12. Write a class called `SantasList` that allows Santa to keep track of all of the children that are naughty and nice. Santa should be able to add children to either of the lists, and the lists can be maintained in any order. Also provide methods to print each list.

    a. Implement `SantasList` using your own linked list.

    b. Implement `SantasList` using the `ListReferenceBased` class.

    c. What changes would you need to make if you wanted to change the implementation in part b to use the `ListArrayBased` class?

13. Given two circular linked lists, one referenced by $p$, the other by $q$, append list $q$ to list $p$. This should result in a circular list reference by $p$ that contains all of the elements of the original list referenced by $p$ followed by the elements of the list referenced by $q$.

14. Imagine a circular linked list of integers that are sorted into ascending order, as Figure 5-33a illustrates. The external reference `list` references the last node, which contains the largest integer. Write a method that revises the list so that its data elements are sorted into descending order, as Figure 5-33b illustrates. Do not allocate new nodes.

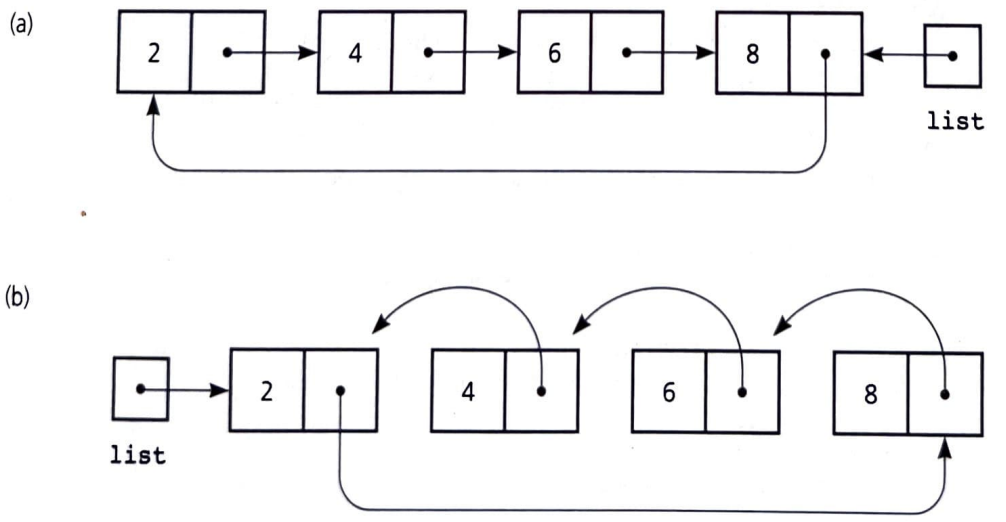


**FIGURE 5-33**

Two circular linked lists

15. Revise the implementations of the ADT list operations *add*, *get*, and *remove* on pages 267–268 under the assumption that the linked list has a dummy head node.

16. Add the operations *save* and *restore* to the reference-based implementation of the ADT list. These operations have a file parameter that indicates the file to save and restore the list items.

17. Consider the sorted doubly linked list shown in Figure 5-27. This list is a circular doubly linked list and has a dummy head node. Write methods for the following operation for a sorted list:

```
+sortedAdd(in item:ListItemType)
// Inserts item into its proper sorted position in a
// sorted list.

+sortedRemove(in item:ListItemType)
// Deletes item from a sorted list.
// Throws an exception if the item is not found.
```

18. Repeat Exercise 17 for the sorted doubly linked list shown in Figure 5-26. This list is not circular and does not have a dummy head node. Watch out for the special cases at the beginning and end of the list.

19. The class *java.util.Vector* implements a growable array of objects. Here is a subset of the methods available in the class *java.util.Vector*:

**Constructors**

```
Vector();
// Constructs an empty vector.

Vector(int initialCapacity, int capacityIncrement)
// Constructs an empty vector with the specified
// initial capacity and capacity increment.
```

**Methods**

```
void addElement(Object obj);
// Adds the specified component to the end of this
// vector, increasing its size by one.

int capacity();
// Returns the current capacity of this vector.

Object elementAt(int index);
// Returns the component at the specified index.

void removeElementAt(int index);
// Deletes the component at the specified index.

int size();
// Returns the number of components in this vector.
```

Complete the following tasks:

a. Declare a vector *first* with an initial capacity of 10 and a capacity increment of 5. Write a *for* loop that initializes all ten elements of the vector to the integers 1 through 10. What do the methods *size* and *capacity* return?

b. Add four more integers to the vector `first`. What do the methods size and `capacity` return now?

c. Write a loop to print all of the elements stored in the vector `first`.

d. Declare a vector `second` using the default constructor. What does `capacity` return in this case? Write a loop to add elements to the vector `second` so that this capacity is exceeded by one. What is the new capacity?

e. Delete all of the elements from the vector `second`. Does the capacity change? Why do you think the `Vector` class behaves this way?

20. You can have a linked list of linked lists, as Figure 5-30 indicates. Assume the Java definitions on page 288–289. Suppose that `curr` references a desired stock item (node) in the inventory list. Write some Java statements that add yourself as a customer to the end of the wait list associated with the node referenced by `curr`.

## Programming Problems

1. Chapter 4 introduced the ADT sorted list, which maintains its data in sorted order. For example, a sorted list of names would be maintained in alphabetical order, and a sorted list of numbers would be maintained in either increasing or decreasing order. The operations for a sorted list are summarized on page 210.

   Some operations—`sortedIsEmpty`, `sortedSize`, and `sortedGet`, for example—are just like those for the ADT list. Insertion and deletion operations, however, are by value, not by position as they are for a list. For example, when you insert an item into a sorted list, you do not specify where in the list the item belongs. Instead, the insertion operation determines the correct position of the item by comparing its value with those of the existing items on the list. A new operation, `locateIndex`, determines from the value of an item its numerical position within the sorted list.

   Note that the specifications given in Chapter 4 do not say anything about duplicate entries in the sorted list. Depending on your application, you might allow duplicates, or you might want to prevent duplicates from entering the list. For example, a sorted list of Social Security numbers probably should disallow duplicate entries. In this example, an attempt to insert a Social Security number that already exists in the sorted list would fail.

   Write a nonrecursive, reference-based implementation of the ADT sorted list of objects as a Java class `SortedListRefBased` such that

   a. Duplicates are allowed

   b. Duplicates are not allowed, and operations must prevent duplicates from entering the list

2. Repeat Programming Problem 1, but write a recursive, reference-based implementation instead. Recall from this chapter that the recursive methods must be in the private section of the class.

9. Execute the HPAir algorithm with the map in Figure 7-17 for the following requests. Show the state of the stack after each step.

   a. Fly from *F* to *I*.

   b. Fly from *F* to *C*.

   c. Fly from *H* to *C*.

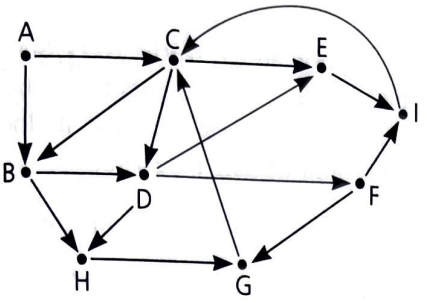


**FIGURE 7-17**
Flight map for Self-Test Exercise 9 and Exercise 15

## Exercises

1. What makes a linked list a good choice for implementing a stack?

2. Compare the operations in the ADT stack with the ADT list. Which operations are basically the same? What operations differ? Explain your answer.

3. Suppose that you have a stack `aStack` and an empty auxiliary stack `auxStack`. Show how you can do each of the following tasks by using only the operations of the ADT stack:

   a. Display the contents of `aStack` in reverse order; that is, display the top last.

   b. Count the number of items in `aStack`, leaving `aStack` unchanged.

   c. Delete every occurrence of a specified item from `aStack`, leaving the order of the remaining items unchanged.

4. Recall the search method from the JCF class `Stack`:

```
public int search(Object o)
// Returns the 1-based position where an object is on this
// stack. The topmost item on the stack is considered to be at
// distance 1.
```

   a. Add this method to the `StackListBased` implementation given in this chapter.

   b. Add this method to the `StackArrayBased` implementation given in this chapter.

   c. Add this method to the `StackReferenceBased` implementation given in this chapter.

5. An operation that displays the contents of a stack can be useful during program debugging. Add a `display` method to the ADT stack such that

   a. The method is a client that only uses only ADT stack operations; that is, it is independent of the stack's implementation.

   b. The method is within the ADT stack implementation and uses the reference-based implementation.

6. Another operation that could be added to the ADT Stack is one that removes and discards the user specified number of elements from the top of the stack. Assume this operation is called `popAndDiscard` and that it does not return a value and accepts a parameter called count of data type `int`.

   a. Add this operation to the `StackListBased` implementation given in this chapter.

   b. Add this operation to the `StackArrayBased` implementation given in this chapter.

   c. Add this operation to the `StackReferenceBased` implementation given in this chapter.

7. The diagram of a railroad switching system in Figure 7-18 is commonly used to illustrate the notion of a stack. Identify three stacks in the figure and show how they relate to one another. Suppose you had four train cars arrive in the order shown: A followed by B, then C, then D.

   a. How could you use the railroad switch to change the order so that A is still first, B is still second, but D is third, and C is fourth?

   b. How could you use the railroad switch to change the order of the cars so that B is first, D is second, A is third and C is fourth?

   c. Can any possible permutation of railroad cars be achieved with this switch? Justify your answer.

8. Suppose you have a stack in which the values 1 through 5 must be pushed on the stack in that order, but that an item on the stack can be popped and printed at any time. So for example, the operations

```
s.push(1)
s.push(2)
print s.pop()
```

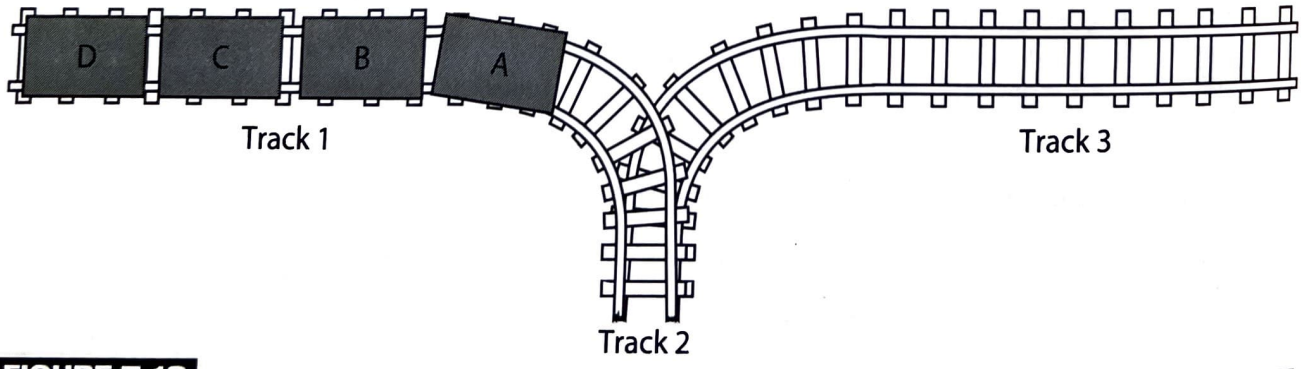


**FIGURE 7-18**
Railroad switching system for Exercise 4

```
s.push(3)
s.push(4)
print s.pop()
s.push(5)
print s.pop()
print s.pop()
print s.pop()
```

produce the sequence `2 4 5 3 1`. Based on the constraints mentioned above, give the list of operations that would produce each of the following sequences. If it is not possible, state so.

a. `1 3 5 4 2`

b. `2 3 4 5 1`

c. `1 4 3 5 2`

d. `1 5 4 2 3`

e. Are there sequences that cannot occur? Explain why or why not.

9. What is the output of the following code for `n = 15`? `n = 65`? What does this program appear to do?

```
public static void mystery(int n) {
  StackInterface s = new StackReferenceBased();
  while (n > 0) {
    s.push(n % 8);
    n = n / 8;
  }
  while (!s.isEmpty())
    System.out.print(stack.pop());
  System.out.println();
} // end mystery
```

10. Consider the stack implementation that uses the ADT list to represent the items in the stack. Discuss the efficiency of the stack's insertion and deletion operations when the ADT list's implementation is

    a. array based.

    b. reference based.

11. The section "Developing an ADT During the Design of a Solution" described an algorithm ***readAndCorrect*** that reads a string of characters, correcting mistakes along the way.

    a. For the following input line, trace the execution of ***readAndCorrect*** and show the contents of the stack at each step:

    `ab←cde←←fgh←i`

    b. The nature of the stack-based algorithm makes it simple to display the string in reverse order (as was done in ***displayBackward***), but somewhat harder to display it in its correct order. Write a pseudocode algorithm called ***displayForward*** that displays the string in its correct forward order.

    c. Implement ***readAndCorrect*** and ***displayForward*** as Java methods.

12. Revise the pseudocode solution to the balanced-braces problem so that the expression can contain three types of delimiters: `()`, `[]`, and `{}`. Thus, `{ab(c[d])e}` is valid, but `{ab(c))` is not.

13. For each of the following strings, trace the execution of the language-recognition algorithm described in the section "Recognizing Strings in a Language," and show the contents of the stack at each step.

    a. `xy$yxz`

    b. `xy$yx`

    c. `xyz$zxy`

    d. `zzz$zz`

    e. `xyzzy`

14. Write a pseudocode method that uses a stack to determine whether a string is in the language $L$, where

    a. $L = \{w : w$ contains equal numbers of A's and B's in any order$\}$

    b. $L = \{w : w$ is of the form $A^{2n}B^n$ for some $n \geq 0\}$

15. Write a method that uses a stack to determine whether a string is in the language $L$, where:

    $L = \{ww' : w$ is a string of characters
    $w' = reverse\ (w)\}$

    *Note:* The empty string, a string with less than 2 characters, or a string with an odd number of characters will not be in the language.

16. Evaluate the following postfix expressions by using the algorithm given in this chapter. Show the status of the stack after each step of the algorithm. Assume that division is integer division as in Java, and the identifiers have the following values: $a = 7$; $b = 3$; $c = 12$; $d = -5$; $e = 1$.

    a. *ab*cd+–*

    b. *abcd+–**

    c. *ab/c+d*e–*

17. Convert the following infix expressions to postfix form by using the algorithm given in this chapter. Show the status of the stack after each step of the algorithm.

    a. *a*b+c*

    b. *a/b*c*

    c. *a*(b+c)*

    d. *a+(b–c)*

    e. *a*(b+c–d)*

    f. *a/(b+c)/(d*e)*

4. For each of the following situations, which of these ADTs (1 through 4) would be most appropriate: (1) a queue; (2) a stack; (3) a list; (4) none of these?

   a. The customers at a deli counter who take numbers to mark their turn

   b. An alphabetic list of names

   c. Integers that need to be sorted

   d. The boxes in a box trace of a recursive method

   e. A grocery list ordered by the occurrence of the items in the store

   f. The items on a cash register tape

   g. A word processor that allows you to correct typing errors by using the backspace key

   h. A program that uses backtracking

   i. A list of ideas in chronological order

   j. Airplanes that stack above a busy airport, waiting to land

   k. People who are put on hold when they call an airline to make reservations

   l. An employer who fires the most recently hired person

   m. People who go to a store on Black Friday hoping to get the best holiday buys

5. In the bank simulation problem that this chapter discusses, why is it impractical to read the entire input file and create a list of all the arrival and departure events before the simulation begins?

6. Complete the hand trace of the bank-line simulation that Figure 8-20 began with the data given on page 436. Show the state of the queue and the event list at each step.

## Exercises

1. What makes a linked list a good choice for implementing a queue? Is an array an equally good choice? Explain your answer.

2. Compare the operations in the ADT stack with the ADT queue. Which operations are basically the same? What operations differ? Explain your answer.

3. Suppose you have a queue in which the values 1 through 5 must be enqueued on the queue in that order, but that an item on the queue can be dequeued and printed at any time. Based on these constraints, give the list of operations that would produce each of the following sequences. If it is not possible, state so.

   a. 1 3 5 4 2

   b. 1 2 3 4 5

   c. Are there sequences that cannot occur? Explain why or why not.

4. Implement the pseudocode conversion algorithm that converts a sequence of character digits in a queue to an integer (it is in the section "Reading a String of Characters"). Assume that you are using a queue to read in a series of characters that represent a correct postfix expression. The postfix expression has operators and multi-digit integers separated by single blanks. When the conversion method is called, the next item in the queue should be a character digit followed by zero or more character digits. The digits should be read until a non-digit character is found, and the resulting integer returned.

5. Consider the palindrome-recognition algorithm described in the section "Simple Applications of the ADT Queue." Is it necessary for the algorithm to look at the entire queue and stack? That is, can you reduce the number of times that the loop must execute?

6. Consider the language

$$L = \{w\$w' : w \text{ is a possibly empty string of characters other than \$}, \; w' = reverse(w)\}$$

as defined in Chapter 7. Write a recognition algorithm for this language that uses both a queue and a stack. Thus, as you traverse the input string, you insert each character of $w$ into a queue and each character of $w'$ into a stack. Assume that each input string contains exactly one \$.

7. What is output by the following code section?

```
QueueInterface aQueue = new QueueReferenceBased();
int num1, num2;
for (int i = 1; i <= 5; i++) {
  aQueue.enqueue(i);
} // end for

for (int i = 1; i <= 5; i++) {
  num1 = (Integer)aQueue.dequeue();
  num2 = (Integer)aQueue.dequeue();
  aQueue.enqueue(num1 + num2);
  aQueue.enqueue(num2 - num1);
} // end for

while(!aQueue.isEmpty()) {
  System.out.print(aQueue.dequeue() + " ");
} // end for
```

8. Assume you have a queue $q$ that has already been populated with data. What does the following code fragment do to the queue $q$?

```
Stack s = new Stack();
while (!q.isEmpty())
   s.push(q.dequeue());
while (!ss.isEmpty())
   q.enqueue(s.pop());
```

9. Another operation that could be added to the ADT Queue is one that removes and discards the user-specified number of elements from the front of the queue.

Assume this operation is called *dequeueAndDiscard* and that it does not return a value and accepts a parameter called *count* of data type `int`.

a. Add this operation to the *QueueListBased* implementation given in this chapter.

b. Add this operation to the *QueueArrayBased* implementation given in this chapter.

c. Add this operation to the *QueueReferenceBased* implementation given in this chapter.

10. The JCF class *Deque* had a method called *contains* that would return true if an *Object* was in the deque. For a queue, the method could be specified as follows:

```
public boolean contains(Object o)
// Returns true if this queue contains the specified element.
```

a. Add this method to the *QueueListBased* implementation given in this chapter.

b. Add this method to the *QueueArrayBased* implementation given in this chapter.

c. Add this method to the *QueueReferenceBased* implementation given in this chapter.

11. Revise the infix-to-postfix conversion algorithm of Chapter 7 so that it uses a queue to represent the postfix expression.

12. Consider the queue implementation that uses the ADT list to represent the items in the queue. Discuss the efficiency of the queue's insertion and deletion operations when the ADT list's implementation is

a. Array based

b. Reference based

13. An operation that displays the contents of a queue can be useful during program debugging. Add a *display* operation to the ADT queue such that

a. *display* uses only ADT queue operations, so it is independent of the queue's implementation

b. *display* assumes and uses the reference-based implementation of the ADT queue

14. Write a client method that returns the last element of a queue, while leaving the queue unchanged. This method can call any of the methods of the queue interface. It can also declare new queue objects. The return type of the method is *Object* and the method accepts a queue object (either a *QueueArrayBased* object or a *QueueListBased* object) as a parameter.

15. The Java Collections Framework provides a class called *ArrayDeque* that is an implementation of the *Deque* interface presented in this chapter. Use the class *ArrayDeque* to solve the read-and-correct problem presented in the "Developing an ADT During the Design of a Solution" section of Chapter 7. In that problem, you enter text at a keyboard and correct typing mistakes by using the backspace key. Each backspace erases the most recently entered character. Your solution should provide a corrected string of characters in the order in which they were entered at the keyboard.

## Exercises

1. The following recursive method *getNumberEqual* searches the array *x* of *n* integers for occurrences of the integer *val*. It returns the number of integers in *x* that are equal to *val*. For example, if *x* contains the 9 integers 1, 2, 4, 4, 5, 6, 7, 8, and 9, then *getNumberEqual(x, 9, 4)* returns the value 2 because 4 occurs twice in *x*.

```
public static int getNumberEqual(int x[], int n, int val) {
  if (n <= 0) {
    return 0;
  }
  else {
    if (x[n-1] == val) {
      return getNumberEqual(x, n-1, val) + 1;
    }
    else {
      return getNumberEqual(x, n-1, val);
    } // end if
  } // end if
} // end getNumberEqual
```

   Demonstrate that this method is recursive by listing the criteria of a recursive solution and stating how the method meets each criterion.

2. Perform a box trace of the following calls to recursive methods that appear in this chapter. Clearly indicate each subsequent recursive call.

   a. *rabbit(4)*

   b. *writeBackward("loop")* (Use the version that strips the last character.)

   c. *maxArray* Find the maximum element in the array {4, 10, 12, 1, 8, 3, 6, 9}

   d. *kSmall* Search for the 3rd smallest element in the array {4, 10, 12, 1, 8, 3, 6, 9}

3. Write a Java method and an accompanying main program for the problem of *Organzing a Parade* as presented in this chapter. Test your code with *n = 5*. Does this return the same value as *rabbit(5)*?

4. Given two integers *start* and *end*, where *end* is greater than *start*, write a recursive Java method that returns the sum of the integers from *start* through *end*, inclusive.

5. Add output code to the Spock method *c(n, k)* that shows the actual sequence of calls that are made and the value that they will return when the method is executed. For example, *c(3, 2)* outputs the following:

```
c(3, 2) = c(2, 1) + c(2, 2)
c(2, 1) = c(1, 0) + c(1, 1)
c(1, 0) = 1
c(1, 1) = 1
c(2, 2) = 1
```

   Use your modified version to run *c(4,2)* to show the actual order that the methods are called in Figure 3-12.

6. Given the following recursive method, answer each of the following questions.

```
public static void countDownByTwo(int n) {
  if (n != 1) {
    System.out.println(n + " ");
    countDownByTwo(n-2);
  } // end if
}  // end countDownByTwo
```

   a. What happens when you execute the method with $n = 7$?

   b. What happens when you execute the method with $n = 6$?

   c. Answer the four questions for constructive recursive solutions to prove or disprove the correctness of this recursive solution.

   d. If the answer to one or more of the questions in part c. indicates that this solution is incorrect, how would you change the method in such a way as to fix the problem?

7. The $n^{th}$ Harmonic number is the sum of the reciprocals of the first $n$ natural numbers:

$$H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}$$

   Write a recursive method to compute the $n^{th}$ Harmonic number.

8. Write a method called *minChar* that returns the minimum character (using the ASCII collating sequence) in a given string. So for example, *minChar("hello")* returns 'e.'

9. Write a recursive Java method that writes the digits of a positive decimal integer in reverse order.

10. a. Write a recursive Java method *writeLine* that writes a character repeatedly to form a line of $n$ characters. For example, *writeLine('*', 5)* produces the line *****.

    b. Now write a recursive method *writeBlock* that uses *writeLine* to write $m$ lines of $n$ characters each. For example, *writeBlock('*', 5, 3)* produces the output

```
*****
*****
*****
```

11. What output does the following program produce?

```
import java.util.Arrays;
public class Exercise11 {

  public static int guess(int[] c, int x) {
    if (c.length==1) {
      System.out.printf("z(%d) = %d\n", c.length-1, c[0]);
      return c[0];
    }
    else {
       System.out.printf("z(%d) = %d * z(%d) + %d\n",
                         c.length-1, x, c.length-2, c[0]);
```

```
        return
          x*guess(Arrays.copyOfRange(c, 1, c.length), x) + c[0];
      } // end if
    }  // end guess

    public static void main(String[] args) {
      int[] x = {2, 4, 1};
      System.out.println(guess(x, 5));
    } // end main
  } // end Exercise11
```

12. What output does the following program produce? Try running it with a couple of different values for *n*. Can you guess what this computes?

```
public class Exercise12 {

  private static int search(int a, int b, int n) {
    int returnValue;

    int mid = (a + b)/2;
    System.out.printf("Enter: a = %2d, b = %2d, mid = %2d\n",
                      a, b, mid);
    if ((mid * mid <= n) && (n < (mid+1) * (mid+1))) {
      returnValue = mid;
    }
    else if (mid * mid > n) {
      returnValue = search(a, mid-1, n);
    }
    else {
      returnValue = search(mid+1, b, n);
    }  // end if
    System.out.printf("Leave: a = %2d, b = %2d, mid = %2d\n",
                      a, b, mid);
   return returnValue;
  }  // end search

  public static void main(String[] args) {
    int n = 64;
    System.out.printf("For n = %2d, the result is %d\n",
                      n, search(1, n, n));
  }  // end main
}  //end Exercise12
```

13. Consider the following method that converts a positive decimal number to base 8 and displays the result.

```
public static void displayOctal(int n) {
  if (n > 0) {
      if (n/8 > 0) {
         displayOctal(n/8);
      }  // end if
      System.out.println(n%8);
  }  // end if
}  // end displayOctal
```

a. Trace the method with `n = 88`.

b. Describe how this method answers the four questions for constructing a recursive solution.

14. Consider the following program:

```
public class Exercise14 {

  public static int f(int n) {
  // Precondition: n >= 0.
    System.out.printf("Enter f: n = %d\n", n);
    switch (n) {
      case 1: case 2: case 3:
        return n + 1;
      default:
        return f(n-1) * f(n-3);
    }  // end switch
  }  // end f

  public static void main(String[] args) {
    System.out.println("f(8) is equal to " + f(8));
  } // end main
} // end Exercise14
```

Show the exact output of the program. What argument values, if any, could you pass to the method `f` to cause the program to run forever?

15. Consider the following method:

```
public static void recurse(int x, int y) {
  if (y > 0) {
    ++x;
    --y;
    System.out.println(x + " " + y);
    recurse(x, y);
    System.out.println(x + " " + y);
  }  // end if
}  // end recurse
```

Execute the method with $x = 5$ and $y = 3$.

16. Perform a box trace of the recursive method `binarySearch`, which appears in the section "Binary Search," with the array 2, 4, 5, 8, 9, 12, 15, 16, 20 for each of the following search values:

a. 5 b. 21 c. 32

17. Imagine that you have 101 dalmatians; no two dalmatians have the same number of spots. Suppose that you create an array of 101 integers: The first integer is the number of spots on the first dalmatian, the second integer is the number of spots on the second dalmatian, and so on.

Your friend wants to know whether you have a dalmatian with 99 spots. Thus, you need to determine whether the array contains the integer 99.

a. If you plan to use a binary search to look for the 99, what, if anything, would you do to the array before searching it?

b. What is the index of the integer in the array that a binary search would examine first?

c. If all your dalmatians have more than 99 spots, exactly how many comparisons will a binary search require to determine that 99 is not in the array?

18. This problem considers several ways to compute $x^n$ for some $n \geq 0$.

a. Write an iterative method `power1` to compute $x^n$ for $n \geq 0$.

b. Write a recursive method `power2` to compute $x^n$ by using the following recursive formulation:

$x^0 = 1$
$x^n = x * x^{n-1}$ if $n > 0$

c. Write a recursive method `power3` to compute $x^n$ by using the following recursive formulation:

$x^0 = 1$
$x^n = (x^{n/2})^2$ if $n > 0$ and $n$ is even
$x^n = x * (x^{n/2})^2$ if $n > 0$ and $n$ is odd

d. How many multiplications will each of the methods `power1`, `power2`, and `power3` perform when computing $3^{32}$? $3^{19}$?

e. How many recursive calls will `power2` and `power3` make when computing $3^{32}$? $3^{19}$?

19. Modify the recursive `rabbit` method so that it is visually easy to follow the flow of execution. Instead of just adding "Enter" and "Leave" messages, indent the trace messages according to how "deep" the current recursive call is. For example, the call `rabbit (4)` should produce the output

```
Enter rabbit:   n = 4
    Enter rabbit:   n = 3
        Enter rabbit:   n = 2
        Leave rabbit:   n = 2   value = 1
        Enter rabbit:   n = 1
        Leave rabbit:   n = 1   value = 1
    Leave rabbit:   n = 3   value = 2
    Enter rabbit:   n = 2
    Leave rabbit:   n = 2   value = 1
Leave rabbit:   n = 4   value = 3
```

Note how this output corresponds to figures such as Figure 3-11.

20. Consider the following recurrence relation:

$f(1) = 1; f(2) = 2; f(3) = 3; f(4) = 2; f(5) = 4;$
$f(n) = 2 * f(n-1) + f(n-5)$ for all $n > 5$.

a. Compute $f(n)$ for the following values of $n$: 6, 7, 10, 12.

b. What is the order of the sequential search algorithm when the desired item is not in the data collection? Do this for both sorted and unsorted data, and consider the best, average, and worst cases.

c. Show that if the sequential search algorithm finds the desired item in the data collection, the algorithm's order does not depend upon whether or not the data items are sorted.

5. Trace the selection sort as it sorts the following array into ascending order: 80 40 25 20 30 60.

6. Repeat Self-Test Exercise 5, but instead sort the array into descending order.

7. Trace the bubble sort as it sorts the following array into ascending order: 80 40 25 20 30 60.

8. Trace the insertion sort as it sorts the array in Self-Test Exercise 7 into ascending order.

9. Show that the mergesort algorithm satisfies the four criteria of recursion that Chapter 3 describes.

10. Trace quicksort's partitioning algorithm for an ascending sort as it partitions the following array. Use the first item as the pivot.

    39 12 16 38 40 27

11. Suppose that you sort a large array of integers by using mergesort. Next you use a binary search to determine whether a given integer occurs in the array. Finally, you display all the integers in the sorted array.

    a. Which algorithm is faster, in general: the mergesort or the binary search? Explain in terms of Big O notation.

    b. Which algorithm is faster, in general: the binary search or displaying the integers? Explain in terms of Big O notation.

## Exercises

1. What is the order of each of the following tasks in the worst case?

    a. Computing the sum of the first half of an array of $n$ items

    b. Initializing each element of an array `items` to 1

    c. Displaying every other integer in a linked list of $n$ nodes

    d. Displaying all $n$ names in a circular linked list

    e. Displaying the third element in a linked list

    f. Displaying the last integer in a linked list of $n$ nodes

    g. Searching an array of $n$ integers for a particular value by using a binary search

    h. Sorting an array of $n$ integers into ascending order by using a mergesort

2. Why do we include the variable `sorted` in the implementation of the bubble sort?

3. For queues and stacks presented earlier in this text, three implementations were provided—a reference based implementation, an array based implementation, and a list based implementation. For each of these implementations, what is the order of each of the following tasks in the worst case?

   a. Adding an item to a stack of n items

   b. Adding an item to a queue of n items

4. Find an array that makes the bubble sort exhibit its worst behavior.

5. Suppose that your implementation of a particular algorithm appears in Java as

```
for (int pass = 1; pass <= n; ++pass) {
  for (int index = 0; index < n; ++index) {
    for (int count = 1; count < 10; ++count) {
      . . .
    } // end for
  } // end for
} // end for
```

   The previous code shows only the repetition in the algorithm, not the computations that occur within the loops. These computations, however, are independent of $n$. What is the order of the algorithm? Justify your answer.

6. Consider the following Java method `f`. Do not be concerned with `f`'s purpose.

```
public static void f(int[] theArray, int n) {
 int temp;
  for (int j = 0; j < n; ++j) {
    int i = 0;
    while (i <= j) {
      if (theArray[i] < (theArray[j])) {
        temp = theArray[i];
        theArray[i] = theArray[j];
        theArray[j] = temp;
      } // end if
      ++i;
    } // end while
  } // end for
} // end f
```

   How many comparisons does `f` perform?

7. For large arrays and in the worst case, is selection sort faster than insertion sort? Explain.

8. Show that any polynomial $f(x) = c_n x^n + c_{n-1}x^{n-1} + \cdots + c_1 x + c_0$ is $O(x^n)$.

9. Show that for all constants $a$, $b > 1$, $f(n)$ is $O(\log_a n)$ if and only if $f(n)$ is $O(\log_b n)$. Thus, you can omit the base when you write $O(\log n)$. *Hint:* Use the identity $\log_a n = \log_b n / \log_b a$ for all constants $a$, $b > 1$.

10. This chapter's analysis of selection sort ignored operations that control loops or manipulate array indexes. Revise this analysis by counting *all* operations, and show that the algorithm is still $O(n^2)$.

11. Trace the insertion sort as it sorts the following array into ascending order:
80 40 25 20 30 60

12. Trace the selection sort as it sorts the following array into ascending order:
8 11 23 1 20 33

13. Trace the bubble sort as it sorts the following array into descending order:
10 12 23 34 5

14. Apply the selection sort, bubble sort, and insertion sort to

    a. An inverted array: 9 7 5 3 1

    b. An ordered array: 1 3 5 7 9

15. How many comparisons would be needed to sort an array containing 100 elements using the bubble sort in

    a. the worst case?

    b. the best case?

16. Write recursive versions of `selectionSort`, `bubbleSort`, and `insertionSort`.

17. One way computer speeds are measured is by the number of instructions they can perform per second. Assume that a comparison or a data move are each a single instruction. If the bubble sort is being used to sort 1,000,000 items in a worst case scenario, what is the approximate amount of time it takes to execute this sort on each of the following computers? Express your answer in days, hours, minutes, and seconds.

    a. An early computer that could only execute one thousand instructions per second

    b. A more recent computer that can execute one billion instructions per second

18. Trace the mergesort algorithm as it sorts the following array into ascending order. List the calls to `mergesort` and to `merge` in the order in which they occur.
80 40 25 20 30 60

19. When sorting an array by using mergesort,

    a. Do the recursive calls to `mergesort` depend on the values in the array, the number of items in the array, or both? Explain.

    b. In what step of `mergesort` are the items in the array actually swapped (that is, sorted)? Explain.

20. Trace the quicksort algorithm as it sorts the following array into ascending order. List the calls to `quicksort` and to `partition` in the order in which they occur.
80 40 25 20 30 60 15

21. Suppose that you remove the call to `merge` from the `mergesort` algorithm to obtain

```
+mystery(inout theArray:ItemArray,
         in first:integer, in last:integer)
// mystery algorithm for theArray[first..last].
```

```
if (first < last) {
  mid = (first + last) / 2
  mystery(theArray, first, mid)
  mystery(theArray, mid+1, last)
} // end if
```

What does this new algorithm do?

22. You can choose any array item as the pivot for `quicksort`. You then interchange items so that your pivot is in `theArray[first]`.

    a. One way to choose a pivot is to take the middle value of the three values `theArray[first]`, `theArray[last]`, and `theArray [(first + last)/2]`. How many comparisons are necessary to sort an array of size n if you always choose the pivot in this way?

    b. If the actual median value could be chosen as the pivot at each step, how many comparisons are necessary to sort an array of size $n$?

23. The partition algorithm that `quicksort` uses moves one item at a time from the unknown region into the appropriate region $S_1$ or $S_2$. If the item to be moved belongs in region $S_1$, and if $S_2$ is empty, the algorithm will swap an array item with itself. Modify the partition algorithm to eliminate this unnecessary swapping. Does this change the order of the algorithm?

24. Use invariants to show that the method `selectionSort` is correct.

25. Describe an iterative version of `mergesort`. Define an appropriate invariant and show the correctness of your algorithm.

26. One criterion used to evaluate sorting algorithms is stability. A sorting algorithm is **stable** if it does not exchange items that have the same sort key. Thus, items with the same sort key (possibly differing in other ways) will maintain their positions relative to one another. For example, you might want to take an array of students sorted by name and sort it by year of graduation. Using a stable sorting algorithm to sort the array by year will ensure that within each year the students will remain sorted by name. Some applications mandate a stable sorting algorithm. Others do not. Which of the sorting algorithms described in this chapter are stable?

27. When we discussed the radix sort, we sorted a hand of cards by first ordering the cards by rank and then by suit. To implement a radix sort for this example, you could use two characters to represent a card, if you used T to represent a 10. For example, S2 is the 2 of spades and HT is the 10 of hearts.

    a. Show a trace of the radix sort for the following cards:

    S2, HT, D6, S4, C9, CJ, DQ, ST, HQ, DK

    b. Suppose that you did not use T to represent a 10—that is, suppose that H10 is the 10 of hearts—and that you padded the two-character strings on the right with a blank to form three-character strings. How would a radix sort order the entire deck of cards in this case?