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

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

typedef struct listNode ListNode;  // synonym for struct listNode
typedef ListNode * ListNodePtr;  // synonym for ListNode*

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

int main(void) {
   ListNodePtr startPtr = NULL;  // initially there are no nodes
   printf("\nline 22: @startPtr -> %d\n", &startPtr);
   printf("\nstartPtr -> %d\n", startPtr);
   char item;  // char entered by user
   printf("\n@item -> %d\n", &item);
   instructions();  // display the menu
   printf("%s", "? ");
   unsigned int choice;  // user's choice
   printf("\nline 29: @choice -> %d\n", &choice);
   scanf("%u", &choice);

   // loop while user does not choose 3
   while (choice != 3) {
      switch (choice) {
         case 1:
            printf("%s", "Enter a character: ");
            scanf("\n%c", &item);
            printf("\nline 38: @startPtr -> %d\n", &startPtr);
            printf("\nstartPtr -> %d\n", startPtr);
            insert(&startPtr, item);  // insert item in list
            printf("\nline 41: @startPtr -> %d\n", &startPtr);
            printf("\nstartPtr -> %d\n", startPtr);
            printList(startPtr);
            break;
         case 2:  // delete an element
            // if list is not empty
```

```
47:             if (!isEmpty(startPtr)) {
48:                printf("\nline 48: @startPtr -> %d\n", &startPtr);
49:                printf("\nstartPtr -> %d\n", startPtr);
50:                printf("%s", "Enter character to be deleted: ");
51:                scanf("\n%c", &item);
52:                // if character is found, remove it
53:                if (delete(&startPtr, item)) {  // remove item
54:                   printf("\nline 54: @startPtr -> %d\n", &startPtr);
55:                   printf("\nstartPtr -> %d\n", startPtr);
56:                   printf("%c deleted.\n", item);
57:                   printf("\nline 57: @startPtr -> %d\n", &startPtr);
58:                   printf("\nstartPtr -> %d\n", startPtr);
59:                   printList(startPtr);
60:                }
61:                else {
62:                   printf("%c not found.\n\n", item);
63:                }
64:             }
65:             else {
66:                puts("List is empty.\n");
67:             }
68:             break;
69:          default:
70:             puts("Invalid choice.\n");
71:             instructions();
72:             break;
73:       }  // end switch
74:       printf("%s", "? ");
75:       scanf("%u", &choice);
76:    }
77:    puts("End of run.");
78: }
79:
80: // display program instructions to user
81: void instructions(void) {
82:    puts("Enter your choice:\n"
83:       "   1 to insert an element into the list.\n"
84:       "   2 to delete an element from the list.\n"
85:       "   3 to end.");
86: }
87:
88: // insert a new value into the list in sorted order
89: void insert(ListNodePtr *sPtr, char value) {
90:    printf("\nline 90: @sPtr -> %d\n", &sPtr);
91:    printf("\nsPtr -> %d\n", sPtr);
92:    ListNodePtr newPtr = malloc(sizeof(ListNode));  // create node
```

```
93:    printf("\nline 93: @newPtr -> %d\n", &newPtr);
94:    printf("\nnewPtr -> %d\n", newPtr);
95:    if (newPtr != NULL) {  // is space available
96:       newPtr->data = value;  // place value in node
97:       newPtr->nextPtr = NULL;  // node does not link to another node
98:       printf("\nline 98: @newPtr -> %d\n", &newPtr);
99:       printf("\nnewPtr -> %d\n", newPtr);
100:      ListNodePtr previousPtr = NULL;
101:      printf("\nline 101: @previousPtr -> %d\n", &previousPtr);
102:      printf("\npreviousPtr -> %d\n", previousPtr);
103:      ListNodePtr currentPtr = *sPtr;
104:      printf("\nline 104: @currentPtr -> %d\n", &currentPtr);
105:      printf("\ncurrentPtr -> %d\n", currentPtr);
106:
107:      // loop to find the correct location in the list
108:      while (currentPtr != NULL && value > currentPtr->data) {
109:         previousPtr = currentPtr;  // walk to ...
110:         printf("\nline 110: @previousPtr -> %d\n", &previousPtr);
111:         printf("\npreviousPtr -> %d\n", previousPtr);
112:         currentPtr = currentPtr->nextPtr;  // ... next node
113:         printf("\nline 113: @currentPtr -> %d\n", &currentPtr);
114:         printf("\ncurrentPtr -> %d\n", currentPtr);
115:      }
116:
117:      // insert new node at beginning of list
118:      if (previousPtr == NULL) {
119:         printf("\nline 119: @sPtr -> %d\n", &sPtr);
120:         printf("\nsPtr -> %d\n", sPtr);
121:         newPtr->nextPtr = *sPtr;
122:         printf("\nline 122: @newPtr -> %d\n", &newPtr);
123:         printf("\nnewPtr -> %d\n", newPtr);
124:         *sPtr = newPtr;
125:         printf("\nline 125: @sPtr -> %d\n", &sPtr);
126:         printf("\nsPtr -> %d\n", sPtr);
127:      }
128:      else {  // insert new node between previousPtr and currentPtr
129:         previousPtr->nextPtr = newPtr;
130:         printf("\nline 130: @previousPtr -> %d\n", &previousPtr);
131:         printf("\npreviousPtr -> %d\n", previousPtr);
132:         newPtr->nextPtr = currentPtr;
133:         printf("\nline 133: @newPtr -> %d\n", &newPtr);
134:         printf("\nnewPtr -> %d\n", newPtr);
135:      }
136:   }
137:   else {
138:      printf("%c not inserted. No memory available.\n", value);
```

```
139:    }
140: }
141:
142: // delete a list element
143: char delete(ListNodePtr *sPtr, char value) {
144:    printf("\nline 144: @sPtr -> %d\n", &sPtr);
145:    printf("\nsPtr -> %d\n", sPtr);
146:    // delete first node if a match is found
147:    if (value == (*sPtr)->data) {
148:       printf("\nline 148: @sPtr -> %d\n", &sPtr);
149:       printf("\nsPtr -> %d\n", sPtr);
150:       ListNodePtr tempPtr = *sPtr;  // hold onto node being removed
151:       printf("\nline 151: @tempPtr -> %d\n", &tempPtr);
152:       printf("\ntempPtr -> %d\n", tempPtr);
153:       *sPtr = (*sPtr)->nextPtr;  // de-thread the node
154:       printf("\nline 154: @sPtr -> %d\n", &sPtr);
155:       printf("\nsPtr -> %d\n", sPtr);
156:       free(tempPtr);  // free the de-threaded node
157:       return value;
158:    }
159:    else {
160:       ListNodePtr previousPtr = *sPtr;
161:       printf("\nline 161: @previousPtr -> %d\n", &previousPtr);
162:       printf("\npreviousPtr -> %d\n", previousPtr);
163:       ListNodePtr currentPtr = (*sPtr)->nextPtr;
164:       printf("\nline 164: @currentPtr -> %d\n", &currentPtr);
165:       printf("\ncurrentPtr -> %d\n", currentPtr);
166:
167:       // loop to find the correct location in the list
168:       while (currentPtr != NULL && currentPtr->data != value) {
169:          previousPtr = currentPtr;  // walk to ...
170:          printf("\nline 170: @previousPtr -> %d\n", &previousPtr);
171:          printf("\npreviousPtr -> %d\n", previousPtr);
172:          currentPtr = currentPtr->nextPtr;  // ... next node
173:          printf("\nline 173: @currentPtr -> %d\n", &currentPtr);
174:          printf("\ncurrentPtr -> %d\n", currentPtr);
175:       }
176:
177:       // delete node at currentPtr
178:       if (currentPtr != NULL) {
179:          ListNodePtr tempPtr = currentPtr;
180:          printf("\nline 180: @tempPtr -> %d\n", &tempPtr);
181:          printf("\ntempPtr -> %d\n", tempPtr);
182:          previousPtr->nextPtr = currentPtr->nextPtr;
183:          printf("\nline 183: @previousPtr -> %d\n", &previousPtr);
184:          printf("\npreviousPtr -> %d\n", previousPtr);
```

```
185:         free(tempPtr);
186:         return value;
187:      }
188:   }
189:
190:   return '\0';
191: }
192:
193: // return 1 if the list is empty, 0 otherwise
194: int isEmpty(ListNodePtr sPtr) {
195:   printf("\nline 195: @sPtr -> %d\n", &sPtr);
196:   printf("\nsPtr -> %d\n", sPtr);
197:   return sPtr == NULL;
198: }
199:
200: // print the list
201: void printList(ListNodePtr currentPtr) {
202:   printf("\nline 202: @currentPtr -> %d\n", &currentPtr);
203:   printf("\ncurrentPtr -> %d\n", currentPtr);
204:   // if list is empty
205:   if (isEmpty(currentPtr)) {
206:      puts("List is empty.\n");
207:   }
208:   else {
209:      puts("The list is:");
210:
211:      // while not the end of the list
212:      while (currentPtr != NULL) {
213:         printf("\nline 213: @currentPtr -> %d\n", &currentPtr);
214:         printf("\ncurrentPtr -> %d\n", currentPtr);
215:         printf("%c --> ", currentPtr->data);
216:         currentPtr = currentPtr->nextPtr;
217:      }
218:      puts("NULL\n");
219:   }
220: }
```