Chapter 15
Lists

Objectives

- ❑ To introduce the basic concepts of linked lists
- ❑ To introduce the basic concepts of stacks
- ❑ To introduce the basic concepts of queues
- ❑ To introduce the basic concepts of tree structures
- ❑ To introduce the basic concepts of graph structures

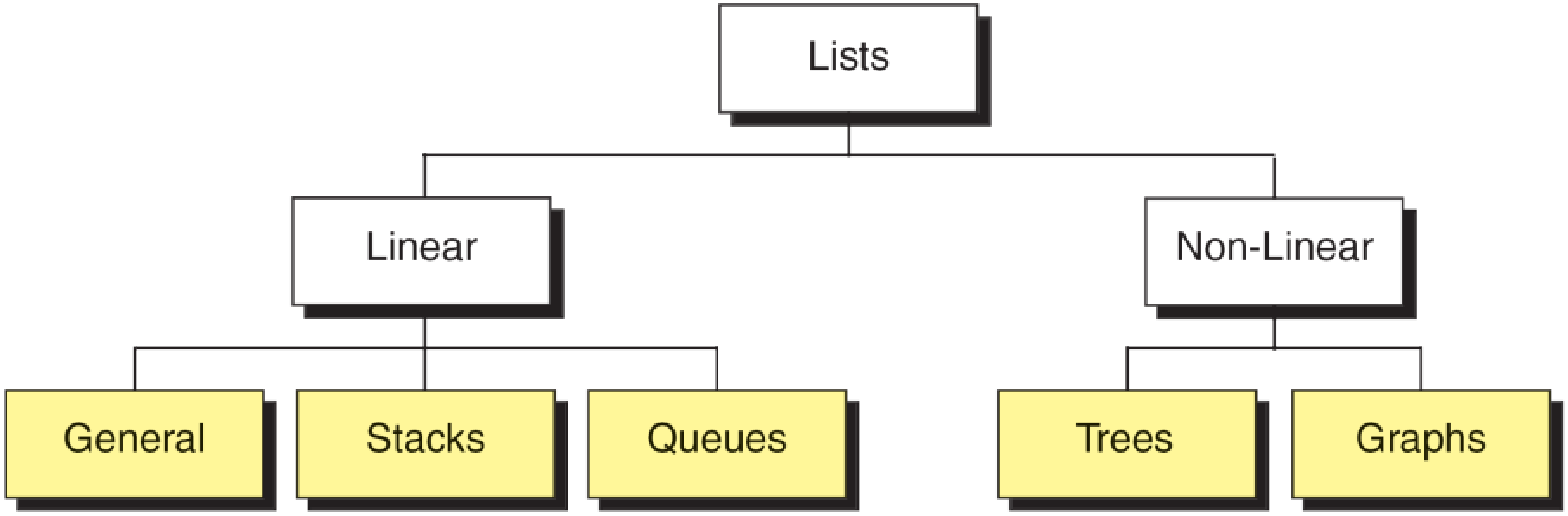


FIGURE 15-1 **Lists**

15-1 List Implementations

The C language does not provide any list structures or implementations. When we need them, we must provide the structures and functions for them. Traditionally, two data types, arrays and pointers, are used for their implementation.

Topics discussed in this section:

Array Implementation
Linked List Implementation
Pointers to Linked Lists

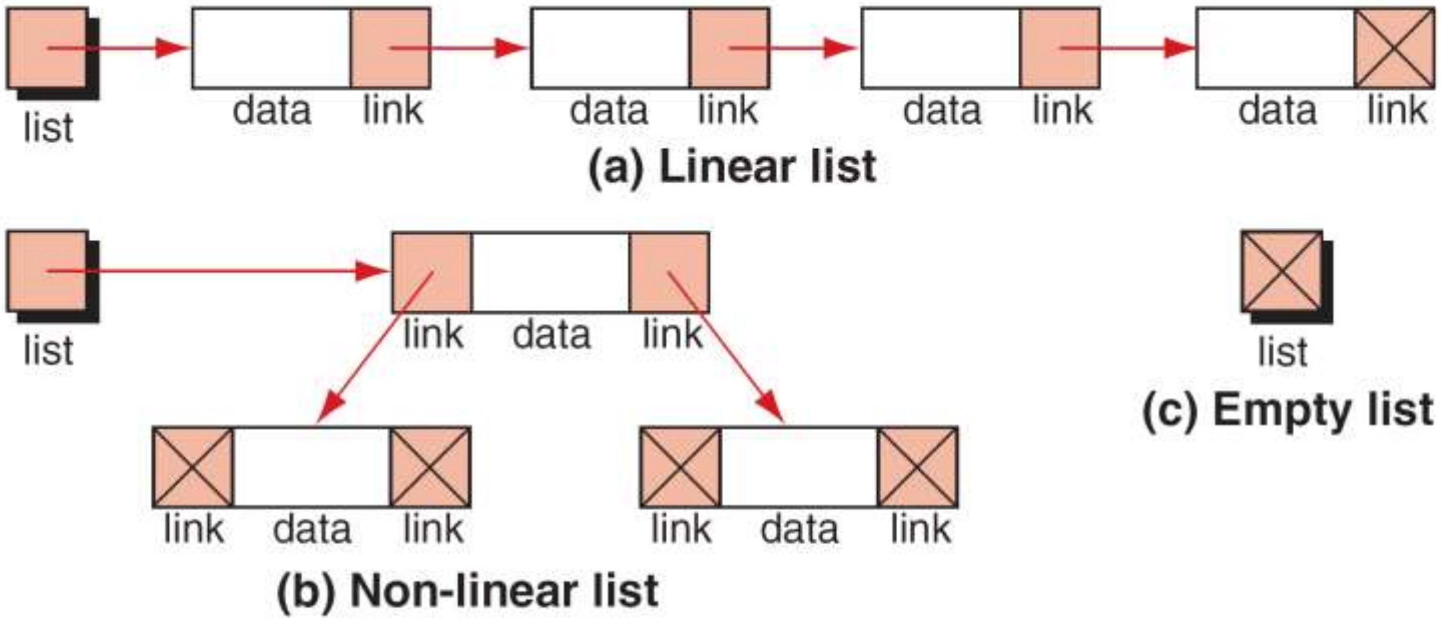


FIGURE 15-2 **Linked Lists**

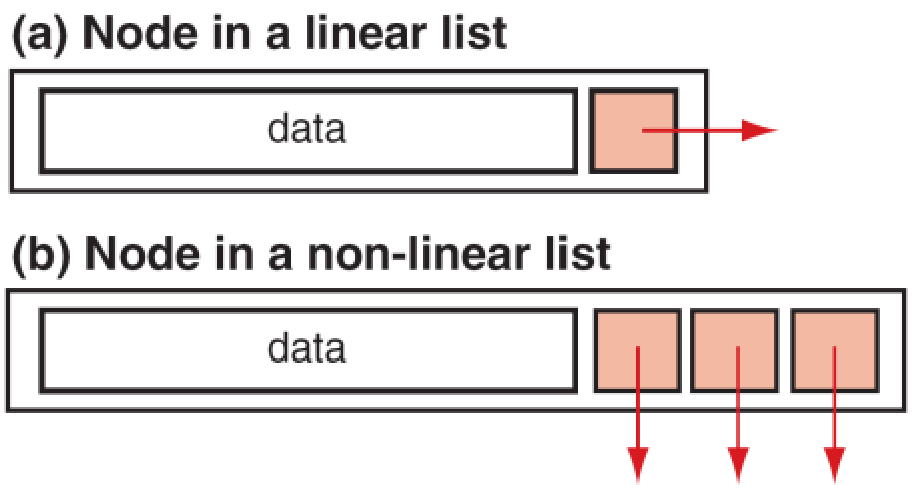


FIGURE 15-3 Nodes

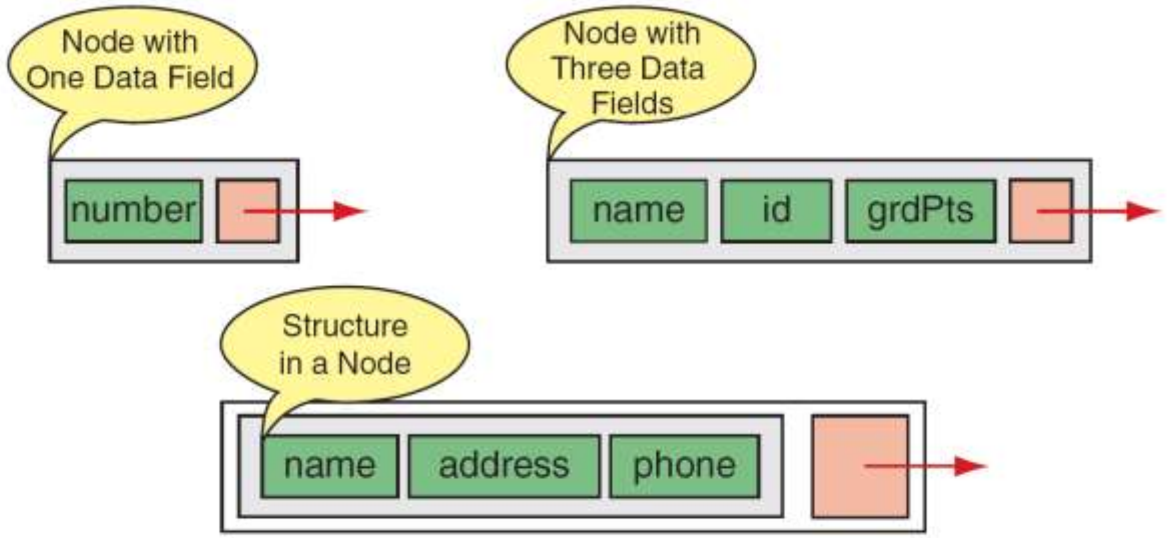


FIGURE 15-4 **Linked List Node Structures**

15-2 General Linear Lists

A general linear list is a list in which operations, such as retrievals, insertions, changes, and deletions, can be done anywhere in the list, that is, at the beginning, in the middle, or at the end of the list..

Topics discussed in this section:

Insert a Node
Delete a Node
Locating Data in Linear Lists
Traversing Linear Lists
Building a Linear List

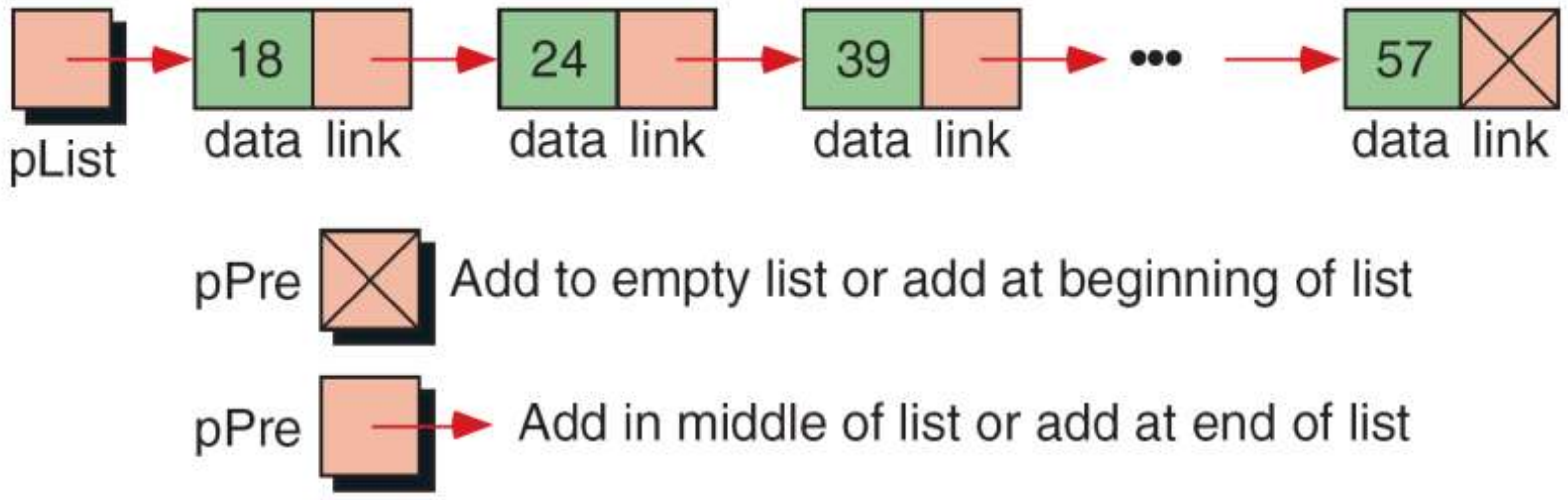

18
24
39
57
data link
data link
data link
data link
pList
pPre
Add to empty list or add at beginning of list
pPre
Add in middle of list or add at end of list

FIGURE 15-5 Pointer Combinations for Insert

```
pNew->link  =  pList ;
pList       =  pNew ;
```

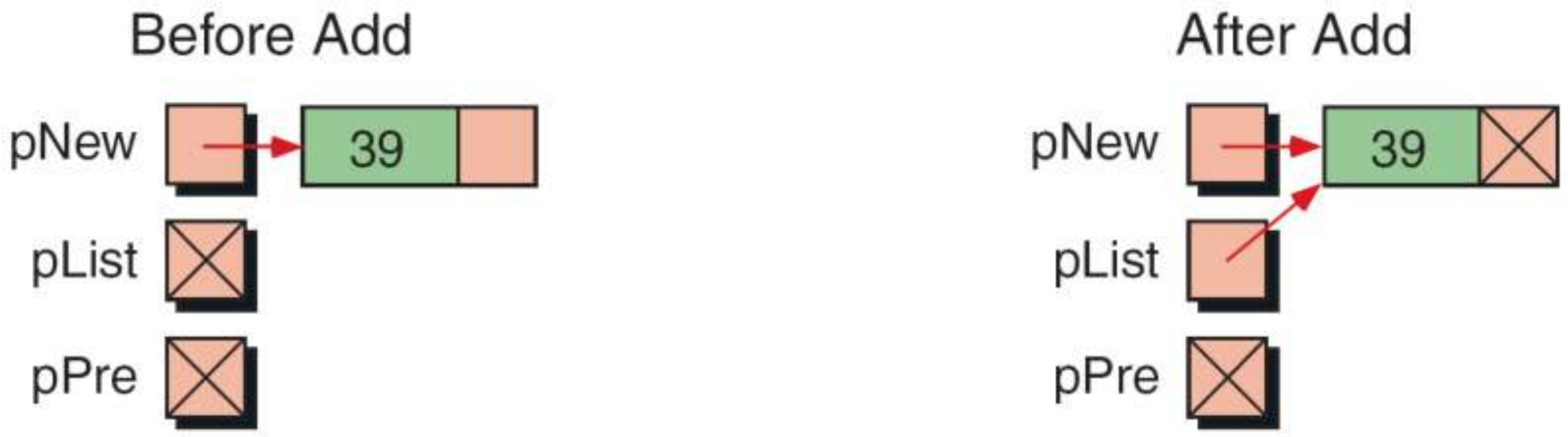


FIGURE 15-6 Insert Node to Empty List

```
pNew->link   = pList ;
pList        = pNew;
```

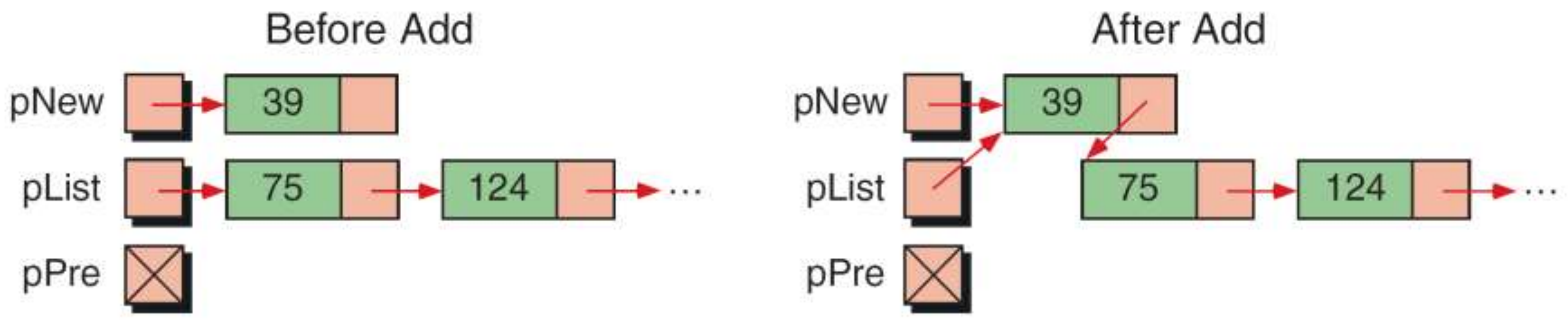


FIGURE 15-7 **Insert Node at Beginning**

```
pNew->link = pPre–>link;
pPre->link  = pNew;
```

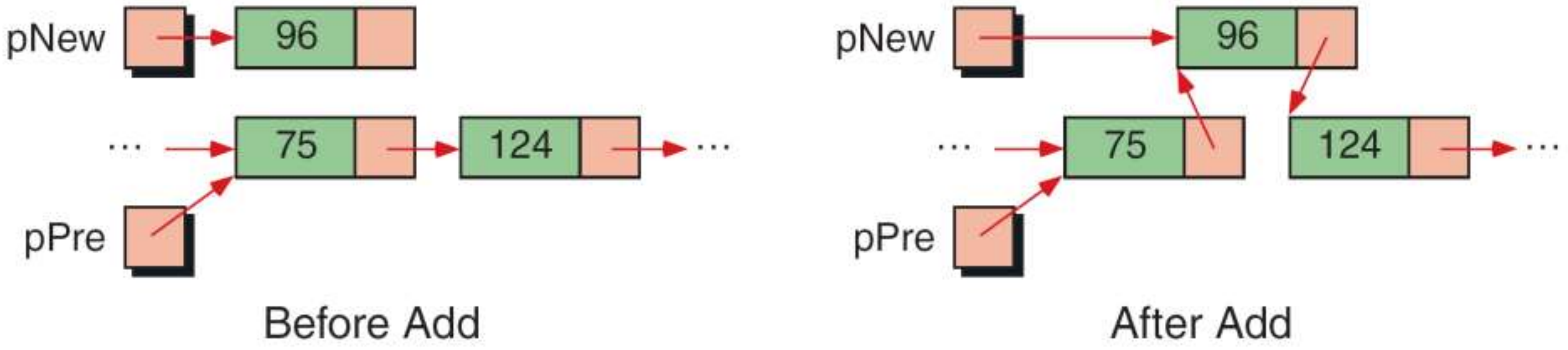
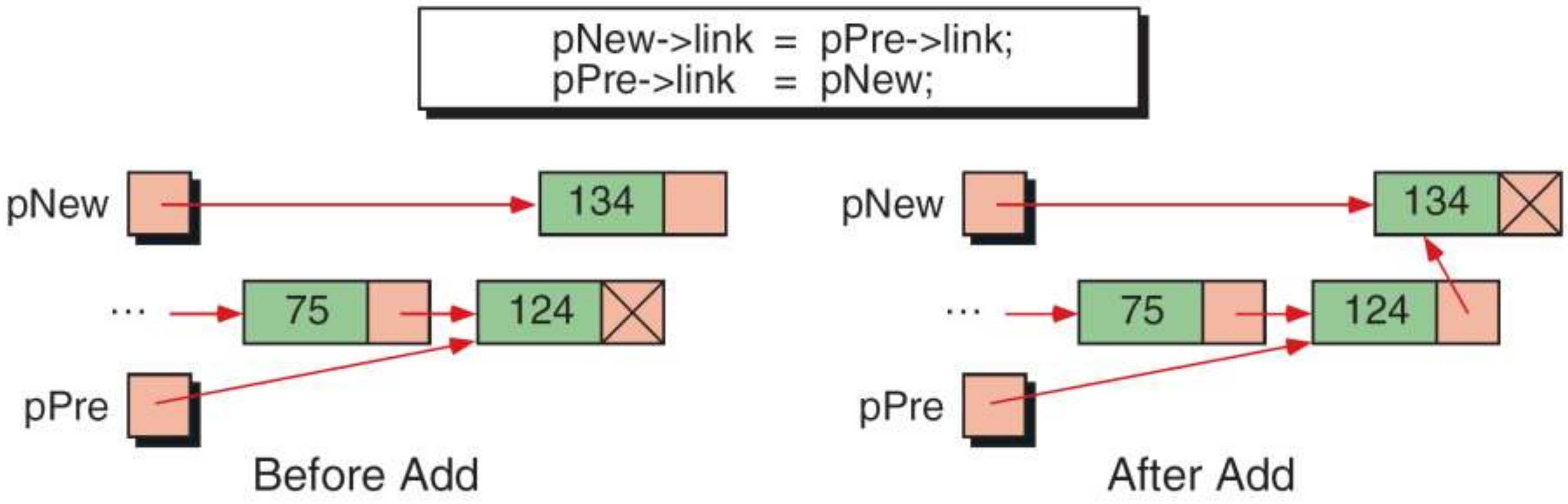


FIGURE 15-8 **Insert Node in Middle**



```
pNew->link  =  pPre->link;
pPre->link  =  pNew;
```

FIGURE 15-9 Insert Node at End

PROGRAM 15-1 Insert a Node

```
/* ==================== insertNode ====================
   This function inserts a single node into a linear list.
      Pre   pList is pointer to the list; may be null
            pPre points to new node's predecessor
            item contains data to be inserted
      Post  returns the head pointer
*/
NODE* insertNode (NODE* pList, NODE* pPre, DATA item)
{
// Local Declarations
   NODE* pNew;

// Statements
   if (!(pNew  = (NODE*)malloc(sizeof(NODE))))
       printf("\aMemory overflow in insert\n"),
              exit (100);

```

PROGRAM 15-1 Insert a Node

```c
   pNew->data = item;
   if (pPre == NULL)
      {
       // Inserting before first node or to empty list
       pNew->link  = pList;
       pList       = pNew;
      } // if pPre
   else
      {
       // Inserting in middle or at end
       pNew->link = pPre->link;
       pPre->link = pNew;
      } // else
   return pList;
}  // insertNode
```

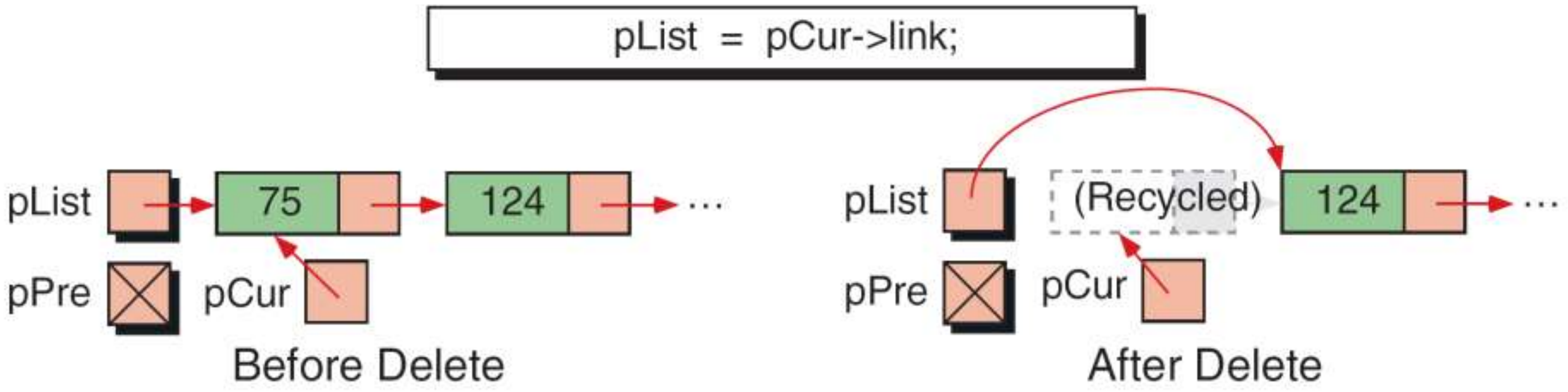


FIGURE 15-10 **Delete First Node**

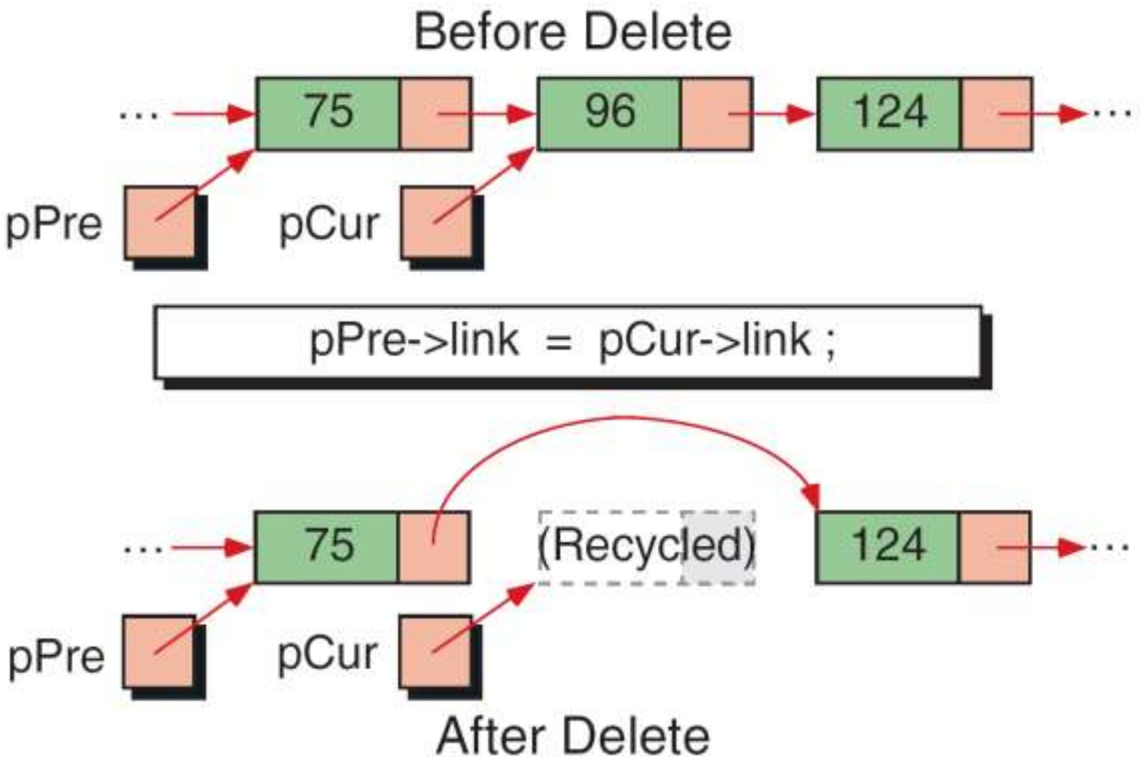


FIGURE 15-11 **Delete—General Case**

PROGRAM 15-2 **Delete a Node**

```
/* ==================== deleteNode ====================
   This function deletes a single node from the link list.
      Pre    pList is a pointer to the head of the list
             pPre points to node before the delete node
             pCur points to the node to be deleted
      Post   deletes and recycles pCur
             returns the head pointer
*/
NODE* deleteNode (NODE* pList, NODE* pPre, NODE* pCur)
{
// Statements
   if (pPre == NULL)
      // Deleting first node
      pList = pCur->link;
   else
      // Deleting other nodes
      pPre->link = pCur->link;
   free  (pCur);
   return pList;
}  // deleteNode
```

Condition	pPre	pCur	Return
target < first node	*NULL*	first node	0
target == first node	*NULL*	first node	1
first < target < last	largest node < target	first node > target	0
target == middle node	node's predecessor	equal node	1
target == last node	last's predecessor	last node	1
target > last node	last node	*NULL*	0

Table 15-1 **Linear List Search Results**

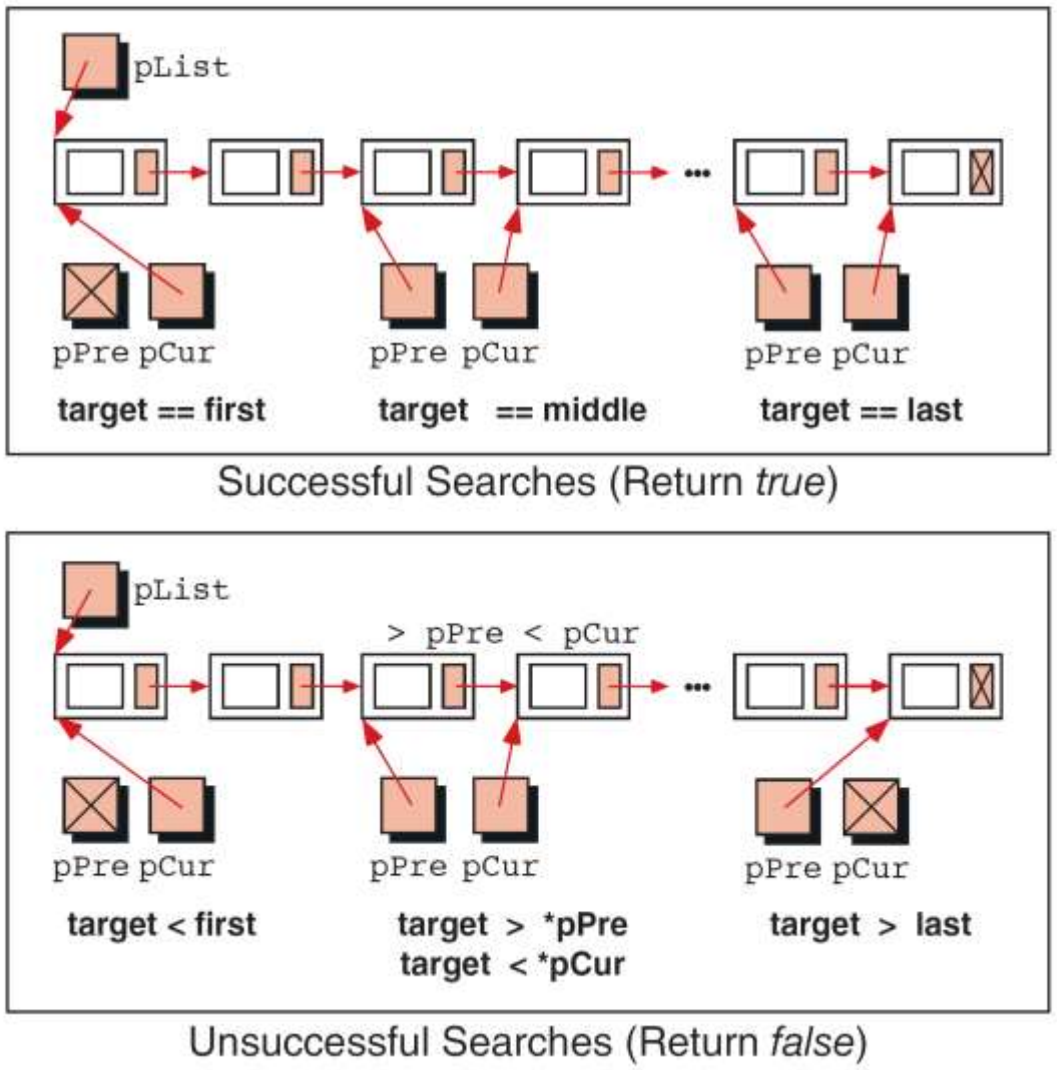


FIGURE 15-12 Search Results

PROGRAM 15-3 Search Linear List

```
/* ==================== searchList ====================
   Given key value, finds the location of a node
      Pre    pList points to a head node
             pPre points to variable to receive pred
             pCur points to variable for current node
             target is key being sought
      Post   pCur points to first node with >= key
             -or- null if target > key of last node
             pPre points to largest node < key
             -or- null if target < key of first node
             function returns true if found
                              false if not found
*/
bool searchList (NODE*  pList, NODE**   pPre,
                 NODE** pCur,  KEY_TYPE target)
{
// Local Declarations
   bool found = false;

```

PROGRAM 15-3 Search Linear List

```c
// Statements
   *pPre = NULL;
   *pCur = pList;

   // start the search from beginning
   while (*pCur != NULL && target > (*pCur)->data.key)
      {
       *pPre = *pCur;
       *pCur = (*pCur)->link;
      } // while

   if (*pCur && target == (*pCur)->data.key)
      found = true;
   return found;
}  // searchList
```

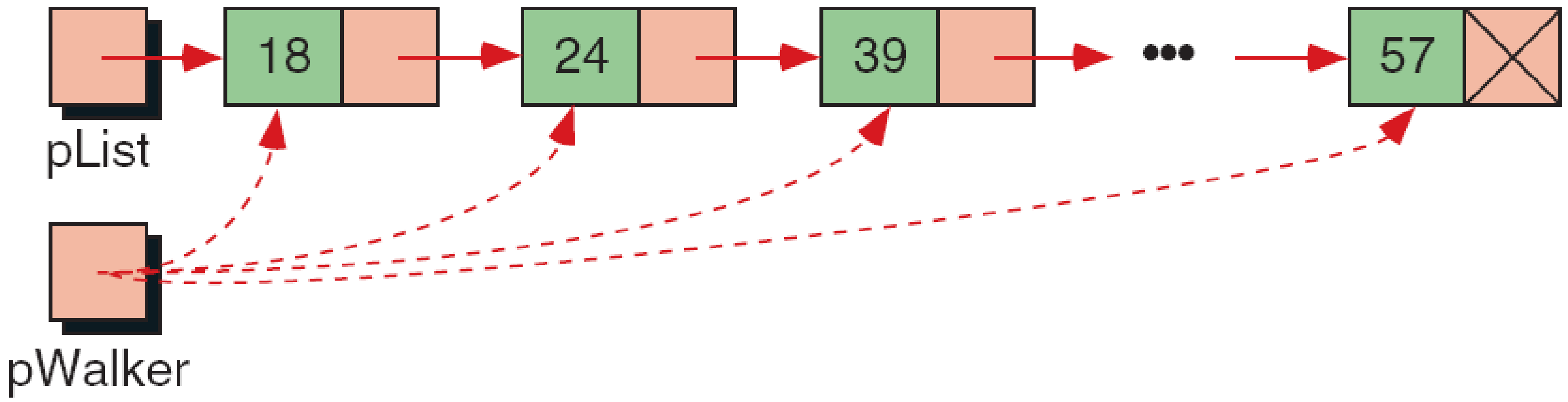


FIGURE 15-13 **Linear List Traversal**

PROGRAM 15-4 Print Linear List

```c
/* Traverse and print a linear list.
      Pre   pList is a valid linear list
      Post  List has been printed
*/
void printList (NODE* pList)
{
// Local Declarations
   NODE* pWalker;

// Statements
   pWalker = pList;
   printf("List contains:\n");

   while (pWalker)
      {
       printf("%3d ", pWalker->data.key);
       pWalker = pWalker->link;
      } // while
   printf("\n");
   return;
} // printList
```

PROGRAM 15-5 Average Linear List

```c
/* This function averages the values in a linear list.
      Pre   pList is a pointer to a linear list
      Post  list average is returned
*/
double averageList (NODE* pList)
{
// Local Declarations
   NODE* pWalker;
   int   total;
   int   count;

// Statements
   total   = count = 0;
   pWalker = pList;
   while (pWalker)
      {
       total += pWalker->data.key;
       count++;
       pWalker = pWalker->link;
      } // while
   return (double)total / count;
} // averageList
```

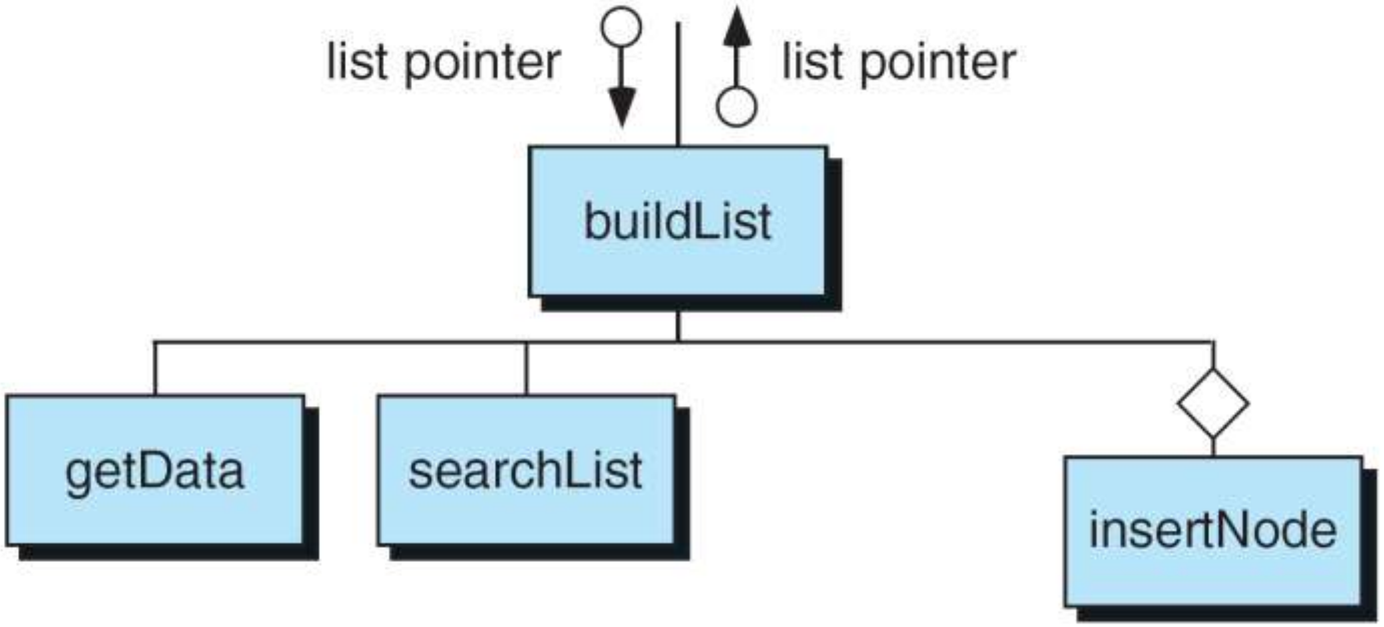


FIGURE 15-14 **Design for Inserting a Node in a List**

PROGRAM 15-6 Build List

```
/* ==================== buildList ====================
   This program builds a key-sequenced linear list.
      Pre   fileID is file that contains data for list
      Post  list built
            returns pointer to head of list
*/
NODE* buildList (char* fileID)
{
// Local Declarations
   DATA  data;
   NODE* pList;
   NODE* pPre;
   NODE* pCur;
   FILE* fpData;

// Statements
   pList = NULL;

```

PROGRAM 15-6 Build List

```c
   fpData = fopen(fileID, "r");
   if (!fpData)
      {
       printf("Error opening file %s\a\n", fileID);
       exit (210);
      } // if open fail

   while (getData (fpData, &data))
      {
       // Determine insert position
       searchList (pList, &pPre, &pCur, data.key);
       pList = insertNode(pList, pPre, data);
      } // while
   return pList;
} // buildList
```

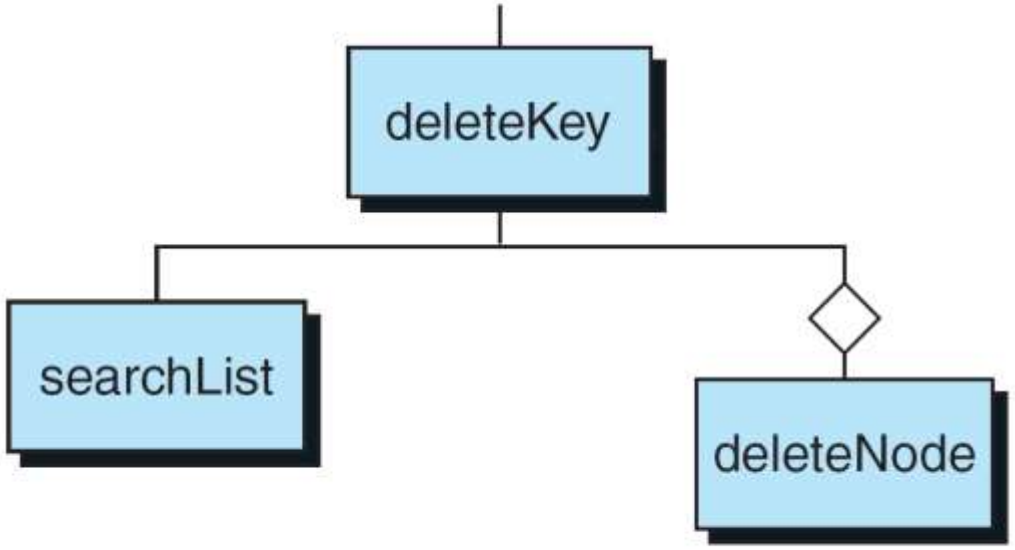


FIGURE 15-15 **Design for Remove Node**

PROGRAM 15-7 **Delete Key**

```
/* ==================== deleteKey ====================
   Delete node from a linear list.
      Pre   list is a pointer to the head of the list
      Post  node has been deleted
            -or- a warning message printed if not found
            returns pointer to first node (pList)
*/
NODE* deleteKey (NODE* pList)
{
// Local Declarations
   int   key;
   NODE* pHead;
   NODE* pCur;
   NODE* pPre;

// Statements
   printf("Enter key of node to be deleted: ");
```

PROGRAM 15-7 Delete Key

```
    scanf ("%d", &key);

    if (!searchList(pList, &pPre, &pCur, key))
        printf("%d is an invalid key\a\n", key);
    else
        pHead = deleteNode (pList, pPre, pCur);

    return pHead;
} // deleteKey
```

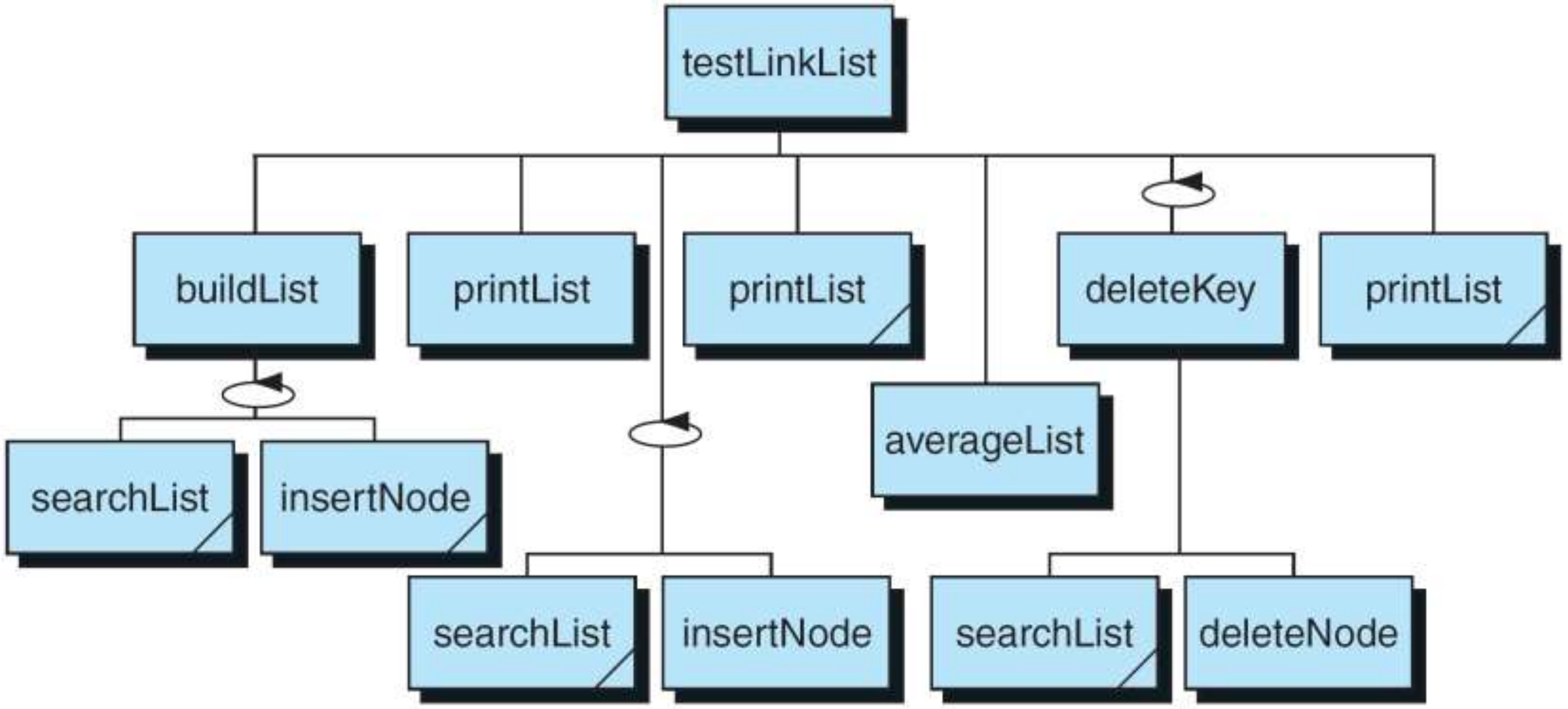


FIGURE 15-16 Link List Test Driver

PROGRAM 15-8 **Test Driver for Link List**

```
/* Test driver for list functions.
      Written by:
      Date:
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Global Declarations
typedef int KEY_TYPE;
typedef struct
   {
    KEY_TYPE key;
   } DATA;
typedef struct nodeTag
   {
    DATA            data;
    struct nodeTag* link;
   } NODE;

```

PROGRAM 15-8 Test Driver for Link List

```
// Function Declarations
NODE* insertNode (NODE*  pList, NODE*  pPre, DATA  item);
NODE* deleteNode (NODE*  List,  NODE*  pPre, NODE* pCur);
bool  searchList (NODE*  pList, NODE**   pPre,
                  NODE** pCur,  KEY_TYPE target);
void  printList  (NODE*  pList);
NODE* buildList  (char*  fileID);
NODE* deleteKey  (NODE*  pList);
bool  getData    (FILE*  fpData, DATA* pData);

double averageList (NODE* pList);

int main (void)
{
// Local Declarations
   NODE*  pList;
   NODE*  pPre;
   NODE*  pCur;
   DATA   data;
   double avrg;
   char   more;

// Statements
   printf("Begin list test driver\n\n");
```

PROGRAM 15-8 Test Driver for Link List

```

    // Build List
    pList = buildList("P15-LIST.DAT");
    if (!pList)
       {
        printf("Error building chron file\a\n");
        exit  (100);
       } // if
    printList (pList);

    printf("\nInsert data tests.\n");
    printf("Enter key:              ");
    scanf ("%d", &data.key);
    do
       {
        if (searchList (pList, &pPre, &pCur, data.key))
           printf("Key already in list. Not inserted\n");
        else
            pList = insertNode(pList, pPre, data);
        printf("Enter key <-1> to stop: ");
        scanf ("%d", &data.key);
       } while (data.key != -1);
```

PROGRAM 15-8 Test Driver for Link List

```
   printList (pList);

   avrg = averageList(pList);
   printf("\nData average: %.1f\n", avrg);

   printf("\nDelete data tests.\n");
   do
      {
       pList = deleteKey (pList);
       printf("Delete another <Y/N>: ");
       scanf (" %c", &more);
      } while (more == 'Y' || more == 'y');

   printList (pList);
   printf("\nTests complete.\n");
   return 0;
} // main
```

PROGRAM 15-8 Test Driver for Link List

```
Results:
Begin list test driver

List contains:
111 222 333 444 555 666 777

Insert data tests.
Enter key:                  50
Enter key <-1> to stop: -1
List contains:
 50 111 222 333 444 555 666 777

Data average: 394.8

Delete data tests.
Enter key of node to be deleted: 50
Delete another <Y/N>: n
List contains:
111 222 333 444 555 666 777

Tests complete.
```