Summary

This chapter is a collection of theoretical material and implementation techniques. Rather than requiring the user to provide a file containing information about the items on the structure, C++ provides a way to provide this information when a structure is declared in a program. Templates are a C++ construct that allows the client to specify the type of the items to be on the structure in angle brackets beside the type name in the declaration statement.

The idea of linking the elements in a list has been extended to include lists with header and trailer nodes, circular lists, and doubly linked lists. The idea of linking the elements is a possibility to consider in the design of many types of data structures.

A shallow copy is a copy where items are copied but not the items to which they might point. A deep copy is a copy where items and the items to which they may point are copied. C++ provides a construct called a copy constructor, which can be used to force a deep copy. The relational operators can be overloaded, so that values of different types can be compared using the standard symbols. The assignment operator can also be overloaded.

In addition to using dynamically allocated nodes to implement a linked structure, we looked at a technique for implementing linked structures in an array of records. In this technique the links are not pointers into the free store but indexes into the array of records. This type of linking is used extensively in systems software.

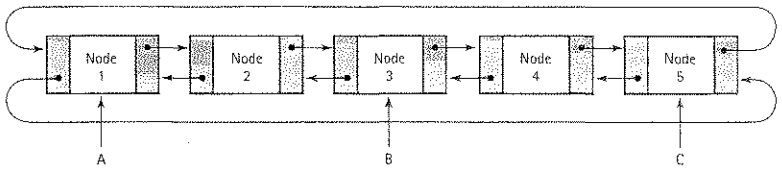
Polymorphism is revisited in this chapter with an example of how to use the C++ virtual function construct to implement dynamic binding. We also examined the concept of deep versus shallow copying and assignment operator overloading.

The Case Study at the end of the chapter designed a Large Integer ADT. The number of digits is bounded only by the size of memory. Several relational and arithmetic operators were overloaded to work with objects of this type.

Exercises

1. Dummy nodes are used to simplify list processing by eliminating some "special case."
   a. What special case is eliminated by a header node in a linear linked list?
   b. What special case is eliminated by a trailer node in a linear linked list?
   c. Would dummy nodes be useful in implementing a linked stack? That is, would their use eliminate a special case?
   d. Would dummy nodes be useful in implementing a linked queue with a pointer to both head and rear elements?
   e. Would dummy nodes be useful in implementing a circular linked queue?
2. Implement the class constructor, destructor, and copy constructor for the circular linked list class.

3. If you were going to implement the FIFO Queue ADT as a circular linked list, with the external pointer accessing the "rear" node of the queue, which member functions would you need to change?
4. Write a member function `PrintReverse` that prints the elements on a list in reverse order. For instance, for the list `X Y Z`, `list.PrintReverse()` would output `Z Y X`. The list is implemented as a circular list with `listData` pointing to the first element in the list. You may assume that the list is not empty.
5. Can you derive a type `DLList` from the class `SpecializedList` that has a member function `InsertItem` that inserts the item into its proper place in the list? If so, derive the class and implement the function. If not, explain why not.
6. If you were to rewrite the implementation of the Sorted List ADT using a doubly linked list, would you have to change the class definition? If so, how?
7. Outline the changes to the member functions that would be necessary to implement the Sorted List ADT as a doubly linked list.
8. Write a member function `Copy` of the Stack ADT, assuming that the stack named in the parameter list is copied into self.
9. Write a member function `Copy` of the Stack ADT, assuming that self is copied into the stack named in the parameter list.
10. Using the circular doubly linked list shown here, give the expression corresponding to each of the following descriptions.



(For example, the expression for the `info` member of Node 1, referenced from pointer A, would be `A->info`.)

   a. The `info` member of Node 1, referenced from pointer C
   b. The `info` member of Node 2, referenced from pointer B
   c. The `next` member of Node 2, referenced from pointer A
   d. The `next` member of Node 4, referenced from pointer C
   e. Node 1, referenced from pointer B
   f. The `back` member of Node 4, referenced from pointer C
   g. The `back` member of Node 1, referenced from pointer A

11. The text edited by a line editor is represented by a doubly linked list of nodes, each of which contains an 80-column line of text (type `LineType`). There is one external pointer (type `LineType*`) to this list, which points to the "current" line in the text being edited. The list has a header node, which contains the string "`- - - Top of File - - -`" and a trailer node, which contains the string "`- - - Bottom of File - - -`".

    a. Draw a sketch of this data structure.

    b. Write the type declarations to support this data structure.

    c. Write the class constructor, which sets up the header and trailer nodes.

    d. Code the following operations:

GoToTop(LineType* linePtr)

Function:	Goes to top of the list.
Postcondition:	currentLine is set to access the first line of text.

GoToBottom(LineType* linePtr)

Function:	Goes to bottom of the list.
Postcondition:	currentLine is set to access the last line of text.

    e. Describe the operations in part (d) in terms of Big-O notation. How could you change the list to make these operations O(1)?

    f. Code the `InsertLine` operation, using the following specification:

InsertLine(LinePtrType linePtr, LineType newLine)

Function:	Inserts newLine at the current line.
Postconditions:	newLine has been inserted after currentLine. currentLine points to newLine

    g. What other member functions should be included?

12. Of the three variations of linked lists (circular, with header and trailer nodes, and doubly linked), which would be most appropriate for each of the following applications?

    a. You want to search a list for a key and return the keys of the two elements that come before it and the keys of the two elements that come after it.

    b. A text file contains integer elements, one per line, *sorted* from smallest to largest. You must read the values from the file and create a sorted linked list containing the values.

    c. A list is short and frequently becomes empty. You want a list that is optimal for inserting an element into the empty list and deleting the last element from the list.

13. What is the Big-O measure for initializing the free list in the array-based linked implementation? For the functions `GetNode` and `FreeNode`?

14. Use the linked lists contained in the array pictured in Figure 6.19 to answer the following questions:

    a. What elements are in the list pointed to by `list1`?

    b. What elements are in the list pointed to by `list2`?

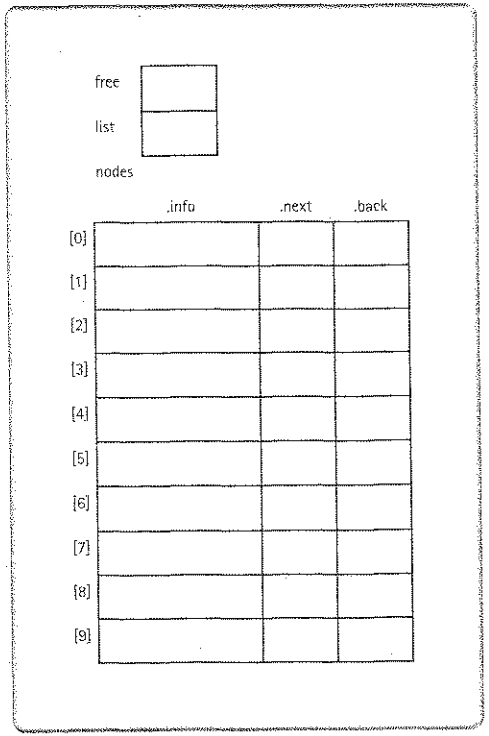
    c. What array positions (indexes) are part of the free space list?

    d. What would the array look like after the deletion of Nell from the first list?

    e. What would the array look like after the insertion of Anne into the second list? Assume that before the insertion the array is as pictured in Figure 6.19.

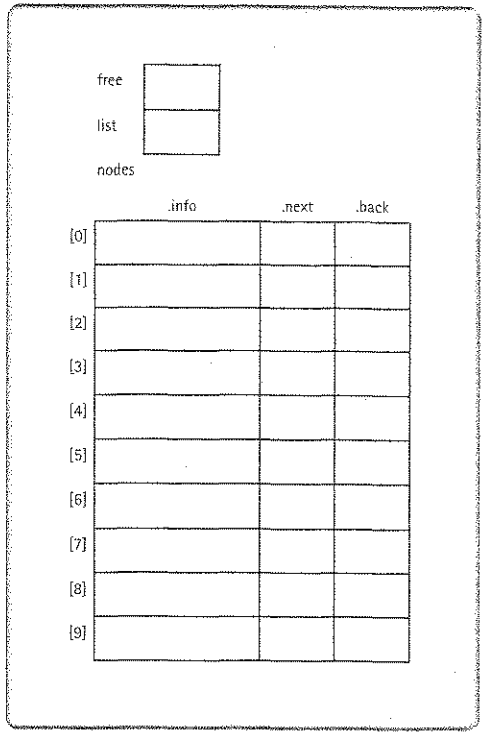
15. An array of records (nodes) is used to contain a doubly linked list, with the `next` and `back` members indicating the indexes of the linked nodes in each direction.

    a. Show how the array would look after it was initialized to an empty state, with all the nodes linked into the free-space list. (The free-space nodes have to be linked in only one direction.)

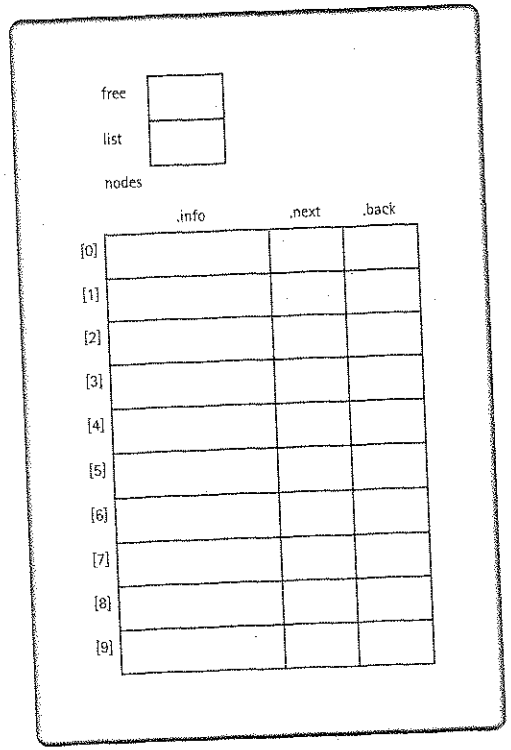


b. Draw a box-and-arrow picture of a doubly linked list into which the following numbers are inserted into their proper places: 17, 4, 25.

c. Fill in the contents of the array on the next page after the following numbers are inserted into their proper places in the doubly linked list: 17, 4, 25.



d. Show how the array in part (c) would look after 17 is deleted.



16. Discuss the changes that would be necessary if more than one digit is stored per node in the `LargeInt` class.
17. Distinguish between static and dynamic binding of functions.
18. Rewrite `SortedType` (array-based) using templates.
19. Rewrite `SortedType` (linked) using templates.
20. Rewrite `StackType` (linked) using templates.
21. Rewrite `QueType` (linked) using templates.
22. Replace function `ComparedTo` in an array-based `UnsortedList` by assuming that member functions of class `ItemType` overload the relational operators.
23. Create the UML diagrams for class `LargeInt`.

chapter 7

Programming with Recursion

After studying this chapter, you should be able to

- Discuss recursion as another form of repetition
- Do the following tasks, given a recursive routine:
  - Determine whether the routine halts
  - Determine the base case(s)
  - Determine the general case(s)
  - Determine what the routine does
  - Determine whether the routine is correct and, if it is not, correct it
- Do the following tasks, given a simple recursive problem:
  - Determine the base case(s)
  - Determine the general case(s)
  - Design and code the solution as a recursive void or value-returning function
- Verify a recursive routine, according to the Three-Question Method
- Decide whether a recursive solution is appropriate for a problem
- Compare and contrast dynamic storage allocation and static storage allocation in relation to using recursion
- Explain how recursion works internally by showing the contents of the run-time stack
- Replace a recursive solution with iteration and/or the use of a stack
- Explain why recursion may or may not be a good choice to implement the solution of a problem