```
/*****************************************************************************
 * cs50.c
 *
 * version 1.1.6
 *
 * Computer Science 50
 * Glenn Holloway
 * David J. Malan
 *
 * Definitions for the CS50 Library.
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * The latest version of this file can be found at
 * http://www.cs50.net/pub/releases/cs50/cs50.c.
 *
 * To compile as a static library on your own system:
 * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
 * % ar rcs libcs50.a cs50.o
 * % rm -f cs50.o
 * % cp cs50.h /usr/local/include
 * % cp libcs50.a /usr/local/lib
 *****************************************************************************/

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

#include "cs50.h"


/*
 * Default capacity of buffer for standard input.
 */

#define CAPACITY 128


/*
 * Reads a line of text from standard input and returns the equivalent
 * char; if text does not represent a char, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  If line can't be read,
 * returns CHAR_MAX.
 */

char
GetChar(void)
{
    // try to get a char from user
    while (true)
    {
        // get line of text, returning CHAR_MAX on failure
        string line = GetString();
        if (line == NULL)
            return CHAR_MAX;

        // return a char if only a char (possibly with
        // leading and/or trailing whitespace) was provided
        char c1, c2;
        if (sscanf(line, " %c %c", &c1, &c2) == 1)
        {
            free(line);
            return c1;
        }
        else
```



```
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns the equivalent
 * double as precisely as possible; if text does not represent a
 * double, user is prompted to retry.  Leading and trailing whitespace
 * is ignored.  For simplicity, overflow and underflow are not detected.
 * If line can't be read, returns DBL_MAX.
 */

double
GetDouble(void)
{
    // try to get a double from user
    while (true)
    {
        // get line of text, returning DBL_MAX on failure
        string line = GetString();
        if (line == NULL)
            return DBL_MAX;

        // return a double if only a double (possibly with
        // leading and/or trailing whitespace) was provided
        double d; char c;
        if (sscanf(line, " %lf %c", &d, &c) == 1)
        {
            free(line);
            return d;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns the equivalent
 * float as precisely as possible; if text does not represent a float,
 * user is prompted to retry.  Leading and trailing whitespace is ignored.
 * For simplicity, overflow and underflow are not detected.  If line can't
 * be read, returns FLT_MAX.
 */

float
GetFloat(void)
{
    // try to get a float from user
    while (true)
    {
        // get line of text, returning FLT_MAX on failure
        string line = GetString();
        if (line == NULL)
            return FLT_MAX;

        // return a float if only a float (possibly with
```

```
        // leading and/or trailing whitespace) was provided
        char c; float f;
        if (sscanf(line, " %f %c", &f, &c) == 1)
        {
            free(line);
            return f;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns it as an
 * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
 * does not represent such an int, user is prompted to retry.  Leading
 * and trailing whitespace is ignored.  For simplicity, overflow is not
 * detected.  If line can't be read, returns INT_MAX.
 */

int
GetInt(void)
{
    // try to get an int from user
    while (true)
    {
        // get line of text, returning INT_MAX on failure
        string line = GetString();
        if (line == NULL)
            return INT_MAX;

        // return an int if only an int (possibly with
        // leading and/or trailing whitespace) was provided
        int n; char c;
        if (sscanf(line, " %d %c", &n, &c) == 1)
        {
            free(line);
            return n;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns an equivalent
 * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
 * does not represent such a long long, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  For simplicity, overflow
 * is not detected.  If line can't be read, returns LLONG_MAX.
 */

long long
GetLongLong(void)
{
    // try to get a long long from user
```



```
    while (true)
    {
        // get line of text, returning LLONG_MAX on failure
        string line = GetString();
        if (line == NULL)
            return LLONG_MAX;

        // return a long long if only a long long (possibly with
        // leading and/or trailing whitespace) was provided
        long long n; char c;
        if (sscanf(line, " %lld %c", &n, &c) == 1)
        {
            free(line);
            return n;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns it as a string,
 * sans trailing newline character.  (Ergo, if user inputs only "\n",
 * returns "" not NULL.)  Leading and trailing whitespace is not ignored.
 * Returns NULL upon error or no input whatsoever (i.e., just EOF).
 */

string
GetString(void)
{
    // growable buffer for chars
    string buffer = NULL;

    // capacity of buffer
    unsigned int capacity = 0;

    // number of chars actually in buffer
    unsigned int n = 0;

    // character read or EOF
    int c;

    // iteratively get chars from standard input
    while ((c = fgetc(stdin)) != '\n' && c != EOF)
    {
        // grow buffer if necessary
        if (n + 1 > capacity)
        {
            // determine new capacity: start at CAPACITY then double
            if (capacity == 0)
                capacity = CAPACITY;
            else if (capacity <= (UINT_MAX / 2))
                capacity *= 2;
            else
            {
                free(buffer);
                return NULL;
            }

            // extend buffer's capacity
```

```
            string temp = realloc(buffer, capacity * sizeof(char));
            if (temp == NULL)
            {
                free(buffer);
                return NULL;
            }
            buffer = temp;
        }

        // append current character to buffer
        buffer[n++] = c;
    }

    // return NULL if user provided no input
    if (n == 0 && c == EOF)
        return NULL;

    // minimize buffer
    string minimal = malloc((n + 1) * sizeof(char));
    strncpy(minimal, buffer, n);
    free(buffer);

    // terminate string
    minimal[n] = '\0';

    // return string
    return minimal;
}
```



```
/*****************************************************************************
 * cs50.h
 *
 * version 1.1.6
 *
 * Computer Science 50
 * Glenn Holloway
 * David J. Malan
 *
 * Declarations for the CS50 Library.
 * Based on Eric Roberts' genlib.h and simpio.h.
 *
 * The latest version of this file can be found at
 * http://www.cs50.net/pub/releases/cs50/cs50.h.
 *
 * To compile as a static library on your own system:
 * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
 * % ar rcs libcs50.a cs50.o
 * % rm -f cs50.o
 * % cp cs50.h /usr/local/include
 * % cp libcs50.a /usr/local/lib
 *****************************************************************************/

#ifndef _CS50_H
#define _CS50_H

#include <float.h>
#include <limits.h>


/*
 * Borrow the standard library's data type for Boolean variables whose
 * values must be (true|false).
 */

#include <stdbool.h>


/*
 * Our own data type for string variables.
 */

typedef char *string;


/*
 * Reads a line of text from standard input and returns the equivalent
 * char; if text does not represent a char, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  If line can't be read,
 * returns CHAR_MAX.
 */

char
GetChar(void);


/*
 * Reads a line of text from standard input and returns the equivalent
 * double as precisely as possible; if text does not represent a
 * double, user is prompted to retry.  Leading and trailing whitespace
 * is ignored.  For simplicity, overflow and underflow are not detected.
 * If line can't be read, returns DBL_MAX.
 */

```

```
double
GetDouble(void);


/*
 * Reads a line of text from standard input and returns the equivalent
 * float as precisely as possible; if text does not represent a float,
 * user is prompted to retry.  Leading and trailing whitespace is ignored.
 * For simplicity, overflow and underflow are not detected.  If line can't
 * be read, returns FLT_MAX.
 */

float
GetFloat(void);


/*
 * Reads a line of text from standard input and returns it as an
 * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
 * does not represent such an int, user is prompted to retry.  Leading
 * and trailing whitespace is ignored.  For simplicity, overflow is not
 * detected.  If line can't be read, returns INT_MAX.
 */

int
GetInt(void);


/*
 * Reads a line of text from standard input and returns an equivalent
 * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
 * does not represent such a long long, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  For simplicity, overflow
 * is not detected.  If line can't be read, returns LLONG_MAX.
 */

long long
GetLongLong(void);


/*
 * Reads a line of text from standard input and returns it as a string,
 * sans trailing newline character.  (Ergo, if user inputs only "\n",
 * returns "" not NULL.)  Leading and trailing whitespace is not ignored.
 * Returns NULL upon error or no input whatsoever (i.e., just EOF).
 */

string GetString(void);


#endif
```



```
/*****************************************************************************
 * 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
        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 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);
            }

            // all done
            break;
        }
        else
        {
            predptr = ptr;
            ptr = ptr->next;
        }
    }

    // traverse list
    traverse();
}


/*
 * Tries to insert a number into list.
 */

void
insert(void)
{
    // try to instantiate node for number
```



```
    node *newptr = malloc(sizeof(node));
    if (newptr == NULL)
        return;

    // initialize node
    printf("Number to insert: ");
    newptr->n = GetInt();
    newptr->next = NULL;

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

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

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

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

            // check for insertion in middle
            else if (predptr->next->n > newptr->n)
            {
                newptr->next = predptr->next;
                predptr->next = newptr;
                break;
            }

            // update pointer
            predptr = predptr->next;
        }
    }

    // traverse list
    traverse();
}


/*
 * Tries to find a number in list.
 */

void
find(void)
```

```
{
    // 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");
}
```



```
/*****************************************************************************
 * 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;
```

```
 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(void);
25: void find(void);
26: void insert(void);
27: void traverse(void);
28:
29:
30: int
31: main(void)
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:  * 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:
```

```
/*
 * Tries to insert a student into list.
 */

void
insert(void)
{
    // try to instantiate node for student
    node *newptr = malloc(sizeof(node));
    if (newptr == NULL)
        return;

    // 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;
    }

    // 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;
            }

            // check for insertion in middle
            else if (predptr->next->student->id > newptr->student->id)
            {
                newptr->next = predptr->next;
                predptr->next = newptr;
                break;
            }

            // update pointer
            predptr = predptr->next;
        }
    }

    // traverse list
    traverse();
}


/*
 * Tries to find a number in list.
 */

void
find(void)
{
    // prompt user for ID
    printf("ID to find: ");
    int id = GetInt();

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

    // try to find student
    while (ptr != NULL)
    {
        if (ptr->student->id == id)
        {
            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");
}
```



```
/*****************************************************************************
 * 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;
```

```
/*****************************************************************************
 * 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;
```



```
/*****************************************************************************
 * 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++)
    {
        free(class[i].name);
        free(class[i].house);
    }
}
```

```
/*****************************************************************************
 * 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");
    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);
```



```
    }
}
```