

```
1. /*****
2.  * argv1.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Prints command-line arguments, one per line.
8.  *
9.  * Demonstrates use of argv.
10. *****/
11.
12. #include <stdio.h>
13.
14.
15. int
16. main(int argc, char *argv[])
17. {
18.     // print arguments
19.     printf("\n");
20.     for (int i = 0; i < argc; i++)
21.         printf("%s\n", argv[i]);
22.     printf("\n");
23. }
```

```
1. /*****
2.  * bar.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Offers opportunities to play with pointers with GDB.
8.  *****/
9.
10. #include <stdio.h>
11.
12.
13. int foo(int n);
14. void bar(int m);
15.
16. int
17. main(void)
18. {
19.     int a;
20.     char * s = "hello, world";
21.     printf("%s\n", &s[7]);
22.     a = 5;
23.     foo(a);
24.     return 0;
25. }
26.
27. int
28. foo(int n)
29. {
30.     int b;
31.     b = n;
32.     b *= 2;
33.     bar(b);
34.     return b;
35. }
36.
37. void
38. bar(int m)
39. {
40.     printf("Hi, I'm bar!\n");
41. }
```

```
1.  /*****
2.  * compare1.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Tries (and fails) to compare two strings.
8.  *
9.  * Demonstrates strings as pointers to arrays.
10. *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14.
15.
16. int
17. main(void)
18. {
19.     // get line of text
20.     printf("Say something: ");
21.     string s1 = GetString();
22.
23.     // get another line of text
24.     printf("Say something: ");
25.     string s2 = GetString();
26.
27.     // try (and fail) to compare strings
28.     if (s1 == s2)
29.         printf("You typed the same thing!\n");
30.     else
31.         printf("You typed different things!\n");
32. }
```

```
1.  /*****
2.   * compare2.c
3.   *
4.   * Computer Science 50
5.   * David J. Malan
6.   *
7.   * Compares two strings.
8.   *
9.   * Demonstrates strings as pointers to arrays.
10.  *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14. #include <string.h>
15.
16.
17. int
18. main(void)
19. {
20.     // get line of text
21.     printf("Say something: ");
22.     char *s1 = GetString();
23.
24.     // get another line of text
25.     printf("Say something: ");
26.     char *s2 = GetString();
27.
28.     // try to compare strings
29.     if (s1 != NULL && s2 != NULL)
30.     {
31.         if (strcmp(s1, s2) == 0)
32.             printf("You typed the same thing!\n");
33.         else
34.             printf("You typed different things!\n");
35.     }
36. }
```

```
1.  /*****
2.  * copy1.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Tries and fails to copy two strings.
8.  *
9.  * Demonstrates strings as pointers to arrays.
10. *****/
11.
12. #include <cs50.h>
13. #include <ctype.h>
14. #include <stdio.h>
15. #include <stdlib.h>
16. #include <string.h>
17.
18.
19. int
20. main(void)
21. {
22.     // get line of text
23.     printf("Say something: ");
24.     char *s1 = GetString();
25.     if (s1 == NULL)
26.         return 1;
27.
28.     // try (and fail) to copy string
29.     char *s2 = s1;
30.
31.     // change "copy"
32.     printf("Capitalizing copy...\n");
33.     if (strlen(s2) > 0)
34.         s2[0] = toupper(s2[0]);
35.
36.     // print original and "copy"
37.     printf("Original: %s\n", s1);
38.     printf("Copy:     %s\n", s2);
39.
40.     // free memory
41.     free(s1);
42. }
```

```
1.  /*****
2.  * copy2.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Copies a string.
8.  *
9.  * Demonstrates strings as pointers to arrays.
10. *****/
11.
12. #include <cs50.h>
13. #include <ctype.h>
14. #include <stdio.h>
15. #include <stdlib.h>
16. #include <string.h>
17.
18.
19. int
20. main(void)
21. {
22.     // get line of text
23.     printf("Say something: ");
24.     char *s1 = GetString();
25.     if (s1 == NULL)
26.         return 1;
27.
28.     // allocate enough space for copy
29.     char *s2 = malloc((strlen(s1) + 1) * sizeof(char));
30.     if (s2 == NULL)
31.         return 1;
32.
33.     // copy string
34.     int n = strlen(s1);
35.     for (int i = 0; i < n; i++)
36.         s2[i] = s1[i];
37.     s2[n] = '\0';
38.
39.     // change copy
40.     printf("Capitalizing copy...\n");
41.     if (strlen(s2) > 0)
42.         s2[0] = toupper(s2[0]);
43.
44.     // print original and copy
45.     printf("Original: %s\n", s1);
46.     printf("Copy:      %s\n", s2);
47.
48.     // free memory
```

---

```
49.     free(s1);
50.     free(s2);
51. }
```

```
1.  /*****
2.   * CS50 Library 3.0
3.   *
4.   * https://manual.cs50.net/Library
5.   *
6.   * Glenn Holloway <holloway@eecs.harvard.edu>
7.   * David J. Malan <malan@harvard.edu>
8.   *
9.   * Based on Eric Roberts' genlib.c and simpio.c.
10.  *
11.  * Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
12.  * http://creativecommons.org/licenses/by-nc-sa/3.0/
13.  *****/
14.
15. #include <stdio.h>
16. #include <stdlib.h>
17. #include <string.h>
18.
19. #include "cs50.h"
20.
21.
22. /*
23.  * Reads a line of text from standard input and returns the equivalent
24.  * char; if text does not represent a char, user is prompted to retry.
25.  * Leading and trailing whitespace is ignored. If line can't be read,
26.  * returns CHAR_MAX.
27.  */
28.
29. char
30. GetChar(void)
31. {
32.     // try to get a char from user
33.     while (true)
34.     {
35.         // get line of text, returning CHAR_MAX on failure
36.         string line = GetString();
37.         if (line == NULL)
38.             return CHAR_MAX;
39.
40.         // return a char if only a char (possibly with
41.         // leading and/or trailing whitespace) was provided
42.         char c1, c2;
43.         if (sscanf(line, " %c %c", &c1, &c2) == 1)
44.         {
45.             free(line);
46.             return c1;
47.         }
48.         else
```



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

```
97.  * For simplicity, overflow and underflow are not detected.  If line can't
98.  * be read, returns FLT_MAX.
99.  */
100.
101. float
102. GetFloat(void)
103. {
104.     // try to get a float from user
105.     while (true)
106.     {
107.         // get line of text, returning FLT_MAX on failure
108.         string line = GetString();
109.         if (line == NULL)
110.             return FLT_MAX;
111.
112.         // return a float if only a float (possibly with
113.         // leading and/or trailing whitespace) was provided
114.         char c; float f;
115.         if (sscanf(line, " %f %c", &f, &c) == 1)
116.         {
117.             free(line);
118.             return f;
119.         }
120.         else
121.         {
122.             free(line);
123.             printf("Retry: ");
124.         }
125.     }
126. }
127.
128.
129. /*
130.  * Reads a line of text from standard input and returns it as an
131.  * int in the range of  $[-2^{31} + 1, 2^{31} - 2]$ , if possible; if text
132.  * does not represent such an int, user is prompted to retry.  Leading
133.  * and trailing whitespace is ignored.  For simplicity, overflow is not
134.  * detected.  If line can't be read, returns INT_MAX.
135.  */
136.
137. int
138. GetInt(void)
139. {
140.     // try to get an int from user
141.     while (true)
142.     {
143.         // get line of text, returning INT_MAX on failure
144.         string line = GetString();
```

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

```
193.     {
194.         free(line);
195.         printf("Retry: ");
196.     }
197. }
198. }
199.
200.
201. /*
202.  * Reads a line of text from standard input and returns it as a
203.  * string (char *), sans trailing newline character. (Ergo, if
204.  * user inputs only "\n", returns "" not NULL.) Returns NULL
205.  * upon error or no input whatsoever (i.e., just EOF). Leading
206.  * and trailing whitespace is not ignored. Stores string on heap
207.  * (via malloc); memory must be freed by caller to avoid leak.
208.  */
209.
210. string
211. GetString(void)
212. {
213.     // growable buffer for chars
214.     string buffer = NULL;
215.
216.     // capacity of buffer
217.     unsigned int capacity = 0;
218.
219.     // number of chars actually in buffer
220.     unsigned int n = 0;
221.
222.     // character read or EOF
223.     int c;
224.
225.     // iteratively get chars from standard input
226.     while ((c = fgetc(stdin)) != '\n' && c != EOF)
227.     {
228.         // grow buffer if necessary
229.         if (n + 1 > capacity)
230.         {
231.             // determine new capacity: start at 32 then double
232.             if (capacity == 0)
233.                 capacity = 32;
234.             else if (capacity <= (UINT_MAX / 2))
235.                 capacity *= 2;
236.             else
237.             {
238.                 free(buffer);
239.                 return NULL;
240.             }

```

```
241.
242.     // extend buffer's capacity
243.     string temp = realloc(buffer, capacity * sizeof(char));
244.     if (temp == NULL)
245.     {
246.         free(buffer);
247.         return NULL;
248.     }
249.     buffer = temp;
250. }
251.
252.     // append current character to buffer
253.     buffer[n++] = c;
254. }
255.
256. // return NULL if user provided no input
257. if (n == 0 && c == EOF)
258.     return NULL;
259.
260. // minimize buffer
261. string minimal = malloc((n + 1) * sizeof(char));
262. strncpy(minimal, buffer, n);
263. free(buffer);
264.
265. // terminate string
266. minimal[n] = '\0';
267.
268. // return string
269. return minimal;
270. }
```

```
1. /*****
2.  * pointers.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Prints a given string one character per line.
8.  *
9.  * Demonstrates pointer arithmetic.
10. *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14. #include <stdlib.h>
15. #include <string.h>
16.
17.
18. int
19. main(void)
20. {
21.     // get line of text
22.     char *s = GetString();
23.     if (s == NULL)
24.         return 1;
25.
26.     // print string, one character per line
27.     for (int i = 0, n = strlen(s); i < n; i++)
28.         printf("%c\n", *(s+i));
29.
30.     // free string
31.     free(s);
32. }
```

```
1. /*****
2.  * scanf1.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Reads a number from the user into an int.
8.  *
9.  * Demonstrates scanf and address-of operator.
10. *****/
11.
12. #include <stdio.h>
13.
14.
15. int
16. main(void)
17. {
18.     int x;
19.     printf("Number please: ");
20.     scanf("%d", &x);
21.     printf("Thanks for the %d!\n", x);
22. }
```

```
1.  /*****
2.   * scanf2.c
3.   *
4.   * Computer Science 50
5.   * David J. Malan
6.   *
7.   * Reads a string from the user into memory it shouldn't.
8.   *
9.   * Demonstrates possible attack!
10.  *****/
11.
12.  #include <stdio.h>
13.
14.
15.  int
16.  main(void)
17.  {
18.      char *buffer;
19.      printf("String please: ");
20.      scanf("%s", buffer);
21.      printf("Thanks for the \"%s\"!\n", buffer);
22.  }
```



```
1. /*****
2.  * scanf3.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Reads a string from the user into an array (dangerously).
8.  *
9.  * Demonstrates potential buffer overflow!
10. *****/
11.
12. #include <stdio.h>
13.
14.
15. int
16. main(void)
17. {
18.     char buffer[16];
19.     printf("String please: ");
20.     scanf("%s", buffer);
21.     printf("Thanks for the \"%s\"!\n", buffer);
22. }
```

```
1. /*****
2.  * structs.h
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Defines a student for structs{1,2}.c.
8.  *****/
9.
10.
11. // structure representing a student
12. typedef struct
13. {
14.     int id;
15.     char *name;
16.     char *house;
17. }
18. student;
```

```
1. /*****
2.  * structs1.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Demonstrates use of structs.
8.  *****/
9.
10. #include <cs50.h>
11. #include <stdio.h>
12. #include <stdlib.h>
13. #include <string.h>
14.
15. #include "structs.h"
16.
17.
18. // class size
19. #define STUDENTS 3
20.
21.
22. int
23. main(void)
24. {
25.     // declare class
26.     student class[STUDENTS];
27.
28.     // populate class with user's input
29.     for (int i = 0; i < STUDENTS; i++)
30.     {
31.         printf("Student's ID: ");
32.         class[i].id = GetInt();
33.
34.         printf("Student's name: ");
35.         class[i].name = GetString();
36.
37.         printf("Student's house: ");
38.         class[i].house = GetString();
39.         printf("\n");
40.     }
41.
42.     // now print anyone in Mather
43.     for (int i = 0; i < STUDENTS; i++)
44.         if (strcmp(class[i].house, "Mather") == 0)
45.             printf("%s is in Mather!\n\n", class[i].name);
46.
47.     // free memory
48.     for (int i = 0; i < STUDENTS; i++)
```

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

```
1.  /*****
2.  * swap.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Swaps two variables' values.
8.  *
9.  * Demonstrates passing by reference.
10. *****/
11.
12. #include <stdio.h>
13.
14.
15. // function prototype
16. void swap(int *a, int *b);
17.
18.
19. int
20. main(void)
21. {
22.     int x = 1;
23.     int y = 2;
24.
25.     printf("x is %d\n", x);
26.     printf("y is %d\n", y);
27.     printf("Swapping...\n");
28.     swap(&x, &y);
29.     printf("Swapped!\n");
30.     printf("x is %d\n", x);
31.     printf("y is %d\n", y);
32. }
33.
34.
35. /*
36. * Swap arguments' values.
37. */
38.
39. void
40. swap(int *a, int *b)
41. {
42.     int tmp = *a;
43.     *a = *b;
44.     *b = tmp;
45. }
```