Chapter 5

Linked Lists

5.1 INTRODUCTION

The everyday usage of the term "list" refers to a linear collection of data items. Figure 5-1(*a*) shows a shopping list; it contains a first element, a second element, . . . , and a last element. Frequently, we want to add items to or delete items from a list. Figure 5-1(*b*) shows the shopping list after three items have been added at the end of the list and two others have been deleted (by being crossed out).

(*a*) (*b*)

Fig. 5-1

Data processing frequently involves storing and processing data organized into lists. One way to store such data is by means of arrays, discussed in Chap. 4. Recall that the linear relationship between the data elements of an array is reflected by the physical relationship of the data in memory, not by any information contained in the data elements themselves. This makes it easy to compute the address of an element in an array. On the other hand, arrays have certain disadvantages—e.g., it is relatively expensive to insert and delete elements in an array. Also, since an array usually occupies a block of memory space, one cannot simply double or triple the size of an array when additional space is required. (For this reason, arrays are called *dense lists* and are said to be *static* data structures.)

Another way of storing a list in memory is to have each element in the list contain a field, called a *link* or *pointer*, which contains the address of the next element in the list. Thus successive elements in the list need not occupy adjacent space in memory. This will make it easier to insert and delete elements in the list. Accordingly, if one were mainly interested in searching through data for inserting and deleting, as in word processing, one would not store the data in an array but rather in a list using pointers. This latter type of data structure is called a *linked list* and is the main subject matter of this chapter. We also discuss circular lists and two-way lists—which are natural generalizations of linked lists—and their advantages and disadvantages.

5.2 LINKED LISTS

A *linked list*, or *one-way list*, is a linear collection of data elements, called *nodes*, where the linear order is given by means of *pointers*. That is, each node is divided into two parts: the first part contains the information of the element, and the second part, called the *link field* or *nextpointer field*, contains the address of the next node in the list.

Figure 5-2 is a schematic diagram of a linked list with 6 nodes, Each node is pictured with two parts. The left part represents the information part of the node, which may contain an entire record of data items (e.g., NAME, ADDRESS, . . .). The right part represents the nextpointer field of the node, and there is an arrow drawn from it to the next node in the list. This follows the usual practice of drawing an arrow from a field to a node when the address of the node appears in the given field. The pointer of the last node contains a special value, called the *null* pointer, which is any invalid address. (In actual practice, 0 or a negative number is used for the null pointer.) The null pointer, denoted by × in the diagram, signals the end of the list. The linked list also contains a *list pointer variable*—called

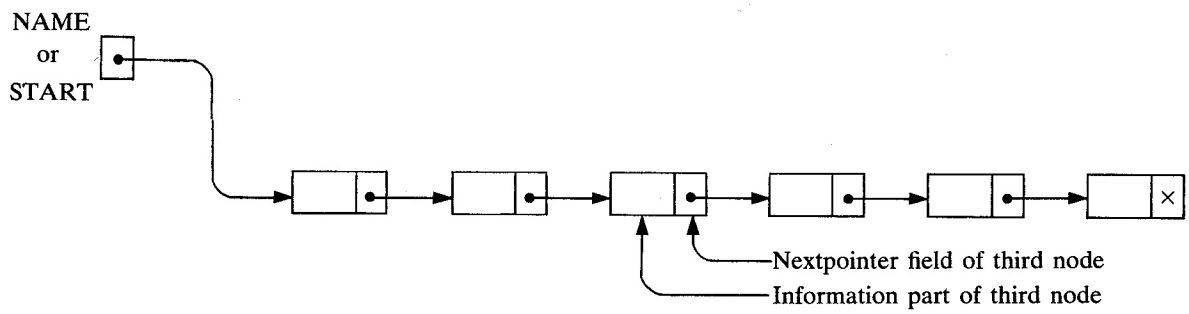

Fig. 5-2 Linked list with 6 nodes.

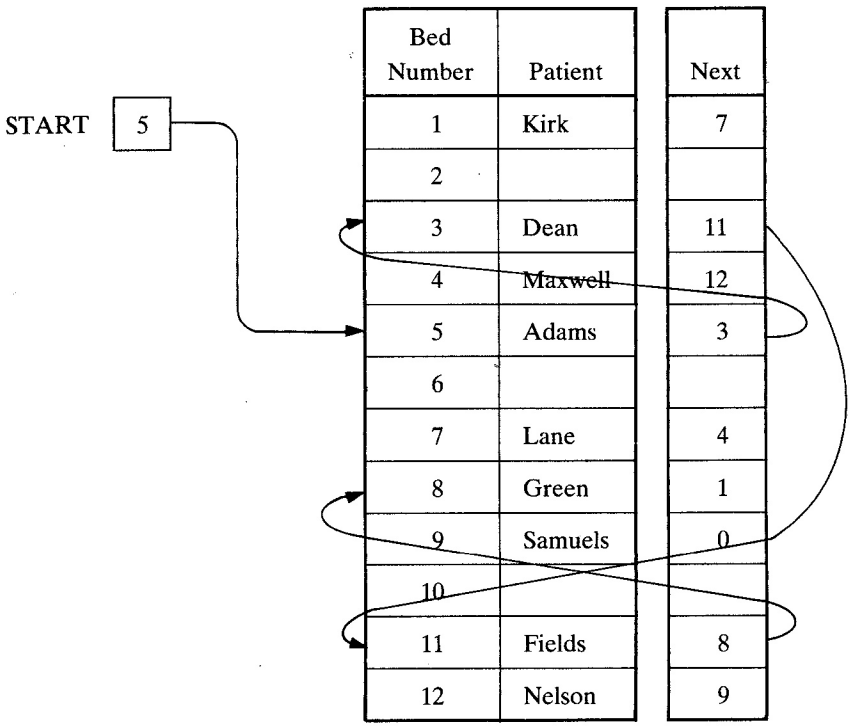

Fig. 5-3

START or NAME—which contains the address of the first node in the list; hence there is an arrow drawn from START to the first node. Clearly, we need only this address in START to trace through the list. A special case is the list that has no nodes. Such a list is called the *null list* or *empty list* and is denoted by the null pointer in the variable START.

EXAMPLE 5.1

A hospital ward contains 12 beds, of which 9 are occupied as shown in Fig. 5-3. Suppose we want an alphabetical listing of the patients. This listing may be given by the pointer field, called Next in the figure. We use the variable START to point to the first patient. Hence START contains 5, since the first patient, Adams, occupies bed 5. Also, Adams's pointer is equal to 3, since Dean, the next patient, occupies bed 3; Dean's pointer is 11, since Fields, the next patient, occupies bed 11; and so on. The entry for the last patient (Samuels) contains the null pointer, denoted by 0. (Some arrows have been drawn to indicate the listing of the first few patients.)

5.3 REPRESENTATION OF LINKED LISTS IN MEMORY

Let LIST be a linked list. Then LIST will be maintained in memory, unless otherwise specified or implied, by means of two parallel arrays, INFO and LINK, and a pointer variable START as follows. First of all, each node N of LIST will correspond to a subscript integer K such that:

(*a*) INFO[K] contains the data at the node N.

(*b*) LINK[K] contains the location of the node following node N.

Furthermore, START will contain the location of the first node of LIST. In addition, the LINK field of the last node of LIST will contain a sentinel value, denoted by NULL, which signals the end of LIST or, if LIST is empty, START will contain NULL. (Since the subscripts of the parallel arrays INFO and LINK will usually be positive, we will choose NULL = 0, unless otherwise stated.)

The following examples of linked lists indicate that the nodes of a list need not occupy adjacent elements in the arrays INFO and LINK, and that more than one list may be maintained in the same linear arrays INFO and LINK. However, each list must have its own pointer variable giving the location of its first node.

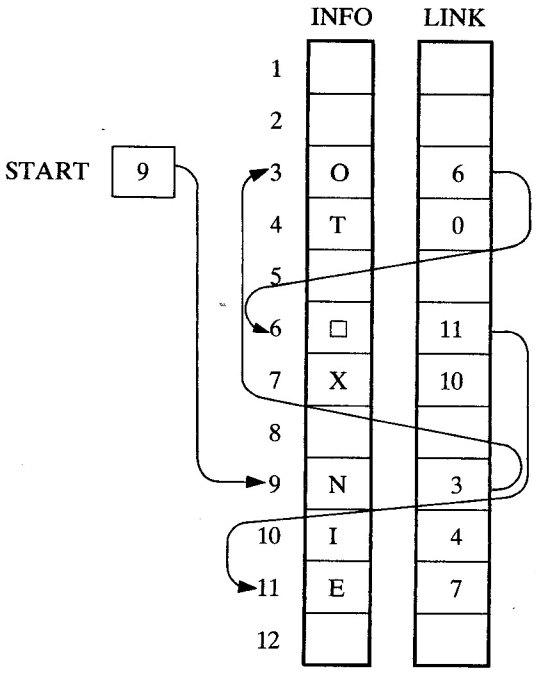

Fig. 5-4

EXAMPLE 5.2

Figure 5-4 pictures a linked list in memory where each node of the list contains a single character. We can obtain the actual list of characters, or, in other words, the string, as follows:

START = 9, so INFO[9] = N is the first character.
LINK[9] = 3, so INFO[3] = O is the second character.
LINK[3] = 6, so INFO[6] = □ (blank) is the third character.
LINK[6] = 11, so INFO[11] = E is the fourth character.
LINK[11] = 7, so INFO[7] = X is the fifth character.
LINK[7] = 10, so INFO[10] = I is the sixth character.
LINK[10] = 4, so INFO[4] = T is the seventh character.
LINK[4] = 0, the NULL value, so the list has ended.

In other words, NO EXIT is the character string.

EXAMPLE 5.3

Figure 5-5 pictures how two lists of test scores, here ALG and GEOM, may be maintained in memory where the nodes of both lists are stored in the same linear arrays TEST and LINK. Observe that the names of the lists are also used as the list pointer variables. Here ALG contains 11, the location of its first node, and GEOM contains 5, the location of its first node. Following the pointers, we see that ALG consists of the test scores

88, 74, 93, 82

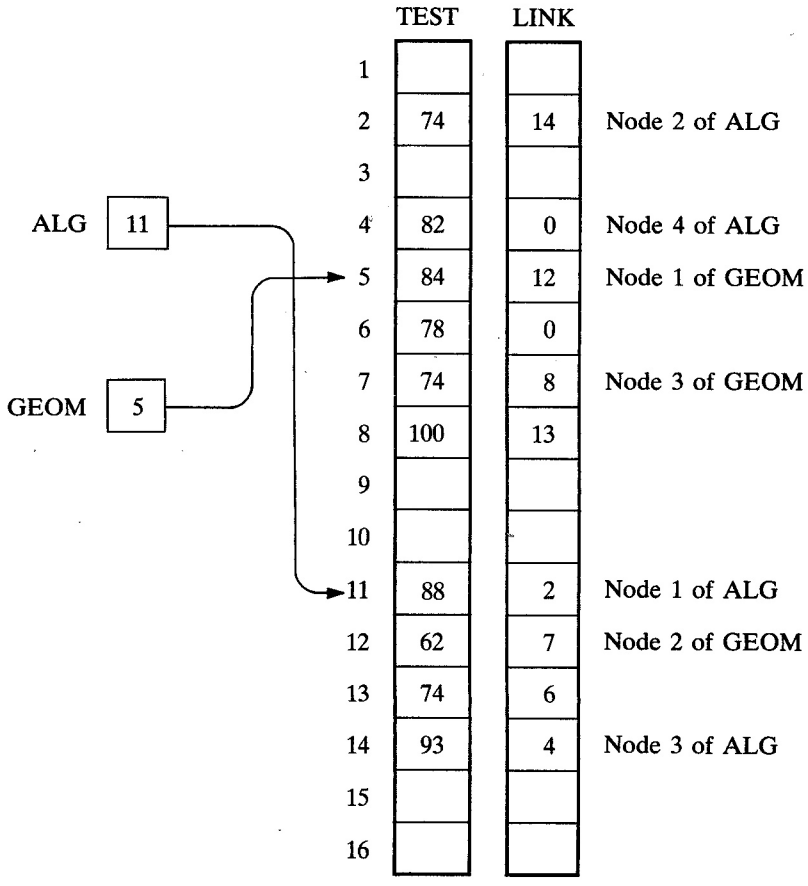

Fig. 5-5

and GEOM consists of the test scores

84, 62, 74, 100, 74, 78

(The nodes of ALG and some of the nodes of GEOM are explicitly labeled in the diagram.)

EXAMPLE 5.4

Suppose a brokerage firm has four brokers and each broker has his own list of customers. Such data may be organized as in Fig. 5-6. That is, all four lists of customers appear in the same array CUSTOMER, and an array LINK contains the nextpointer fields of the nodes of the lists. There is also an array BROKER which contains the list of brokers, and a pointer array POINT such that POINT[K] points to the beginning of the list of customers of BROKER[K].

Accordingly, Bond's list of customers, as indicated by the arrows, consists of

Grant, Scott, Vito, Katz

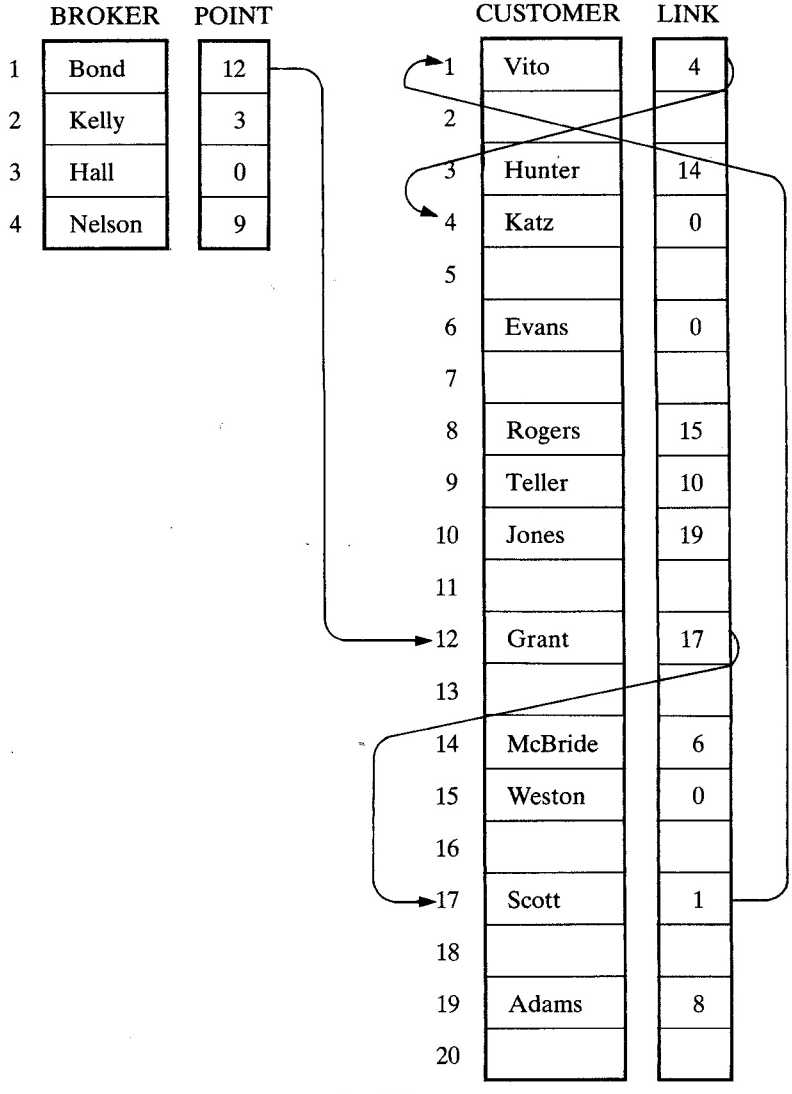

Fig. 5-6

Similarly, Kelly's list consists of

Hunter, McBride, Evans

and Nelson's list consists of

Teller, Jones, Adams, Rogers, Weston

Hall's list is the null list, since the null pointer 0 appears in POINT[3].

Generally speaking, the information part of a node may be a record with more than one data item. In such a case, the data must be stored in some type of record structure or in a collection of parallel arrays, such as that illustrated in the following example.

EXAMPLE 5.5

Suppose the personnel file of a small company contains the following data on its nine employees:

Name, Social Security Number, Sex, Monthly Salary

Normally, four parallel arrays, say NAME, SSN, SEX, SALARY, are required to store the data as discussed in Sec. 4.12. Figure 5-7 shows how the data may be stored as a sorted (alphabetically) linked list using only an additional array LINK for the nextpointer field of the list and the variable START to point to the first record in the list. Observe that 0 is used as the null pointer.

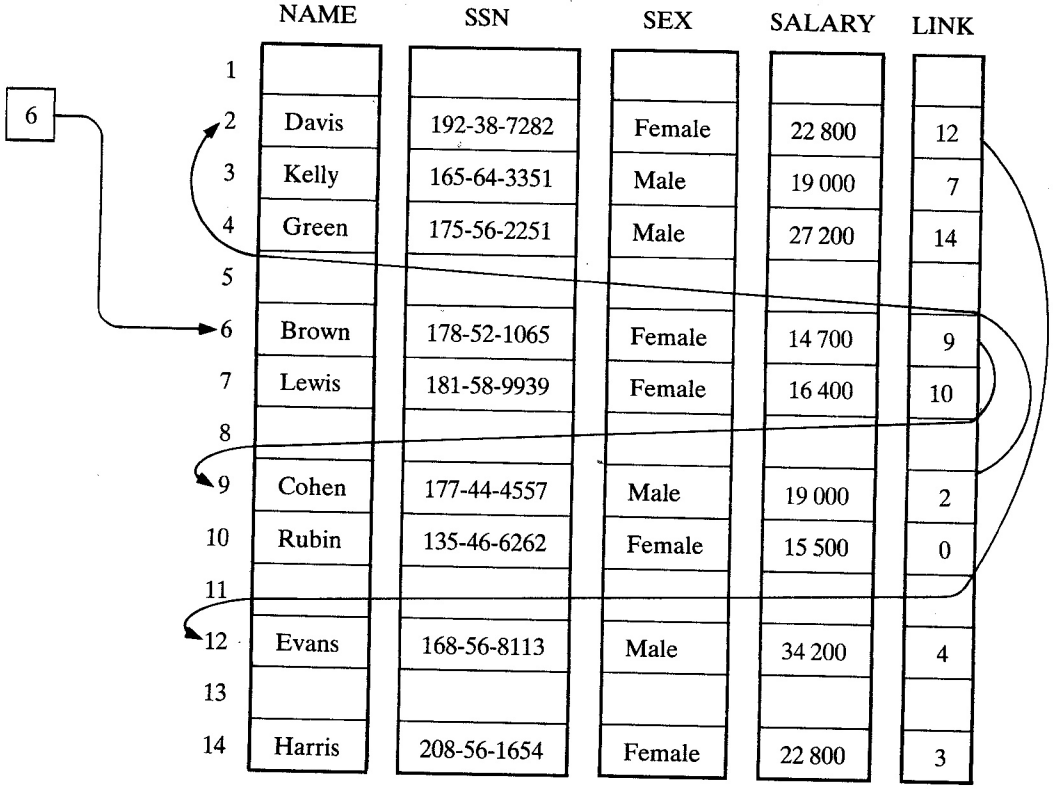

Fig. 5-7