

```
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. #ifndef _CS50_H
16. #define _CS50_H
17.
18. #include <float.h>
19. #include <limits.h>
20. #include <stdbool.h>
21. #include <stdlib.h>
22.
23.
24. /*
25.  * Our own data type for string variables.
26.  */
27.
28. typedef char *string;
29.
30.
31. /*
32.  * Reads a line of text from standard input and returns the equivalent
33.  * char; if text does not represent a char, user is prompted to retry.
34.  * Leading and trailing whitespace is ignored. If line can't be read,
35.  * returns CHAR_MAX.
36.  */
37.
38. char
39. GetChar(void);
40.
41.
42. /*
43.  * Reads a line of text from standard input and returns the equivalent
44.  * double as precisely as possible; if text does not represent a
45.  * double, user is prompted to retry. Leading and trailing whitespace
46.  * is ignored. For simplicity, overflow and underflow are not detected.
47.  * If line can't be read, returns DBL_MAX.
48.  */
```

```
49.
50. double
51. GetDouble(void);
52.
53.
54. /*
55.  * Reads a line of text from standard input and returns the equivalent
56.  * float as precisely as possible; if text does not represent a float,
57.  * user is prompted to retry. Leading and trailing whitespace is ignored.
58.  * For simplicity, overflow and underflow are not detected. If line can't
59.  * be read, returns FLT_MAX.
60.  */
61.
62. float
63. GetFloat(void);
64.
65.
66. /*
67.  * Reads a line of text from standard input and returns it as an
68.  * int in the range of  $[-2^{31} + 1, 2^{31} - 2]$ , if possible; if text
69.  * does not represent such an int, user is prompted to retry. Leading
70.  * and trailing whitespace is ignored. For simplicity, overflow is not
71.  * detected. If line can't be read, returns INT_MAX.
72.  */
73.
74. int
75. GetInt(void);
76.
77.
78. /*
79.  * Reads a line of text from standard input and returns an equivalent
80.  * long long in the range  $[-2^{63} + 1, 2^{63} - 2]$ , if possible; if text
81.  * does not represent such a long long, user is prompted to retry.
82.  * Leading and trailing whitespace is ignored. For simplicity, overflow
83.  * is not detected. If line can't be read, returns LLONG_MAX.
84.  */
85.
86. long long
87. GetLongLong(void);
88.
89.
90. /*
91.  * Reads a line of text from standard input and returns it as a
92.  * string (char *), sans trailing newline character. (Ergo, if
93.  * user inputs only "\n", returns "" not NULL.) Returns NULL
94.  * upon error or no input whatsoever (i.e., just EOF). Leading
95.  * and trailing whitespace is not ignored. Stores string on heap
96.  * (via malloc); memory must be freed by caller to avoid leak.
```

```
97.  */
98.
99.  string GetString(void);
100.
101.
102.
103. #endif
```



```
1.  /*****
2.   *  sigma1.c
3.   *
4.   *  Computer Science 50
5.   *  David J. Malan
6.   *
7.   *  Adds the numbers 1 through n.
8.   *
9.   *  Demonstrates iteration.
10.  *****/
11.
12.  #include <cs50.h>
13.  #include <stdio.h>
14.
15.
16.  // prototype
17.  int sigma(int);
18.
19.
20.  int
21.  main(void)
22.  {
23.      // ask user for a positive int
24.      int n;
25.      do
26.      {
27.          printf("Positive integer please: ");
28.          n = GetInt();
29.      }
30.      while (n < 1);
31.
32.      // compute sum of 1 through n
33.      int answer = sigma(n);
34.
35.      // report answer
36.      printf("%d\n", answer);
37.  }
38.
39.
40.  /*
41.   *  Returns sum of 1 through m; returns 0 if m is not positive.
42.   */
43.
44.  int
45.  sigma(int m)
46.  {
47.      // avoid risk of infinite loop
48.      if (m < 1)
```

```
49.     return 0;
50.
51.     // return sum of 1 through m
52.     int sum = 0;
53.     for (int i = 1; i <= m; i++)
54.         sum += i;
55.     return sum;
56. }
```

```
1.  /*****
2.   * sigma2.c
3.   *
4.   * Computer Science 50
5.   * David J. Malan
6.   *
7.   * Adds the numbers 1 through n.
8.   *
9.   * Demonstrates recursion.
10.  *****/
11.
12.  #include <cs50.h>
13.  #include <stdio.h>
14.
15.
16.  // prototype
17.  int sigma(int);
18.
19.
20.  int
21.  main(void)
22.  {
23.      // ask user for a positive int
24.      int n;
25.      do
26.      {
27.          printf("Positive integer please: ");
28.          n = GetInt();
29.      }
30.      while (n < 1);
31.
32.      // compute sum of 1 through n
33.      int answer = sigma(n);
34.
35.      // report answer
36.      printf("%d\n", answer);
37.  }
38.
39.
40.  /*
41.   * Returns sum of 1 through m; returns 0 if m is not positive.
42.   */
43.
44.  int
45.  sigma(int m)
46.  {
47.      // base case
48.      if (m <= 0)
```

```
49.     return 0;
50.
51.     // recursive case
52.     else
53.         return (m + sigma(m-1));
54. }
```