```c
/**
 * list-0.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Demonstrates a linked list for numbers.
 */

#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "list-0.h"

// linked list
node* first = NULL;

// prototypes
void delete(void);
void insert(void);
void search(void);
void traverse(void);

int main(void)
{
    int c;
    do
    {
        // print instructions
        printf("\nMENU\n\n"
            "1 - delete\n"
            "2 - insert\n"
            "3 - search \n"
            "4 - traverse\n"
            "0 - quit\n\n");

        // get command
        printf("Command: ");
        c = GetInt();

        // try to execute command
        switch (c)
        {
            case 1: delete(); break;
            case 2: insert(); break;
            case 3: search(); break;
```

```
            case 4: traverse(); break;
        }
    }
    while (c != 0);

    // free list before quitting
    node* ptr = first;
    while (ptr != NULL)
    {
        node* predptr = ptr;
        ptr = ptr->next;
        free(predptr);
    }
}

/**
 * Tries to delete a number.
 */
void delete(void)
{
    // prompt user for number
    printf("Number to delete: ");
    int n = GetInt();

    // get list's first node
    node* ptr = first;

    // try to delete number from list
    node* predptr = NULL;
    while (ptr != NULL)
    {
        // check for number
        if (ptr->n == n)
        {
            // delete from head
            if (ptr == first)
            {
                first = ptr->next;
                free(ptr);
            }

            // delete from middle or tail
            else
            {
                predptr->next = ptr->next;
                free(ptr);
            }

```

```
 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.     {
```

```c
        if (ptr->n == n)
        {
            printf("\nFound %i!\n", n);
            sleep(1);
            break;
        }
        ptr = ptr->next;
    }
}

/**
 * Traverses list, printing its numbers.
 */
void traverse(void)
{
    // traverse list
    printf("\nLIST IS NOW: ");
    node* ptr = first;
    while (ptr != NULL)
    {
        printf("%i ", ptr->n);
        ptr = ptr->next;
    }

    // flush standard output since we haven't outputted any newlines yet
    fflush(stdout);

    // pause before continuing
    sleep(1);
    printf("\n\n");
}
```

```c
/**
 * list-0.h
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Defines a node for a linked list of integers.
 */

typedef struct node
{
    int n;
    struct node* next;
}
node;
```

```c
/**
 * list-1.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Demonstrates a linked list for students.
 */

#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "list-1.h"

// linked list
node* first = NULL;

// prototypes
void delete(void);
void insert(void);
void search(void);
void traverse(void);

int main(void)
{
    int c;
    do
    {
        // print instructions
        printf("\nMENU\n\n"
            "1 - delete\n"
            "2 - insert\n"
            "3 - search\n"
            "4 - traverse\n"
            "0 - quit\n\n");

        // get command
        printf("Command: ");
        c = GetInt();

        // try to execute command
        switch (c)
        {
            case 1: delete(); break;
            case 2: insert(); break;
            case 3: search(); break;
```

```c
            case 4: traverse(); break;
        }
    }
    while (c != 0);

    // free list before quitting
    node* ptr = first;
    while (ptr != NULL)
    {
        node* predptr = ptr;
        ptr = ptr->next;
        if (predptr->student != NULL)
        {
            if (predptr->student->name != NULL)
            {
                free(predptr->student->name);
            }
            if (predptr->student->house != NULL)
            {
                free(predptr->student->house);
            }
            free(predptr->student);
        }
        free(predptr);
    }
}

/**
 * Tries to delete a student.
 */
void delete(void)
{
    // prompt user for ID
    printf("ID to delete: ");
    int n = GetInt();

    // get list's first node
    node* ptr = first;

    // try to delete student from list
    node* predptr = NULL;
    while (ptr != NULL)
    {
        // check for ID
        if (ptr->student->id == n)
        {
            // delete from head
            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;
```

```c
    }

    // initialize node
    newptr->next = NULL;

    // try to instantiate student
    newptr->student = malloc(sizeof(student));
    if (newptr->student == NULL)
    {
        free(newptr);
        return;
    }

    // try to initialize student
    printf("Student's ID: ");
    newptr->student->id = GetInt();
    printf("Student's name: ");
    newptr->student->name = GetString();
    printf("Student's house: ");
    newptr->student->house = GetString();
    if (newptr->student->name == NULL || newptr->student->house == NULL)
    {
        if (newptr->student->name != NULL)
        {
            free(newptr->student->name);
        }
        if (newptr->student->house != NULL)
        {
            free(newptr->student->house);
        }
        free(newptr->student);
        free(newptr);
        return;
    }

    // check for empty list
    if (first == NULL)
    {
        first = newptr;
    }

    // else check if student belongs at list's head
    else if (newptr->student->id < first->student->id)
    {
        newptr->next = first;
        first = newptr;
    }

```

```
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: ");
```

```c
    int id = GetInt();

    // get list's first node
    node* ptr = first;

    // search for student
    while (ptr != NULL)
    {
        if (ptr->student->id == id)
        {
            printf("\nFound %s of %s (%i)!\n",
                ptr->student->name, ptr->student->house, id);
            sleep(1);
            break;
        }
        ptr = ptr->next;
    }
}

/**
 * Traverses list, printing its numbers.
 */
void traverse(void)
{
    // traverse list
    printf("\nLIST IS NOW: ");
    node* ptr = first;
    while (ptr != NULL)
    {
        printf("%s of %s (%i)  ",
            ptr->student->name, ptr->student->house, ptr->student->id);
        ptr = ptr->next;
    }

    // flush standard output since we haven't outputted any newlines yet
    fflush(stdout);

    // pause before continuing
    sleep(1);
    printf("\n\n");
}
```

```c
/**
 * list-1.h
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Defines structures for students and linked lists thereof.
 */

typedef struct
{
    int id;
    char* name;
    char* house;
}
student;

typedef struct node
{
    student* student;
    struct node* next;
}
node;
```