

```

1.  /*****
2.   * CS50 Library 4
3.   * https://manual.cs50.net/Library
4.   *
5.   * Based on Eric Roberts' genlib.c and simpio.c.
6.   *
7.   * Copyright (c) 2012
8.   * Glenn Holloway <holloway@eecs.harvard.edu>
9.   * David J. Malan <malan@harvard.edu>
10.  * All Rights Reserved
11.  *
12.  * BSD 3-Clause License
13.  * http://www.opensource.org/licenses/BSD-3-Clause
14.  *
15.  * Redistribution and use in source and binary forms, with or without
16.  * modification, are permitted provided that the following conditions are
17.  * met:
18.  *
19.  * * Redistributions of source code must retain the above copyright notice,
20.  *   this list of conditions and the following disclaimer.
21.  * * Redistributions in binary form must reproduce the above copyright
22.  *   notice, this list of conditions and the following disclaimer in the
23.  *   documentation and/or other materials provided with the distribution.
24.  * * Neither the name of CS50 nor the names of its contributors may be used
25.  *   to endorse or promote products derived from this software without
26.  *   specific prior written permission.
27.  *
28.  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
29.  * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
30.  * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
31.  * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
32.  * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
33.  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
34.  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
35.  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
36.  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
37.  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
38.  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
39.  *****/
40.
41. #include <stdio.h>
42. #include <stdlib.h>
43. #include <string.h>
44.
45. #include "cs50.h"
46.
47.
48. /*

```

```
49.  * Reads a line of text from standard input and returns the equivalent
50.  * char; if text does not represent a char, user is prompted to retry.
51.  * Leading and trailing whitespace is ignored. If line can't be read,
52.  * returns CHAR_MAX.
53.  */
54.
55. char GetChar(void)
56. {
57.     // try to get a char from user
58.     while (true)
59.     {
60.         // get line of text, returning CHAR_MAX on failure
61.         string line = GetString();
62.         if (line == NULL)
63.             return CHAR_MAX;
64.
65.         // return a char if only a char (possibly with
66.         // leading and/or trailing whitespace) was provided
67.         char c1, c2;
68.         if (sscanf(line, " %c %c", &c1, &c2) == 1)
69.         {
70.             free(line);
71.             return c1;
72.         }
73.         else
74.         {
75.             free(line);
76.             printf("Retry: ");
77.         }
78.     }
79. }
80.
81.
82. /*
83.  * Reads a line of text from standard input and returns the equivalent
84.  * double as precisely as possible; if text does not represent a
85.  * double, user is prompted to retry. Leading and trailing whitespace
86.  * is ignored. For simplicity, overflow and underflow are not detected.
87.  * If line can't be read, returns DBL_MAX.
88.  */
89.
90. double GetDouble(void)
91. {
92.     // try to get a double from user
93.     while (true)
94.     {
95.         // get line of text, returning DBL_MAX on failure
96.         string line = GetString();
```

```
97.         if (line == NULL)
98.             return DBL_MAX;
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. /*
118.  * Reads a line of text from standard input and returns the equivalent
119.  * float as precisely as possible; if text does not represent a float,
120.  * user is prompted to retry. Leading and trailing whitespace is ignored.
121.  * For simplicity, overflow and underflow are not detected. If line can't
122.  * be read, returns FLT_MAX.
123.  */
124.
125. float GetFloat(void)
126. {
127.     // try to get a float from user
128.     while (true)
129.     {
130.         // get line of text, returning FLT_MAX on failure
131.         string line = GetString();
132.         if (line == NULL)
133.             return FLT_MAX;
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. /*
153.  * Reads a line of text from standard input and returns it as an
154.  * int in the range of  $[-2^{31} + 1, 2^{31} - 2]$ , if possible; if text
155.  * does not represent such an int, user is prompted to retry. Leading
156.  * and trailing whitespace is ignored. For simplicity, overflow is not
157.  * detected. If line can't be read, returns INT_MAX.
158.  */
159.
160. int GetInt(void)
161. {
162.     // try to get an int from user
163.     while (true)
164.     {
165.         // get line of text, returning INT_MAX on failure
166.         string line = GetString();
167.         if (line == NULL)
168.             return INT_MAX;
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, " %d %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. /*
188.  * Reads a line of text from standard input and returns an equivalent
189.  * long long in the range  $[-2^{63} + 1, 2^{63} - 2]$ , if possible; if text
190.  * does not represent such a long long, user is prompted to retry.
191.  * Leading and trailing whitespace is ignored. For simplicity, overflow
192.  * is not detected. If line can't be read, returns LLONG_MAX.
```

```
193.  */
194.
195. long long GetLongLong(void)
196. {
197.     // try to get a long long from user
198.     while (true)
199.     {
200.         // get line of text, returning LLONG_MAX on failure
201.         string line = GetString();
202.         if (line == NULL)
203.             return LLONG_MAX;
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. /*
223.  * Reads a line of text from standard input and returns it as a
224.  * string (char*), sans trailing newline character. (Ergo, if
225.  * user inputs only "\n", returns "" not NULL.) Returns NULL
226.  * upon error or no input whatsoever (i.e., just EOF). Leading
227.  * and trailing whitespace is not ignored. Stores string on heap
228.  * (via malloc); memory must be freed by caller to avoid leak.
229.  */
230.
231. string GetString(void)
232. {
233.     // growable buffer for chars
234.     string buffer = NULL;
235.
236.     // capacity of buffer
237.     unsigned int capacity = 0;
238.
239.     // number of chars actually in buffer
240.     unsigned int n = 0;
```

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

```
289.     return minimal;
290. }
```

```

1.  /*****
2.   * CS50 Library 4
3.   * https://manual.cs50.net/Library
4.   *
5.   * Based on Eric Roberts' genlib.c and simpio.c.
6.   *
7.   * Copyright (c) 2012
8.   * Glenn Holloway <holloway@eecs.harvard.edu>
9.   * David J. Malan <malan@harvard.edu>
10.  * All Rights Reserved
11.  *
12.  * BSD 3-Clause License
13.  * http://www.opensource.org/licenses/BSD-3-Clause
14.  *
15.  * Redistribution and use in source and binary forms, with or without
16.  * modification, are permitted provided that the following conditions are
17.  * met:
18.  *
19.  * * Redistributions of source code must retain the above copyright notice,
20.  *   this list of conditions and the following disclaimer.
21.  * * Redistributions in binary form must reproduce the above copyright
22.  *   notice, this list of conditions and the following disclaimer in the
23.  *   documentation and/or other materials provided with the distribution.
24.  * * Neither the name of CS50 nor the names of its contributors may be used
25.  *   to endorse or promote products derived from this software without
26.  *   specific prior written permission.
27.  *
28.  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
29.  * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
30.  * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
31.  * PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
32.  * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
33.  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
34.  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
35.  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
36.  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
37.  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
38.  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
39.  *****/
40.
41. #ifndef _CS50_H
42. #define _CS50_H
43.
44. #include <float.h>
45. #include <limits.h>
46. #include <stdbool.h>
47. #include <stdlib.h>
48.

```



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

```
97. int GetInt(void);
98.
99.
100. /*
101.  * Reads a line of text from standard input and returns an equivalent
102.  * long long in the range  $[-2^{63} + 1, 2^{63} - 2]$ , if possible; if text
103.  * does not represent such a long long, user is prompted to retry.
104.  * Leading and trailing whitespace is ignored. For simplicity, overflow
105.  * is not detected. If line can't be read, returns LLONG_MAX.
106.  */
107.
108. long long GetLongLong(void);
109.
110.
111. /*
112.  * Reads a line of text from standard input and returns it as a
113.  * string (char *), sans trailing newline character. (Ergo, if
114.  * user inputs only "\n", returns "" not NULL.) Returns NULL
115.  * upon error or no input whatsoever (i.e., just EOF). Leading
116.  * and trailing whitespace is not ignored. Stores string on heap
117.  * (via malloc); memory must be freed by caller to avoid leak.
118.  */
119.
120. string GetString(void);
121.
122.
123.
124. #endif
```