```c
/**************************************************************************
 * bmp.h
 *
 * Computer Science 50
 * Problem Set 5
 *
 * BMP-related data types based on Microsoft's own.
 **************************************************************************/

#include <stdint.h>


/*
 * Common Data Types
 *
 * The data types in this section are essentially aliases for C/C++
 * primitive data types.
 *
 * Adapted from http://msdn.microsoft.com/en-us/library/cc230309(PROT.10).aspx.
 * See http://en.wikipedia.org/wiki/Stdint.h for more on stdint.h.
 */

typedef uint8_t  BYTE;
typedef uint32_t DWORD;
typedef int32_t  LONG;
typedef uint16_t WORD;


/*
 * BITMAPFILEHEADER
 *
 * The BITMAPFILEHEADER structure contains information about the type, size,
 * and layout of a file that contains a DIB [device-independent bitmap].
 *
 * Adapted from http://msdn.microsoft.com/en-us/library/dd183374(VS.85).aspx.
 */

typedef struct
{
    WORD   bfType;
    DWORD  bfSize;
    WORD   bfReserved1;
    WORD   bfReserved2;
    DWORD  bfOffBits;
} __attribute__((__packed__))
BITMAPFILEHEADER;


```

```
49. /*
50.  * BITMAPINFOHEADER
51.  *
52.  * The BITMAPINFOHEADER structure contains information about the
53.  * dimensions and color format of a DIB [device-independent bitmap].
54.  *
55.  * Adapted from http://msdn.microsoft.com/en-us/library/dd183376(VS.85).aspx.
56.  */
57.
58. typedef struct
59. {
60.     DWORD  biSize;
61.     LONG   biWidth;
62.     LONG   biHeight;
63.     WORD   biPlanes;
64.     WORD   biBitCount;
65.     DWORD  biCompression;
66.     DWORD  biSizeImage;
67.     LONG   biXPelsPerMeter;
68.     LONG   biYPelsPerMeter;
69.     DWORD  biClrUsed;
70.     DWORD  biClrImportant;
71. } __attribute__((__packed__))
72. BITMAPINFOHEADER;
73.
74.
75. /*
76.  * RGBTRIPLE
77.  *
78.  * This structure describes a color consisting of relative intensities of
79.  * red, green, and blue.
80.  *
81.  * Adapted from http://msdn.microsoft.com/en-us/library/aa922590.aspx.
82.  */
83.
84. typedef struct
85. {
86.     BYTE  rgbtBlue;
87.     BYTE  rgbtGreen;
88.     BYTE  rgbtRed;
89. } __attribute__((__packed__))
90. RGBTRIPLE;
```

```c
/****************************************************************************
 * list1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * 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
```

```
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.  * Tries to delete a number.
73.  */
74.
75. void
76. delete(void)
77. {
78.     // prompt user for number
79.     printf("Number to delete: ");
80.     int n = GetInt();
81.
82.     // get list's first node
83.     node *ptr = first;
84.
85.     // try to delete number from list
86.     node *predptr = NULL;
87.     while (ptr != NULL)
88.     {
89.         // check for number
90.         if (ptr->n == n)
91.         {
92.             // delete from head
93.             if (ptr == first)
94.             {
95.                 first = ptr->next;
96.                 free(ptr);
```

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

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

```c
{
    // prompt user for number
    printf("Number to find: ");
    int n = GetInt();

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

    // try to find number
    while (ptr != NULL)
    {
        if (ptr->n == n)
        {
            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
 *
 * Computer Science 50
 * David J. Malan
 *
 * Defines a node for a linked list of integers.
 **********************************************************************/


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

```c
/****************************************************************************
 * list2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * 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
```

```
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.  * Tries to delete a student.
73.  */
74.
75. void
76. delete(void)
77. {
78.     // prompt user for ID
79.     printf("ID to delete: ");
80.     int n = GetInt();
81.
82.     // get list's first node
83.     node *ptr = first;
84.
85.     // try to delete student from list
86.     node *predptr = NULL;
87.     while (ptr != NULL)
88.     {
89.         // check for ID
90.         if (ptr->student->id == n)
91.         {
92.             // delete from head
93.             if (ptr == first)
94.             {
95.                 first = ptr->next;
96.                 free(ptr->student->name);
```

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

```c
    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;
    }

    // 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);
```

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

```c
            printf("\nFound %s of %s (%d)!\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 (%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
 *
 * Computer Science 50
 * David J. Malan
 *
 * 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
 *
 * Computer Science 50
 * David J. Malan
 *
 * 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;
}
```

```c
/**********************************************************************
 * structs.h
 *
 * Computer Science 50
 * David J. Malan
 *
 * Defines a student for structs{1,2}.c.
 **********************************************************************/


// structure representing a student
typedef struct
{
    int id;
    char *name;
    char *house;
}
student;
```

```c
/**************************************************************************
 * structs1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Demonstrates use of structs.
 **************************************************************************/

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

#include "structs.h"


// class size
#define STUDENTS 3


int
main(void)
{
    // declare class
    student class[STUDENTS];

    // populate class with user's input
    for (int i = 0; i < STUDENTS; i++)
    {
        printf("Student's ID: ");
        class[i].id = GetInt();

        printf("Student's name: ");
        class[i].name = GetString();

        printf("Student's house: ");
        class[i].house = GetString();
        printf("\n");
    }

    // now print anyone in Mather
    for (int i = 0; i < STUDENTS; i++)
        if (strcmp(class[i].house, "Mather") == 0)
            printf("%s is in Mather!\n\n", class[i].name);

    // free memory
    for (int i = 0; i < STUDENTS; i++)
```

```
49.     {
50.         free(class[i].name);
51.         free(class[i].house);
52.     }
53. }
```

```c
/**************************************************************************
 * structs.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Demonstrates use of structs.
 **************************************************************************/

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

#include "structs.h"


// class size
#define STUDENTS 3


int
main(void)
{
    // declare class
    student class[STUDENTS];

    // populate class with user's input
    for (int i = 0; i < STUDENTS; i++)
    {
        printf("Student's ID: ");
        class[i].id = GetInt();

        printf("Student's name: ");
        class[i].name = GetString();

        printf("Student's house: ");
        class[i].house = GetString();
        printf("\n");
    }

    // now print anyone in Mather
    for (int i = 0; i < STUDENTS; i++)
        if (strcmp(class[i].house, "Mather") == 0)
            printf("%s is in Mather!\n\n", class[i].name);

    // let's save these students to disk
    FILE *fp = fopen("database", "w");
```

```c
    if (fp != NULL)
    {
        for (int i = 0; i < STUDENTS; i++)
        {
            fprintf(fp, "%d\n", class[i].id);
            fprintf(fp, "%s\n", class[i].name);
            fprintf(fp, "%s\n", class[i].house);
        }
        fclose(fp);
    }

    // free memory
    for (int i = 0; i < STUDENTS; i++)
    {
        free(class[i].name);
        free(class[i].house);
    }
}
```

```c
/**********************************************************************
 * uint.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Prints a signed 8-bit integer and an unsigned 8-bit integer.
 *
 * Demonstrates signed, fixed-width types.
 **********************************************************************/

#include <stdint.h>
#include <stdio.h>


int
main(void)
{
    // declare and print signed 8-bit integer
    int8_t i = 0xff;
    printf("%d\n", i);

    // declare and print signed 8-bit integer
    uint8_t u = 0xff;
    printf("%d\n", u);
}
```