```c
#include <stdio.h>
#include <stdlib.h>

/* self-referential structure */
typedef struct listNode {
    char data; /* each listNode contains a character */
    struct listNode* nextPtr; /* pointer to next node*/
} ListNode_t; /* end structure listNode */



/* prototypes */
void insert(ListNode_t** sPtr, char value);
char delete(ListNode_t** sPtr, char value);
int isEmpty(ListNode_t* sPtr);
void printList(ListNode_t* currentPtr);
void instructions(void);

int main()
{
    ListNode_t* startPtr = NULL; /* initially there are no nodes */
    int choice = 0; /* user's choice */
    char item = 0;  /* char entered by user */

    instructions(); /* display the menu */
    printf("? ");
    scanf("%d", &choice);

    /* loop while user does not choose 3 */
    while (choice != 3) {

        switch (choice) {

        case 1:
            printf("Enter a character: ");
            scanf("\n%c", &item);
            insert(&startPtr, item); /* insert item in list */
            printList(startPtr);
            break;

        case 2:

            /* if list is not empty */
            if (!isEmpty(startPtr)) {
                printf("Enter character to be deleted: ");
                scanf("\n%c", &item);

                /* if character is found, remove it */
                if (delete(&startPtr, item)) { /* remove item */
                    printf("%c deleted.\n", item);
                    printList(startPtr);
                } /* end if */
                else {
```

```
54                      printf("%c not found.\n\n", item);
55                  } /* end else */
56
57              } /* end if */
58              else {
59                  printf("List is empty.\n\n");
60              } /* end else */
61
62              break;
63
64          default:
65              printf("Invalid choice.\n\n");
66              break;
67
68          } /* end switch */
69          instructions(); /* display the menu */
70          printf("? ");
71          scanf("%d", &choice);
72      } /* end while */
73
74      printf("End of run.\n");
75
76      return 0; /* indicates successful termination */
77
78  } /* end main */
79
80  /* display program instructions to user */
81  void instructions(void)
82  {
83      printf("Enter your choice:\n"
84          "   1 to insert an element into the list.\n"
85          "   2 to delete an element from the list.\n"
86          "   3 to end.\n");
87  } /* end function instructions */
88
89  /* Insert a new value into the list in sorted order */
90  void insert(ListNode_t** sPtr, char value)
91  {
92      ListNode_t* newPtr = NULL;      /* pointer to new node */
93      ListNode_t* previousPtr = NULL; /* pointer to previous node in list */
94      ListNode_t* currentPtr = NULL;  /* pointer to current node in list */
95
96      newPtr = malloc(sizeof(ListNode_t)); /* create node on heap */
97
98      if (newPtr != NULL) { /* is space available */
99          newPtr->data = value; /* place value in node */
100         newPtr->nextPtr = NULL; /* node does not link to another node */
101
102         previousPtr = NULL;
103         currentPtr = *sPtr;
104
105         /* loop to find the correct location in the list */
106         while (currentPtr != NULL && value > currentPtr->data) {
```

```
107             previousPtr = currentPtr;          /* walk to ...   */
108             currentPtr = currentPtr->nextPtr;  /* ... next node */
109         } /* end while */
110
111         /* insert new node at beginning of list */
112         if (previousPtr == NULL) {
113             newPtr->nextPtr = *sPtr;
114             *sPtr = newPtr;
115         } /* end if */
116         else { /* insert new node between previousPtr and currentPtr */
117             previousPtr->nextPtr = newPtr;
118             newPtr->nextPtr = currentPtr;
119         } /* end else */
120
121     } /* end if */
122     else {
123         printf("%c not inserted. No memory available.\n", value);
124     } /* end else */
125
126 } /* end function insert */
127
128 /* Delete a list element */
129 char delete(ListNode_t** sPtr, char value)
130 {
131     ListNode_t* previousPtr = NULL; /* pointer to previous node in list */
132     ListNode_t* currentPtr = NULL;  /* pointer to current node in list */
133     ListNode_t* tempPtr = NULL;     /* temporary node pointer */
134
135     /* delete first node */
136     if (value == (*sPtr)->data) {
137         tempPtr = *sPtr; /* hold onto node being removed */
138         *sPtr = (*sPtr)->nextPtr; /* de-thread the node */
139         free(tempPtr); /* free the de-threaded node */
140         return value;
141     } /* end if */
142     else {
143         previousPtr = *sPtr;
144         currentPtr = (*sPtr)->nextPtr;
145
146         /* loop to find the correct location in the list */
147         while (currentPtr != NULL && currentPtr->data != value) {
148             previousPtr = currentPtr;          /* walk to ...   */
149             currentPtr = currentPtr->nextPtr; /* ... next node */
150         } /* end while */
151
152         /* delete node at currentPtr */
153         if (currentPtr != NULL) {
154             tempPtr = currentPtr;
155             previousPtr->nextPtr = currentPtr->nextPtr;
156             free(tempPtr);
157             return value;
158         } /* end if */
159
```

```
160     } /* end else */
161
162     return '\0';
163
164 } /* end function delete */
165
166 /* Return 1 if the list is empty, 0 otherwise */
167 int isEmpty(ListNode_t* sPtr)
168 {
169     return sPtr == NULL;
170
171 } /* end function isEmpty */
172
173 /* Print the list */
174 void printList(ListNode_t* currentPtr)
175 {
176
177     /* if list is empty */
178     if (currentPtr == NULL) {
179         printf("List is empty.\n\n");
180     } /* end if */
181     else {
182         printf("The list is:\n");
183
184         /* while not the end of the list */
185         while (currentPtr != NULL) {
186             printf("%c --> ", currentPtr->data);
187             currentPtr = currentPtr->nextPtr;
188         } /* end while */
189
190         printf("NULL\n\n");
191     } /* end else */
192
193 } /* end function printList */
194
195
```