```c
/**************************************************************************
 * CS50 Library 6
 * https://manual.cs50.net/library/
 *
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * Copyright (c) 2013,
 * Glenn Holloway <holloway@eecs.harvard.edu>
 * David J. Malan <malan@harvard.edu>
 * All rights reserved.
 *
 * BSD 3-Clause License
 * http://www.opensource.org/licenses/BSD-3-Clause
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 * * Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 * * Neither the name of CS50 nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without
 *   specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 **************************************************************************/

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

#include "cs50.h"

/**
 * 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)
        {
```

```
 97.             return DBL_MAX;
 98.         }
 99.
100.         // return a double if only a double (possibly with
101.         // leading and/or trailing whitespace) was provided
102.         double d; char c;
103.         if (sscanf(line, " %lf %c", &d, &c) == 1)
104.         {
105.             free(line);
106.             return d;
107.         }
108.         else
109.         {
110.             free(line);
111.             printf("Retry: ");
112.         }
113.     }
114. }
115.
116. /**
117.  * Reads a line of text from standard input and returns the equivalent
118.  * float as precisely as possible; if text does not represent a float,
119.  * user is prompted to retry.  Leading and trailing whitespace is ignored.
120.  * For simplicity, overflow and underflow are not detected.  If line can't
121.  * be read, returns FLT_MAX.
122.  */
123. float GetFloat(void)
124. {
125.     // try to get a float from user
126.     while (true)
127.     {
128.         // get line of text, returning FLT_MAX on failure
129.         string line = GetString();
130.         if (line == NULL)
131.         {
132.             return FLT_MAX;
133.         }
134.
135.         // return a float if only a float (possibly with
136.         // leading and/or trailing whitespace) was provided
137.         char c; float f;
138.         if (sscanf(line, " %f %c", &f, &c) == 1)
139.         {
140.             free(line);
141.             return f;
142.         }
143.         else
144.         {
```

```
145.             free(line);
146.             printf("Retry: ");
147.         }
148.     }
149. }
150.
151. /**
152.  * Reads a line of text from standard input and returns it as an
153.  * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
154.  * does not represent such an int, user is prompted to retry.  Leading
155.  * and trailing whitespace is ignored.  For simplicity, overflow is not
156.  * detected.  If line can't be read, returns INT_MAX.
157.  */
158. int GetInt(void)
159. {
160.     // try to get an int from user
161.     while (true)
162.     {
163.         // get line of text, returning INT_MAX on failure
164.         string line = GetString();
165.         if (line == NULL)
166.         {
167.             return INT_MAX;
168.         }
169.
170.         // return an int if only an int (possibly with
171.         // leading and/or trailing whitespace) was provided
172.         int n; char c;
173.         if (sscanf(line, " %i %c", &n, &c) == 1)
174.         {
175.             free(line);
176.             return n;
177.         }
178.         else
179.         {
180.             free(line);
181.             printf("Retry: ");
182.         }
183.     }
184. }
185.
186. /**
187.  * Reads a line of text from standard input and returns an equivalent
188.  * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
189.  * does not represent such a long long, user is prompted to retry.
190.  * Leading and trailing whitespace is ignored.  For simplicity, overflow
191.  * is not detected.  If line can't be read, returns LLONG_MAX.
192.  */
```

```
193. long long GetLongLong(void)
194. {
195.     // try to get a long long from user
196.     while (true)
197.     {
198.         // get line of text, returning LLONG_MAX on failure
199.         string line = GetString();
200.         if (line == NULL)
201.         {
202.             return LLONG_MAX;
203.         }
204.
205.         // return a long long if only a long long (possibly with
206.         // leading and/or trailing whitespace) was provided
207.         long long n; char c;
208.         if (sscanf(line, " %lld %c", &n, &c) == 1)
209.         {
210.             free(line);
211.             return n;
212.         }
213.         else
214.         {
215.             free(line);
216.             printf("Retry: ");
217.         }
218.     }
219. }
220.
221. /**
222.  * Reads a line of text from standard input and returns it as a
223.  * string (char*), sans trailing newline character.  (Ergo, if
224.  * user inputs only "\n", returns "" not NULL.)  Returns NULL
225.  * upon error or no input whatsoever (i.e., just EOF).  Leading
226.  * and trailing whitespace is not ignored.  Stores string on heap
227.  * (via malloc); memory must be freed by caller to avoid leak.
228.  */
229. string GetString(void)
230. {
231.     // growable buffer for chars
232.     string buffer = NULL;
233.
234.     // capacity of buffer
235.     unsigned int capacity = 0;
236.
237.     // number of chars actually in buffer
238.     unsigned int n = 0;
239.
240.     // character read or EOF
```

```
241.     int c;
242.
243.     // iteratively get chars from standard input
244.     while ((c = fgetc(stdin)) != '\n' && c != EOF)
245.     {
246.         // grow buffer if necessary
247.         if (n + 1 > capacity)
248.         {
249.             // determine new capacity: start at 32 then double
250.             if (capacity == 0)
251.             {
252.                 capacity = 32;
253.             }
254.             else if (capacity <= (UINT_MAX / 2))
255.             {
256.                 capacity *= 2;
257.             }
258.             else
259.             {
260.                 free(buffer);
261.                 return NULL;
262.             }
263.
264.             // extend buffer's capacity
265.             string temp = realloc(buffer, capacity * sizeof(char));
266.             if (temp == NULL)
267.             {
268.                 free(buffer);
269.                 return NULL;
270.             }
271.             buffer = temp;
272.         }
273.
274.         // append current character to buffer
275.         buffer[n++] = c;
276.     }
277.
278.     // return NULL if user provided no input
279.     if (n == 0 && c == EOF)
280.     {
281.         return NULL;
282.     }
283.
284.     // minimize buffer
285.     string minimal = malloc((n + 1) * sizeof(char));
286.     strncpy(minimal, buffer, n);
287.     free(buffer);
288.
```

```
289.     // terminate string
290.     minimal[n] = '\0';
291.
292.     // return string
293.     return minimal;
294. }
```

```
/**************************************************************************
 * CS50 Library 6
 * https://manual.cs50.net/library/
 *
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * Copyright (c) 2013,
 * Glenn Holloway <holloway@eecs.harvard.edu>
 * David J. Malan <malan@harvard.edu>
 * All rights reserved.
 *
 * BSD 3-Clause License
 * http://www.opensource.org/licenses/BSD-3-Clause
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 * * Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 * * Neither the name of CS50 nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without
 *   specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 **************************************************************************/

#ifndef _CS50_H
#define _CS50_H

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

```

```
49. /**
50.  * Our own data type for string variables.
51.  */
52. typedef char* string;
53.
54. /**
55.  * Reads a line of text from standard input and returns the equivalent
56.  * char; if text does not represent a char, user is prompted to retry.
57.  * Leading and trailing whitespace is ignored.  If line can't be read,
58.  * returns CHAR_MAX.
59.  */
60. char GetChar(void);
61.
62. /**
63.  * Reads a line of text from standard input and returns the equivalent
64.  * double as precisely as possible; if text does not represent a
65.  * double, user is prompted to retry.  Leading and trailing whitespace
66.  * is ignored.  For simplicity, overflow and underflow are not detected.
67.  * If line can't be read, returns DBL_MAX.
68.  */
69. double GetDouble(void);
70.
71. /**
72.  * Reads a line of text from standard input and returns the equivalent
73.  * float as precisely as possible; if text does not represent a float,
74.  * user is prompted to retry.  Leading and trailing whitespace is ignored.
75.  * For simplicity, overflow and underflow are not detected.  If line can't
76.  * be read, returns FLT_MAX.
77.  */
78. float GetFloat(void);
79.
80. /**
81.  * Reads a line of text from standard input and returns it as an
82.  * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
83.  * does not represent such an int, user is prompted to retry.  Leading
84.  * and trailing whitespace is ignored.  For simplicity, overflow is not
85.  * detected.  If line can't be read, returns INT_MAX.
86.  */
87. int GetInt(void);
88.
89. /**
90.  * Reads a line of text from standard input and returns an equivalent
91.  * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
92.  * does not represent such a long long, user is prompted to retry.
93.  * Leading and trailing whitespace is ignored.  For simplicity, overflow
94.  * is not detected.  If line can't be read, returns LLONG_MAX.
95.  */
96. long long GetLongLong(void);
```

```
 97.
 98. /**
 99.  * Reads a line of text from standard input and returns it as a
100.  * string (char *), sans trailing newline character.  (Ergo, if
101.  * user inputs only "\n", returns "" not NULL.)  Returns NULL
102.  * upon error or no input whatsoever (i.e., just EOF).  Leading
103.  * and trailing whitespace is not ignored.  Stores string on heap
104.  * (via malloc); memory must be freed by caller to avoid leak.
105.  */
106. string GetString(void);
107.
108. #endif
```

```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;
}
```

```c
/**
 * scanf-0.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Reads a number from the user into an int.
 *
 * Demonstrates scanf and address-of operator.
 */

#include <stdio.h>

int main(void)
{
    int x;
    printf("Number please: ");
    scanf("%i", &x);
    printf("Thanks for the %i!\n", x);
}
```

```c
/**
 * scanf-1.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Reads a string from the user into memory it shouldn't.
 *
 * Demonstrates possible attack!
 */

#include <stdio.h>

int main(void)
{
    char* buffer;
    printf("String please: ");
    scanf("%s", buffer);
    printf("Thanks for the %s!\n", buffer);
}
```

```c
/**
 * scanf-2.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Reads a string from the user into an array (dangerously).
 *
 * Demonstrates potential buffer overflow!
 */

#include <stdio.h>

int main(void)
{
    char buffer[16];
    printf("String please: ");
    scanf("%s", buffer);
    printf("Thanks for the %s!\n", buffer);
}
```

```c
/**
 * swap.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Swaps two variables' values.
 *
 * Demonstrates passing by reference.
 */

#include <stdio.h>

// function prototype
void swap(int* a, int* b);

int main(void)
{
    int x = 1;
    int y = 2;

    printf("x is %i\n", x);
    printf("y is %i\n", y);
    printf("Swapping...\n");
    swap(&x, &y);
    printf("Swapped!\n");
    printf("x is %i\n", x);
    printf("y is %i\n", y);
}

/**
 * Swap arguments' values.
 */
void swap(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```