```c
/****************************************************************************
 * list1.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 "list1.h"

// linked list
node* first = NULL;

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

int main(void)
{
    int c;
    do
    {
        // print instructions
        printf("\nMENU\n\n"
               "1 - delete\n"
               "2 - find\n"
               "3 - insert\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: find(); break;
            case 3: insert(); 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;
        free(predptr);
    }

    return 0;
}

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

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

```c
            printf("\nFound %d!\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("%d ", 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
/**************************************************************************
 * list1.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
/****************************************************************************
 * list2.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 "list2.h"

// linked list
node* first = NULL;

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

int main(void)
{
    int c;
    do
    {
        // print instructions
        printf("\nMENU\n\n"
               "1 - delete\n"
               "2 - find\n"
               "3 - insert\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: find(); break;
            case 3: insert(); 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);
    }
    return 0;
}


/**
 * 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)
            {
                first = ptr->next;
                free(ptr->student->name);
                free(ptr->student->house);
                free(ptr->student);
                free(ptr);
            }

            // delete from middle or tail
            else
```

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

```c
    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;
    }

    // else try to insert student in middle or tail
    else
    {
        node* predptr = first;
        while (true)
        {
            // avoid duplicates
            if (predptr->student->id == newptr->student->id)
            {
                free(newptr->student->name);
                free(newptr->student->house);
                free(newptr->student);
                free(newptr);
                break;
            }

            // check for insertion at tail
            else if (predptr->next == NULL)
            {
                predptr->next = newptr;
                break;
```

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

```c
 */
void traverse(void)
{
    // traverse list
    printf("\nLIST IS NOW: ");
    node* ptr = first;
    while (ptr != NULL)
    {
        printf("%s of %s (%d)  ",
         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
/**************************************************************************
 * list2.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;
```

```c
/**************************************************************************
 * memory.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Demonstrates memory-related errors.
 *
 * problem 1: heap block overrun
 * problem 2: memory leak -- x not freed
 *
 * Adapted from
 * http://valgrind.org/docs/manual/quick-start.html#quick-start.prepare.
 **************************************************************************/

#include <stdlib.h>

void f(void)
{
    int *x = malloc(10 * sizeof(int));
    x[10] = 0;
}

int main(void)
{
    f();
    return 0;
}
```