```
 1: /****************************************************************************
 2:  * list1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Demonstrates a linked list for numbers.
 8:  ****************************************************************************/
 9:
10:
11: #include <cs50.h>
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <unistd.h>
15:
16: #include "list1.h"
17:
18:
19: // linked list
20: node *first = NULL;
21:
22:
23: // prototypes
24: void delete();
25: void find();
26: void insert();
27: void traverse();
28:
29:
30: int
31: main(int argc, char *argv[])
32: {
33:     int c;
34:     do
35:     {
36:         // print instructions
37:         printf("\nMENU\n\n"
38:                "1 - delete\n"
39:                "2 - find\n"
40:                "3 - insert\n"
41:                "4 - traverse\n"
42:                "0 - quit\n\n");
43:
44:         // get command
45:         printf("Command: ");
46:         c = GetInt();
47:
48:         // try to execute command
49:         switch (c)
50:         {
51:             case 1: delete(); break;
52:             case 2: find(); break;
53:             case 3: insert(); break;
54:             case 4: traverse(); break;
55:         }
56:     }
57:     while (c != 0);
58:
59:     // free list before quitting
60:     node *ptr = first;
61:     while (ptr != NULL)
62:     {
63:         node *predptr = ptr;
64:         ptr = ptr->next;
```



```
65:         free(predptr);
66:     }
67:     return 0;
68: }
69:
70:
71: /*
72:  * void
73:  * delete()
74:  *
75:  * Tries to delete a number.
76:  */
77: void
78: delete()
79: {
80:     // prompt user for number
81:     printf("Number to delete: ");
82:     int n = GetInt();
83:
84:     // get list's first node
85:     node *ptr = first;
86:
87:     // try to delete number from list
88:     node *predptr = NULL;
89:     while (ptr != NULL)
90:     {
91:         // check for number
92:         if (ptr->n == n)
93:         {
94:             // delete from head
95:             if (ptr == first)
96:             {
97:                 first = ptr->next;
98:                 free(ptr);
99:             }
100:
101:            // delete from middle or tail
102:            else
103:            {
104:                predptr->next = ptr->next;
105:                free(ptr);
106:            }
107:
108:            // all done
109:            break;
110:        }
111:        else
112:        {
113:            predptr = ptr;
114:            ptr = ptr->next;
115:        }
116:    }
117:
118:    // traverse list
119:    traverse();
120: }
121:
122:
123: /*
124:  * void
125:  * insert()
126:  *
127:  * Tries to insert a number into list.
128:  */
```

```
129: void
130: insert()
131: {
132:     // try to instantiate node for number
133:     node *newptr = malloc(sizeof(node));
134:     if (newptr == NULL)
135:         return;
136:
137:     // initialize node
138:     printf("Number to insert: ");
139:     newptr->n = GetInt();
140:     newptr->next = NULL;
141:
142:     // check for empty list
143:     if (first == NULL)
144:         first = newptr;
145:
146:     // else check if number belongs at list's head
147:     else if (newptr->n < first->n)
148:     {
149:         newptr->next = first;
150:         first = newptr;
151:     }
152:
153:     // else try to insert number in middle or tail
154:     else
155:     {
156:         node *predptr = first;
157:         while (TRUE)
158:         {
159:             // avoid duplicates
160:             if (predptr->n == newptr->n)
161:             {
162:                 free(newptr);
163:                 break;
164:             }
165:
166:             // check for insertion at tail
167:             else if (predptr->next == NULL)
168:             {
169:                 predptr->next = newptr;
170:                 break;
171:             }
172:
173:             // check for insertion in middle
174:             else if (predptr->next->n > newptr->n)
175:             {
176:                 newptr->next = predptr->next;
177:                 predptr->next = newptr;
178:                 break;
179:             }
180:
181:             // update pointer
182:             predptr = predptr->next;
183:         }
184:     }
185:
186:     // traverse list
187:     traverse();
188: }
189:
190:
191: /*
192:  * void
```



```
193:  * find()
194:  *
195:  * Tries to find a number in list.
196:  */
197: void
198: find()
199: {
200:     // prompt user for number
201:     printf("Number to find: ");
202:     int n = GetInt();
203:
204:     // get list's first node
205:     node *ptr = first;
206:
207:     // try to find number
208:     while (ptr != NULL)
209:     {
210:         if (ptr->n == n)
211:         {
212:             printf("\nFound %d!\n", n);
213:             sleep(1);
214:             break;
215:         }
216:         ptr = ptr->next;
217:     }
218: }
219:
220:
221: /*
222:  * void
223:  * traverse()
224:  *
225:  * Traverses list, printing its numbers.
226:  */
227: void
228: traverse()
229: {
230:     // traverse list
231:     printf("\nLIST IS NOW: ");
232:     node *ptr = first;
233:     while (ptr != NULL)
234:     {
235:         printf("%d ", ptr->n);
236:         ptr = ptr->next;
237:     }
238:
239:     // flush standard output since we haven't outputted any newlines yet
240:     fflush(stdout);
241:
242:     // pause before continuing
243:     sleep(1);
244:     printf("\n\n");
245: }
246:
```

```
 1: /**************************************************************************
 2:  * list1.h
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Defines a node for a linked list of integers.
 8:  **************************************************************************/
 9:
10:
11: typedef struct _node
12: {
13:     int n;
14:     struct _node *next;
15: }
16: node;
17:
```



```
 1: /**************************************************************************
 2:  * list2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Demonstrates a linked list for students.
 8:  **************************************************************************/
 9:
10:
11: #include <cs50.h>
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <unistd.h>
15:
16: #include "list2.h"
17:
18:
19: // linked list
20: node *first = NULL;
21:
22:
23: // prototypes
24: void delete();
25: void find();
26: void insert();
27: void traverse();
28:
29:
30: int
31: main(int argc, char *argv[])
32: {
33:     int c;
34:     do
35:     {
36:         // print instructions
37:         printf("\nMENU\n\n"
38:                "1 - delete\n"
39:                "2 - find\n"
40:                "3 - insert\n"
41:                "4 - traverse\n"
42:                "0 - quit\n\n");
43:
44:         // get command
45:         printf("Command: ");
46:         c = GetInt();
47:
48:         // try to execute command
49:         switch (c)
50:         {
51:             case 1: delete(); break;
52:             case 2: find(); break;
53:             case 3: insert(); break;
54:             case 4: traverse(); break;
55:         }
56:     }
57:     while (c != 0);
58:
59:     // free list before quitting
60:     node *ptr = first;
61:     while (ptr != NULL)
62:     {
63:         node *predptr = ptr;
64:         ptr = ptr->next;
```

```
65:         free(predptr);
66:     }
67:     return 0;
68: }
69:
70:
71: /*
72:  * void
73:  * delete()
74:  *
75:  * Tries to delete a student.
76:  */
77: void
78: delete()
79: {
80:     // prompt user for ID
81:     printf("ID to delete: ");
82:     int n = GetInt();
83:
84:     // get list's first node
85:     node *ptr = first;
86:
87:     // try to delete student from list
88:     node *predptr = NULL;
89:     while (ptr != NULL)
90:     {
91:         // check for ID
92:         if (ptr->student->id == n)
93:         {
94:             // delete from head
95:             if (ptr == first)
96:             {
97:                 first = ptr->next;
98:                 free(ptr->student->name);
99:                 free(ptr->student->house);
100:                free(ptr->student);
101:                free(ptr);
102:            }
103:
104:            // delete from middle or tail
105:            else
106:            {
107:                predptr->next = ptr->next;
108:                if (ptr->student->name != NULL)
109:                    free(ptr->student->name);
110:                if (ptr->student->house != NULL)
111:                    free(ptr->student->house);
112:                free(ptr->student);
113:                free(ptr);
114:            }
115:
116:            // all done
117:            break;
118:        }
119:        else
120:        {
121:            predptr = ptr;
122:            ptr = ptr->next;
123:        }
124:    }
125:
126:    // traverse list
127:    traverse();
128: }
```



```
129:
130:
131: /*
132:  * void
133:  * insert()
134:  *
135:  * Tries to insert a student into list.
136:  */
137: void
138: insert()
139: {
140:     // try to instantiate node for student
141:     node *newptr = malloc(sizeof(node));
142:     if (newptr == NULL)
143:         return;
144:
145:     // initialize node
146:     newptr->next = NULL;
147:
148:     // try to instantiate student
149:     newptr->student = malloc(sizeof(student));
150:     if (newptr->student == NULL)
151:     {
152:         free(newptr);
153:         return;
154:     }
155:
156:     // try to initialize student
157:     printf("Student's ID: ");
158:     newptr->student->id = GetInt();
159:     printf("Student's name: ");
160:     newptr->student->name = GetString();
161:     printf("Student's house: ");
162:     newptr->student->house = GetString();
163:     if (newptr->student->name == NULL || newptr->student->house == NULL)
164:     {
165:         if (newptr->student->name != NULL)
166:             free(newptr->student->name);
167:         if (newptr->student->house != NULL)
168:             free(newptr->student->house);
169:         free(newptr->student);
170:         free(newptr);
171:         return;
172:     }
173:
174:     // check for empty list
175:     if (first == NULL)
176:         first = newptr;
177:
178:     // else check if student belongs at list's head
179:     else if (newptr->student->id < first->student->id)
180:     {
181:         newptr->next = first;
182:         first = newptr;
183:     }
184:
185:     // else try to insert student in middle or tail
186:     else
187:     {
188:         node *predptr = first;
189:         while (TRUE)
190:         {
191:             // avoid duplicates
192:             if (predptr->student->id == newptr->student->id)
```

```
193:             {
194:                 free(newptr->student->name);
195:                 free(newptr->student->house);
196:                 free(newptr->student);
197:                 free(newptr);
198:                 break;
199:             }
200:
201:             // check for insertion at tail
202:             else if (predptr->next == NULL)
203:             {
204:                 predptr->next = newptr;
205:                 break;
206:             }
207:
208:             // check for insertion in middle
209:             else if (predptr->next->student->id > newptr->student->id)
210:             {
211:                 newptr->next = predptr->next;
212:                 predptr->next = newptr;
213:                 break;
214:             }
215:
216:             // update pointer
217:             predptr = predptr->next;
218:         }
219:     }
220:
221:     // traverse list
222:     traverse();
223: }
224:
225:
226: /*
227:  * void
228:  * find()
229:  *
230:  * Tries to find a number in list.
231:  */
232: void
233: find()
234: {
235:     // prompt user for ID
236:     printf("ID to find: ");
237:     int id = GetInt();
238:
239:     // get list's first node
240:     node *ptr = first;
241:
242:     // try to find student
243:     while (ptr != NULL)
244:     {
245:         if (ptr->student->id == id)
246:         {
247:             printf("\nFound %s of %s (%d)!\n",
248:                    ptr->student->name, ptr->student->house, id);
249:             sleep(1);
250:             break;
251:         }
252:         ptr = ptr->next;
253:     }
254: }
255:
256:
```



```
257: /*
258:  * void
259:  * traverse()
260:  *
261:  * Traverses list, printing its numbers.
262:  */
263: void
264: traverse()
265: {
266:     // traverse list
267:     printf("\nLIST IS NOW: ");
268:     node *ptr = first;
269:     while (ptr != NULL)
270:     {
271:         printf("%s of %s (%d)  ",
272:                ptr->student->name, ptr->student->house, ptr->student->id);
273:         ptr = ptr->next;
274:     }
275:
276:     // flush standard output since we haven't outputted any newlines yet
277:     fflush(stdout);
278:
279:     // pause before continuing
280:     sleep(1);
281:     printf("\n\n");
282: }
283:
```

```
 1: /**********************************************************************
 2:  * list2.h
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Defines structures for students and linked lists thereof.
 8:  **********************************************************************/
 9:
10:
11: typedef struct
12: {
13:     int id;
14:     char *name;
15:     char *house;
16: }
17: student;
18:
19:
20: typedef struct _node
21: {
22:     student *student;
23:     struct _node *next;
24: }
25: node;
26:
```