

```
1. /**
2.  * list-0.c
3.  *
4.  * David J. Malan
5.  * malan@harvard.edu
6.  *
7.  * Demonstrates a linked list for numbers.
8.  */
9.
10. #include <cs50.h>
11. #include <stdio.h>
12. #include <stdlib.h>
13. #include <unistd.h>
14.
15. #include "list-0.h"
16.
17. // linked list
18. node* first = NULL;
19.
20. // prototypes
21. void delete(void);
22. void insert(void);
23. void search(void);
24. void traverse(void);
25.
26. int main(void)
27. {
28.     int c;
29.     do
30.     {
31.         // print instructions
32.         printf("\nMENU\n\n"
33.             "1 - delete\n"
34.             "2 - insert\n"
35.             "3 - search \n"
36.             "4 - traverse\n"
37.             "0 - quit\n\n");
38.
39.         // get command
40.         printf("Command: ");
41.         c = GetInt();
42.
43.         // try to execute command
44.         switch (c)
45.         {
46.             case 1: delete(); break;
47.             case 2: insert(); break;
48.             case 3: search(); break;
```

```
49.         case 4: traverse(); break;
50.     }
51. }
52. while (c != 0);
53.
54. // free list before quitting
55. node* ptr = first;
56. while (ptr != NULL)
57. {
58.     node* predptr = ptr;
59.     ptr = ptr->next;
60.     free(predptr);
61. }
62. }
63.
64. /**
65.  * Tries to delete a number.
66.  */
67. void delete(void)
68. {
69.     // prompt user for number
70.     printf("Number to delete: ");
71.     int n = GetInt();
72.
73.     // get list's first node
74.     node* ptr = first;
75.
76.     // try to delete number from list
77.     node* predptr = NULL;
78.     while (ptr != NULL)
79.     {
80.         // check for number
81.         if (ptr->n == n)
82.         {
83.             // delete from head
84.             if (ptr == first)
85.             {
86.                 first = ptr->next;
87.                 free(ptr);
88.             }
89.
90.             // delete from middle or tail
91.             else
92.             {
93.                 predptr->next = ptr->next;
94.                 free(ptr);
95.             }
96.
```

```
97.         // all done
98.         break;
99.     }
100.    else
101.    {
102.        predptr = ptr;
103.        ptr = ptr->next;
104.    }
105. }
106.
107. // traverse list
108. traverse();
109. }
110.
111. /**
112.  * Tries to insert a number into list.
113.  */
114. void insert(void)
115. {
116.     // try to instantiate node for number
117.     node* newptr = malloc(sizeof(node));
118.     if (newptr == NULL)
119.     {
120.         return;
121.     }
122.
123.     // initialize node
124.     printf("Number to insert: ");
125.     newptr->n = GetInt();
126.     newptr->next = NULL;
127.
128.     // check for empty list
129.     if (first == NULL)
130.     {
131.         first = newptr;
132.     }
133.
134.     // else check if number belongs at list's head
135.     else if (newptr->n < first->n)
136.     {
137.         newptr->next = first;
138.         first = newptr;
139.     }
140.
141.     // else try to insert number in middle or tail
142.     else
143.     {
144.         node* predptr = first;
```

```
145.     while (true)
146.     {
147.         // avoid duplicates
148.         if (predptr->n == newptr->n)
149.         {
150.             free(newptr);
151.             break;
152.         }
153.
154.         // check for insertion at tail
155.         else if (predptr->next == NULL)
156.         {
157.             predptr->next = newptr;
158.             break;
159.         }
160.
161.         // check for insertion in middle
162.         else if (predptr->next->n > newptr->n)
163.         {
164.             newptr->next = predptr->next;
165.             predptr->next = newptr;
166.             break;
167.         }
168.
169.         // update pointer
170.         predptr = predptr->next;
171.     }
172. }
173.
174. // traverse list
175. traverse();
176. }
177.
178. /**
179.  * Searches for a number in list.
180.  */
181. void search(void)
182. {
183.     // prompt user for number
184.     printf("Number to search for: ");
185.     int n = GetInt();
186.
187.     // get list's first node
188.     node* ptr = first;
189.
190.     // search for number
191.     while (ptr != NULL)
192.     {
```

```
193.     if (ptr->n == n)
194.     {
195.         printf("\nFound %i!\n", n);
196.         sleep(1);
197.         break;
198.     }
199.     ptr = ptr->next;
200. }
201. }
202.
203. /**
204.  * Traverses list, printing its numbers.
205.  */
206. void traverse(void)
207. {
208.     // traverse list
209.     printf("\nLIST IS NOW: ");
210.     node* ptr = first;
211.     while (ptr != NULL)
212.     {
213.         printf("%i ", ptr->n);
214.         ptr = ptr->next;
215.     }
216.
217.     // flush standard output since we haven't outputted any newlines yet
218.     fflush(stdout);
219.
220.     // pause before continuing
221.     sleep(1);
222.     printf("\n\n");
223. }
```

```
1. /**
2.  * list-0.h
3.  *
4.  * David J. Malan
5.  * malan@harvard.edu
6.  *
7.  * Defines a node for a linked list of integers.
8.  */
9.
10. typedef struct node
11. {
12.     int n;
13.     struct node* next;
14. }
15. node;
```

```
1. /**
2.  * list-1.c
3.  *
4.  * David J. Malan
5.  * malan@harvard.edu
6.  *
7.  * Demonstrates a linked list for students.
8.  */
9.
10. #include <cs50.h>
11. #include <stdio.h>
12. #include <stdlib.h>
13. #include <unistd.h>
14.
15. #include "list-1.h"
16.
17. // linked list
18. node* first = NULL;
19.
20. // prototypes
21. void delete(void);
22. void insert(void);
23. void search(void);
24. void traverse(void);
25.
26. int main(void)
27. {
28.     int c;
29.     do
30.     {
31.         // print instructions
32.         printf("\nMENU\n\n"
33.             "1 - delete\n"
34.             "2 - insert\n"
35.             "3 - search\n"
36.             "4 - traverse\n"
37.             "0 - quit\n\n");
38.
39.         // get command
40.         printf("Command: ");
41.         c = GetInt();
42.
43.         // try to execute command
44.         switch (c)
45.         {
46.             case 1: delete(); break;
47.             case 2: insert(); break;
48.             case 3: search(); break;
```

```
49.         case 4: traverse(); break;
50.     }
51. }
52. while (c != 0);
53.
54. // free list before quitting
55. node* ptr = first;
56. while (ptr != NULL)
57. {
58.     node* predptr = ptr;
59.     ptr = ptr->next;
60.     if (predptr->student != NULL)
61.     {
62.         if (predptr->student->name != NULL)
63.         {
64.             free(predptr->student->name);
65.         }
66.         if (predptr->student->house != NULL)
67.         {
68.             free(predptr->student->house);
69.         }
70.         free(predptr->student);
71.     }
72.     free(predptr);
73. }
74. }
75.
76. /**
77.  * Tries to delete a student.
78.  */
79. void delete(void)
80. {
81.     // prompt user for ID
82.     printf("ID to delete: ");
83.     int n = GetInt();
84.
85.     // get list's first node
86.     node* ptr = first;
87.
88.     // try to delete student from list
89.     node* predptr = NULL;
90.     while (ptr != NULL)
91.     {
92.         // check for ID
93.         if (ptr->student->id == n)
94.         {
95.             // delete from head
96.             if (ptr == first)
```



```
97.     {
98.         first = ptr->next;
99.         free(ptr->student->name);
100.        free(ptr->student->house);
101.        free(ptr->student);
102.        free(ptr);
103.    }
104.
105.    // delete from middle or tail
106.    else
107.    {
108.        predptr->next = ptr->next;
109.        if (ptr->student->name != NULL)
110.        {
111.            free(ptr->student->name);
112.        }
113.        if (ptr->student->house != NULL)
114.        {
115.            free(ptr->student->house);
116.        }
117.        free(ptr->student);
118.        free(ptr);
119.    }
120.
121.    // all done
122.    break;
123. }
124. else
125. {
126.     predptr = ptr;
127.     ptr = ptr->next;
128. }
129. }
130.
131. // traverse list
132. traverse();
133. }
134.
135. /**
136.  * Tries to insert a student into list.
137.  */
138. void insert(void)
139. {
140.     // try to instantiate node for student
141.     node* newptr = malloc(sizeof(node));
142.     if (newptr == NULL)
143.     {
144.         return;
```

```
145.     }
146.
147.     // initialize node
148.     newptr->next = NULL;
149.
150.     // try to instantiate student
151.     newptr->student = malloc(sizeof(student));
152.     if (newptr->student == NULL)
153.     {
154.         free(newptr);
155.         return;
156.     }
157.
158.     // try to initialize student
159.     printf("Student's ID: ");
160.     newptr->student->id = GetInt();
161.     printf("Student's name: ");
162.     newptr->student->name = GetString();
163.     printf("Student's house: ");
164.     newptr->student->house = GetString();
165.     if (newptr->student->name == NULL || newptr->student->house == NULL)
166.     {
167.         if (newptr->student->name != NULL)
168.         {
169.             free(newptr->student->name);
170.         }
171.         if (newptr->student->house != NULL)
172.         {
173.             free(newptr->student->house);
174.         }
175.         free(newptr->student);
176.         free(newptr);
177.         return;
178.     }
179.
180.     // check for empty list
181.     if (first == NULL)
182.     {
183.         first = newptr;
184.     }
185.
186.     // else check if student belongs at list's head
187.     else if (newptr->student->id < first->student->id)
188.     {
189.         newptr->next = first;
190.         first = newptr;
191.     }
192.
```

```
193. // else try to insert student in middle or tail
194. else
195. {
196.     node* predptr = first;
197.     while (true)
198.     {
199.         // avoid duplicates
200.         if (predptr->student->id == newptr->student->id)
201.         {
202.             free(newptr->student->name);
203.             free(newptr->student->house);
204.             free(newptr->student);
205.             free(newptr);
206.             break;
207.         }
208.
209.         // check for insertion at tail
210.         else if (predptr->next == NULL)
211.         {
212.             predptr->next = newptr;
213.             break;
214.         }
215.
216.         // check for insertion in middle
217.         else if (predptr->next->student->id > newptr->student->id)
218.         {
219.             newptr->next = predptr->next;
220.             predptr->next = newptr;
221.             break;
222.         }
223.
224.         // update pointer
225.         predptr = predptr->next;
226.     }
227. }
228.
229. // traverse list
230. traverse();
231. }
232.
233.
234. /**
235.  * Searches for student in list via student's ID.
236.  */
237. void search(void)
238. {
239.     // prompt user for ID
240.     printf("ID to search for: ");
```

```
241.     int id = GetInt();
242.
243.     // get list's first node
244.     node* ptr = first;
245.
246.     // search for student
247.     while (ptr != NULL)
248.     {
249.         if (ptr->student->id == id)
250.         {
251.             printf("\nFound %s of %s (%i)!\n",
252.                 ptr->student->name, ptr->student->house, id);
253.             sleep(1);
254.             break;
255.         }
256.         ptr = ptr->next;
257.     }
258. }
259.
260. /**
261.  * Traverses list, printing its numbers.
262.  */
263. void traverse(void)
264. {
265.     // traverse list
266.     printf("\nLIST IS NOW: ");
267.     node* ptr = first;
268.     while (ptr != NULL)
269.     {
270.         printf("%s of %s (%i) ",
271.             ptr->student->name, ptr->student->house, ptr->student->id);
272.         ptr = ptr->next;
273.     }
274.
275.     // flush standard output since we haven't outputted any newlines yet
276.     fflush(stdout);
277.
278.     // pause before continuing
279.     sleep(1);
280.     printf("\n\n");
281. }
```

```
1. /**
2.  * list-1.h
3.  *
4.  * David J. Malan
5.  * malan@harvard.edu
6.  *
7.  * Defines structures for students and linked lists thereof.
8.  */
9.
10. typedef struct
11. {
12.     int id;
13.     char* name;
14.     char* house;
15. }
16. student;
17.
18. typedef struct node
19. {
20.     student* student;
21.     struct node* next;
22. }
23. node;
```