Hai.java 1/1

lectures/2/src/

```
1: class Hai
2: {
3:     public static void main(String [] args)
4:     {
5:         System.out.println("O hai, world!");
6:     }
7: }
```

argv1.c 1/1

lectures/2/src/

```
 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:     int i;
19:
20:     // print arguments
21:     printf("\n");
22:     for (i = 0; i < argc; i++)
23:         printf("%s\n", argv[i]);
24:     printf("\n");
25: }
```

argv2.c 1/1
lectures/2/src/

```
 1: /****************************************************************************
 2:  * argv2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Prints command-line arguments, one character per line.
 8:  *
 9:  * Demonstrates argv as a two-dimensional array.
10:  ****************************************************************************/
11:
12: #include <stdio.h>
13: #include <string.h>
14:
15:
16: int
17: main(int argc, char *argv[])
18: {
19:     int i, j, n;
20:
21:     // print arguments
22:     printf("\n");
23:     for (i = 0; i < argc; i++)
24:     {
25:         for (j = 0, n = strlen(argv[i]); j < n; j++)
26:             printf("%c\n", argv[i][j]);
27:         printf("\n");
28:     }
29: }
```

array.c 1/1
lectures/2/src/

```
 1: /****************************************************************************
 2:  * array.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Computes a student's average across 3 quizzes (without dropping lowest).
 8:  *
 9:  * Demonstrates use of an array, a constant, a multiline string, and
10:  * rounding.
11:  ****************************************************************************/
12:
13: #include <cs50.h>
14: #include <stdio.h>
15:
16: /* number of quizzes per term */
17: #define QUIZZES 3
18:
19:
20: int
21: main(int argc, char * argv[])
22: {
23:     float grades[QUIZZES];
24:     int average, i, sum;
25:
26:     /* ask user for grades */
27:     printf("\nWhat were your quiz scores?\n\n");
28:     for (i = 0; i < QUIZZES; i++)
29:     {
30:         printf("Quiz #%d of %d: ", i+1, QUIZZES);
31:         grades[i] = GetFloat();
32:     }
33:
34:     /* compute average */
35:     sum = 0;
36:     for (i = 0; i < QUIZZES; i++)
37:         sum += grades[i];
38:     average = (int) (sum / (float) QUIZZES + 0.5);
39:
40:     /* report average */
41:     printf("\nWithout dropping your lowest score, "
42:            "your average is: %d\n\n", average);
43: }
```

```
 1: /***************************************************************************
 2:  * array1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Computes a student's average across 2 quizzes.
 8:  *
 9:  * Demonstrates use of an array, a constant, and rounding.
10:  ***************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14:
15:
16: // number of quizzes per term
17: #define QUIZZES 2
18:
19:
20: int
21: main(int argc, char *argv[])
22: {
23:     float grades[QUIZZES], sum;
24:     int average, i;
25:
26:     // ask user for grades
27:     printf("\nWhat were your quiz scores?\n\n");
28:     for (i = 0; i < QUIZZES; i++)
29:     {
30:         printf("Quiz #%d of %d: ", i+1, QUIZZES);
31:         grades[i] = GetFloat();
32:     }
33:
34:     // compute average
35:     sum = 0;
36:     for (i = 0; i < QUIZZES; i++)
37:         sum += grades[i];
38:     average = (int) (sum / QUIZZES + 0.5);
39:
40:     // report average
41:     printf("\nYour average is: %d\n\n", average);
42: }
```



```
 1: /***************************************************************************
 2:  * array2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Computes a student's average across 2 quizzes.
 8:  *
 9:  * Demonstrates C's math library.
10:  ***************************************************************************/
11:
12: #include <cs50.h>
13: #include <math.h>
14: #include <stdio.h>
15:
16:
17: // number of quizzes per term
18: #define QUIZZES 2
19:
20:
21: int
22: main(int argc, char *argv[])
23: {
24:     float grades[QUIZZES], sum;
25:     int average, i;
26:
27:     // ask user for grades
28:     printf("\nWhat were your quiz scores?\n\n");
29:     for (i = 0; i < QUIZZES; i++)
30:     {
31:         printf("Quiz #%d of %d: ", i+1, QUIZZES);
32:         grades[i] = GetFloat();
33:     }
34:
35:     // compute average
36:     sum = 0;
37:     for (i = 0; i < QUIZZES; i++)
38:         sum += grades[i];
39:     average = (int) round(sum / QUIZZES);
40:
41:     // report average
42:     printf("\nYour average is: %d\n\n", average);
43: }
```

ascii1.c 1/1
lectures/2/src/

```
 1: /**************************************************************************
 2:  * ascii1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Displays the mapping between alphabetical ASCII characters and
 8:  * their decimal equivalents using one column.
 9:  *
10:  * Demonstrates casting from int to char.
11:  **************************************************************************/
12:
13: #include <stdio.h>
14:
15:
16: int
17: main(int argc, char *argv[])
18: {
19:     int i;
20:
21:     // display mapping for uppercase letters
22:     for (i = 65; i < 65 + 26; i++)
23:         printf("%c: %d\n", (char) i, i);
24:
25:     // separate uppercase from lowercase
26:     printf("\n");
27:
28:     // display mapping for lowercase letters
29:     for (i = 97; i < 97 + 26; i++)
30:         printf("%c: %d\n", (char) i, i);
31: }
32:
```

ascii2.c 1/1
lectures/2/src/

```
 1: /**************************************************************************
 2:  * ascii2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Displays the mapping between alphabetical ASCII characters and
 8:  * their decimal equivalents using two columns.
 9:  *
10:  * Demonstrates specification of width in format string.
11:  **************************************************************************/
12:
13: #include <stdio.h>
14:
15:
16: int
17: main(int argc, char *argv[])
18: {
19:     int i;
20:
21:     // display mapping for uppercase letters
22:     for (i = 65; i < 65 + 26; i++)
23:         printf("%c  %d    %3d  %c\n", (char) i, i, i + 32, (char) (i + 32));
24: }
25:
```

ascii3.c 1/1
lectures/2/src/

```
 1: /****************************************************************************
 2:  * ascii3.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Displays the mapping between alphabetical ASCII characters and
 8:  * their decimal equivalents.
 9:  *
10:  * Demonstrates iteration with a char.
11:  ***************************************************************************/
12:
13: #include <stdio.h>
14:
15:
16: int
17: main(int argc, char *argv[])
18: {
19:     char c;
20:
21:     // display mapping for uppercase letters
22:     for (c = 'A'; c <= 'Z'; c = (char) ((int) c + 1))
23:         printf("%c: %d\n", c, (int) c);
24: }
```

battleship.c 1/1
lectures/2/src/

```
 1: /****************************************************************************
 2:  * battleship.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Prints a Battleship board.
 8:  *
 9:  * Demonstrates nested loop.
10:  ***************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: int
16: main(int argc, char *argv[])
17: {
18:     int i, j;
19:
20:     // print top row of numbers
21:     printf("\n   ");
22:     for (i = 1; i <= 10; i++)
23:         printf("%d  ", i);
24:     printf("\n");
25:
26:     // print rows of holes, with letters in leftmost column
27:     for (i = 0; i < 10; i++)
28:     {
29:         printf("%c  ", 'A' + i);
30:         for (j = 1; j <= 10; j++)
31:             printf("o  ");
32:         printf("\n");
33:     }
34:     printf("\n");
35: }
```

beer1.c 1/1
lectures/2/src/

```
 1: /****************************************************************************
 2:  * beer1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Sings "99 Bottles of Beer on the Wall."
 8:  *
 9:  * Demonstrates a for loop (and an opportunity for hierarchical
10:  * decomposition).
11:  ****************************************************************************/
12:
13: #include <cs50.h>
14: #include <stdio.h>
15:
16:
17: int
18: main(int argc, char *argv[])
19: {
20:     int i, n;
21:
22:     // ask user for number
23:     printf("How many bottles will there be? ");
24:     n = GetInt();
25:
26:     // exit upon invalid input
27:     if (n < 1)
28:     {
29:         printf("Sorry, that makes no sense.\n");
30:         return 1;
31:     }
32:
33:     // sing the annoying song
34:     printf("\n");
35:     for (i = n; i > 0; i--)
36:     {
37:         printf("%d bottle(s) of beer on the wall,\n", i);
38:         printf("%d bottle(s) of beer,\n", i);
39:         printf("Take one down, pass it around,\n");
40:         printf("%d bottle(s) of beer on the wall.\n\n", i - 1);
41:     }
42:
43:     // exit when song is over
44:     printf("Wow, that's annoying.\n");
45:     return 0;
46: }
```

beer2.c 1/1
lectures/2/src/

```
 1: /****************************************************************************
 2:  * beer2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Sings "99 Bottles of Beer on the Wall."
 8:  *
 9:  * Demonstrates a while loop (and an opportunity for hierarchical
10:  * decomposition).
11:  ****************************************************************************/
12:
13: #include <cs50.h>
14: #include <stdio.h>
15:
16:
17: int
18: main(int argc, char *argv[])
19: {
20:     int n;
21:
22:     // ask user for number
23:     printf("How many bottles will there be? ");
24:     n = GetInt();
25:
26:     // exit upon invalid input
27:     if (n < 1)
28:     {
29:         printf("Sorry, that makes no sense.\n");
30:         return 1;
31:     }
32:
33:     // sing the annoying song
34:     printf("\n");
35:     while (n > 0)
36:     {
37:         printf("%d bottle(s) of beer on the wall,\n", n);
38:         printf("%d bottle(s) of beer,\n", n);
39:         printf("Take one down, pass it around,\n");
40:         printf("%d bottle(s) of beer on the wall.\n\n", n - 1);
41:         n--;
42:     }
43:
44:     // exit when song is over
45:     printf("Wow, that's annoying.\n");
46:     return 0;
47: }
```

```
 1: /**************************************************************************
 2:  * beer3.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Sings "99 Bottles of Beer on the Wall."
 8:  *
 9:  * Demonstrates a condition within a for loop.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14:
15:
16: int
17: main(int argc, char *argv[])
18: {
19:     int i, n;
20:     string s1, s2;
21:
22:     // ask user for number
23:     printf("How many bottles will there be? ");
24:     n = GetInt();
25:
26:     // exit upon invalid input
27:     if (n < 1)
28:     {
29:         printf("Sorry, that makes no sense.\n");
30:         return 1;
31:     }
32:
33:     // sing the annoying song
34:     printf("\n");
35:     for (i = n; i > 0; i--)
36:     {
37:         // use proper grammar
38:         s1 = (i == 1) ? "bottle" : "bottles";
39:         s2 = (i == 2) ? "bottle" : "bottles";
40:
41:         // sing verses
42:         printf("%d %s of beer on the wall,\n", i, s1);
43:         printf("%d %s of beer,\n", i, s1);
44:         printf("Take one down, pass it around,\n");
45:         printf("%d %s of beer on the wall.\n\n", i - 1, s2);
46:     }
47:
48:     // exit when song is over
49:     printf("Wow, that's annoying.\n");
50:     return 0;
51: }
```



```
 1: /**************************************************************************
 2:  * beer4.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Sings "99 Bottles of Beer on the Wall."
 8:  *
 9:  * Demonstrates hierarchical decomposition and parameter passing.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14:
15:
16: // function prototype
17: void chorus(int);
18:
19:
20: int
21: main(int argc, char *argv[])
22: {
23:     int n;
24:
25:     // ask user for number
26:     printf("How many bottles will there be? ");
27:     n = GetInt();
28:
29:     // exit upon invalid input
30:     if (n < 1)
31:     {
32:         printf("Sorry, that makes no sense.\n");
33:         return 1;
34:     }
35:
36:     // sing the annoying song
37:     printf("\n");
38:     while (n)
39:         chorus(n--);
40:
41:     // exit when song is over
42:     printf("Wow, that's annoying.\n");
43:     return 0;
44: }
45:
46:
47: /*
48:  * void
49:  * chorus(int bottles)
50:  *
51:  * Sings about specified number of bottles.
52:  */
53:
54: void
55: chorus(int b)
56: {
57:     string s1, s2;
58:
59:     // use proper grammar
60:     s1 = (b == 1) ? "bottle" : "bottles";
61:     s2 = (b == 2) ? "bottle" : "bottles";
62:
63:     // sing verses
64:     printf("%d %s of beer on the wall,\n", b, s1);
```

```
65:     printf("%d %s of beer,\n", b, s1);
66:     printf("Take one down, pass it around,\n");
67:     printf("%d %s of beer on the wall.\n\n", b - 1, s2);
68: }
```



```
 1: /**************************************************************************
 2:  * buggy1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Should print 10 asterisks but doesn't!
 8:  * Can you find the bug?
 9:  **************************************************************************/
10:
11: #include <stdio.h>
12:
13: int
14: main(int argc, char *argv[])
15: {
16:     int i;
17:
18:     for (i = 0; i <= 10; i++)
19:         printf("*");
20: }
```

buggy2.c 1/1
lectures/2/src/

```
 1: /**************************************************************************
 2:  * buggy2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Should print 10 asterisks, one per line, but doesn't!
 8:  * Can you find the bug?
 9:  **************************************************************************/
10:
11: #include <stdio.h>
12:
13: int
14: main(int argc, char *argv[])
15: {
16:     int i;
17:
18:     for (i = 0; i <= 10; i++)
19:         printf("*");
20:         printf("\n");
21: }
```

buggy3.c 1/1
lectures/2/src/

```
 1: /**************************************************************************
 2:  * buggy3.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Should swap two variables' values, but doesn't!
 8:  * Can you find the bug?
 9:  **************************************************************************/
10:
11: #include <stdio.h>
12:
13:
14: // function prototype
15: void swap(int, int);
16:
17:
18: int
19: main(int argc, char *argv[])
20: {
21:     int x = 1;
22:     int y = 2;
23:
24:     printf("x is %d\n", x);
25:     printf("y is %d\n", y);
26:     printf("Swapping...\n");
27:     swap(x, y);
28:     printf("Swapped!\n");
29:     printf("x is %d\n", x);
30:     printf("y is %d\n", y);
31: }
32:
33:
34: /*
35:  * void
36:  * swap(int a, int b)
37:  *
38:  * Swap arguments' values.
39:  */
40:
41: void
42: swap(int a, int b)
43: {
44:     int tmp;
45:
46:     tmp = a;
47:     a = b;
48:     b = tmp;
49: }
```

```
 1: /****************************************************************************
 2:  * buggy4.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Should increment a variable, but doesn't!
 8:  * Can you find the bug?
 9:  ****************************************************************************/
10:
11: #include <stdio.h>
12:
13:
14: // function prototype
15: void increment();
16:
17:
18: int
19: main(int argc, char *argv[])
20: {
21:     int x = 1;
22:     printf("x is now %d\n", x);
23:     printf("Incrementing...\n");
24:     increment();
25:     printf("Incremented!\n");
26:     printf("x is now %d\n", x);
27: }
28:
29:
30: /*
31:  * void
32:  * increment()
33:  *
34:  * Tries to increment x.
35:  */
36:
37: void
38: increment()
39: {
40:     x++;
41: }
```



```
 1: /****************************************************************************
 2:  * buggy5.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Should increment a variable, but doesn't!
 8:  * Can you find the bug?
 9:  ****************************************************************************/
10:
11: #include <stdio.h>
12:
13:
14: // global variable
15: int x;
16:
17: // function prototype
18: void increment();
19:
20:
21: int
22: main(int argc, char *argv[])
23: {
24:     printf("x is now %d\n", x);
25:     printf("Initializing...\n");
26:     x = 1;
27:     printf("Initialized!\n");
28:     printf("x is now %d\n", x);
29:     printf("Incrementing...\n");
30:     increment();
31:     printf("Incremented!\n");
32:     printf("x is now %d\n", x);
33: }
34:
35:
36: /*
37:  * void
38:  * increment()
39:  *
40:  * Increments x.
41:  */
42:
43: void
44: increment()
45: {
46:     int x = 10;
47:     x++;
48: }
```

buggy6.c 1/1
lectures/2/src/

```
 1: /**************************************************************************
 2:  * buggy6.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Asks student for their grades but prints too many!
 8:  * Can you find the bug?
 9:  *
10:  * Demonstrates accidental use of a "magic number."
11:  **************************************************************************/
12:
13: #include <cs50.h>
14: #include <stdio.h>
15:
16:
17: // number of quizzes per term
18: #define QUIZZES 2
19:
20:
21: int
22: main(int argc, char *argv[])
23: {
24:     float grades[QUIZZES];
25:     int i;
26:
27:     // ask user for scores
28:     printf("\nWhat were your quiz scores?\n\n");
29:     for (i = 0; i < QUIZZES; i++)
30:     {
31:         printf("Quiz #%d of %d: ", i+1, QUIZZES);
32:         grades[i] = GetFloat();
33:     }
34:
35:     // print scores
36:     for (i = 0; i < 3; i++)
37:         printf("%.2f\n", grades[i]);
38: }
```

capitalize.c 1/1
lectures/2/src/

```
 1: /**************************************************************************
 2:  * capitalize.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Capitalizes a given string.
 8:  *
 9:  * Demonstrates casting and iteration over strings as arrays of chars.
10:  **************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <string.h>
15:
16:
17: int
18: main(int argc, char *argv[])
19: {
20:     int i, n;
21:     string s;
22:
23:     // get line of text
24:     s = GetString();
25:
26:     // capitalize text
27:     for (i = 0, n = strlen(s); i < n; i++)
28:     {
29:         if (s[i] >= 'a' && s[i] <= 'z')
30:             printf("%c", s[i] - ('a' - 'A'));
31:         else
32:             printf("%c", s[i]);
33:     }
34:     printf("\n");
35: }
```

```
/**************************************************************************
 * cs50.c
 *
 * version 1.1.2
 *
 * Computer Science 50
 * Glenn Holloway
 * David J. Malan
 *
 * Definitions for CS 50's library.
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * The latest version of this file can be found at
 * http://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


/*
 * char
 * GetChar()
 *
 * 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()
{
    // 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: ");
        }
    }
}


/*
 * double
 * GetDouble()
 *
 * 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()
{
    // 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: ");
        }
    }
}


/*
 * float
 * GetFloat()
 *
 * 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()
{
```

```
129:     // try to get a float from user
130:     while (true)
131:     {
132:         // get line of text, returning FLT_MAX on failure
133:         string line = GetString();
134:         if (line == NULL)
135:             return FLT_MAX;
136:
137:         // return a float if only a float (possibly with
138:         // leading and/or trailing whitespace) was provided
139:         char c; float f;
140:         if (sscanf(line, " %f %c", &f, &c) == 1)
141:         {
142:             free(line);
143:             return f;
144:         }
145:         else
146:         {
147:             free(line);
148:             printf("Retry: ");
149:         }
150:     }
151: }
152:
153:
154: /*
155:  * int
156:  * GetInt()
157:  *
158:  * Reads a line of text from standard input and returns it as an
159:  * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
160:  * does not represent such an int, user is prompted to retry.  Leading
161:  * and trailing whitespace is ignored.  For simplicity, overflow is not
162:  * detected.  If line can't be read, returns INT_MAX.
163:  */
164:
165: int
166: GetInt()
167: {
168:     // try to get an int from user
169:     while (true)
170:     {
171:         // get line of text, returning INT_MAX on failure
172:         string line = GetString();
173:         if (line == NULL)
174:             return INT_MAX;
175:
176:         // return an int if only an int (possibly with
177:         // leading and/or trailing whitespace) was provided
178:         int n; char c;
179:         if (sscanf(line, " %d %c", &n, &c) == 1)
180:         {
181:             free(line);
182:             return n;
183:         }
184:         else
185:         {
186:             free(line);
187:             printf("Retry: ");
188:         }
189:     }
190: }
191:
192:
```



```
193: /*
194:  * long long
195:  * GetLongLong()
196:  *
197:  * Reads a line of text from standard input and returns an equivalent
198:  * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
199:  * does not represent such a long long, user is prompted to retry.
200:  * Leading and trailing whitespace is ignored.  For simplicity, overflow
201:  * is not detected.  If line can't be read, returns LLONG_MAX.
202:  */
203:
204: long long
205: GetLongLong()
206: {
207:     // try to get a long long from user
208:     while (true)
209:     {
210:         // get line of text, returning LLONG_MAX on failure
211:         string line = GetString();
212:         if (line == NULL)
213:             return LLONG_MAX;
214:
215:         // return a long long if only a long long (possibly with
216:         // leading and/or trailing whitespace) was provided
217:         long long n; char c;
218:         if (sscanf(line, " %lld %c", &n, &c) == 1)
219:         {
220:             free(line);
221:             return n;
222:         }
223:         else
224:         {
225:             free(line);
226:             printf("Retry: ");
227:         }
228:     }
229: }
230:
231:
232: /*
233:  * string
234:  * GetString()
235:  *
236:  * Reads a line of text from standard input and returns it as a string,
237:  * sans trailing newline character.  (Ergo, if user inputs only "\n",
238:  * returns "" not NULL.)  Leading and trailing whitespace is not ignored.
239:  * Returns NULL upon error or no input whatsoever (i.e., just EOF).
240:  */
241:
242: string
243: GetString()
244: {
245:     // growable buffer for chars
246:     string buffer = NULL;
247:
248:     // capacity of buffer
249:     unsigned int capacity = 0;
250:
251:     // number of chars actually in buffer
252:     unsigned int n = 0;
253:
254:     // character read or EOF
255:     int c;
256:
```

```
257:     // iteratively get chars from standard input
258:     while ((c = fgetc(stdin)) != '\n' && c != EOF)
259:     {
260:         // grow buffer if necessary
261:         if (n + 1 > capacity)
262:         {
263:             // determine new capacity: start at CAPACITY then double
264:             if (capacity == 0)
265:                 capacity = CAPACITY;
266:             else if (capacity <= (UINT_MAX / 2))
267:                 capacity += 2;
268:             else
269:             {
270:                 free(buffer);
271:                 return NULL;
272:             }
273:
274:             // extend buffer's capacity
275:             string temp = realloc(buffer, capacity * sizeof(char));
276:             if (temp == NULL)
277:             {
278:                 free(buffer);
279:                 return NULL;
280:             }
281:             buffer = temp;
282:         }
283:
284:         // append current character to buffer
285:         buffer[n++] = c;
286:     }
287:
288:     // return NULL if user provided no input
289:     if (n == 0 && c == EOF)
290:         return NULL;
291:
292:     // minimize buffer
293:     string minimal = malloc((n + 1) * sizeof(char));
294:     strncpy(minimal, buffer, n);
295:     free(buffer);
296:
297:     // terminate string
298:     minimal[n] = '\0';
299:
300:     // return string
301:     return minimal;
302: }
303:
```



```
  1: /**************************************************************************
  2:  * cs50.h
  3:  *
  4:  * version 1.1.2
  5:  *
  6:  * Computer Science 50
  7:  * Glenn Holloway
  8:  * David J. Malan
  9:  *
 10:  * Declarations for CS 50's library.
 11:  * Based on Eric Roberts' genlib.h and simpio.h.
 12:  *
 13:  * The latest version of this file can be found at
 14:  * http://cs50.net/pub/releases/cs50/cs50.h.
 15:  *
 16:  * To compile as a static library on your own system:
 17:  * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
 18:  * % ar rcs libcs50.a cs50.o
 19:  * % rm -f cs50.o
 20:  * % cp cs50.h /usr/local/include
 21:  * % cp libcs50.a /usr/local/lib
 22:  **************************************************************************/
 23:
 24: #ifndef _CS50_H
 25: #define _CS50_H
 26:
 27: #include <float.h>
 28: #include <limits.h>
 29:
 30:
 31: /*
 32:  * bool
 33:  *
 34:  * Borrow the standard library's data type for Boolean variables whose
 35:  * values must be (true|false).
 36:  */
 37:
 38: #include <stdbool.h>
 39:
 40:
 41: /*
 42:  * string
 43:  *
 44:  * Our own data type for string variables.
 45:  */
 46:
 47: typedef char *string;
 48:
 49:
 50: /*
 51:  * char
 52:  * GetChar()
 53:  *
 54:  * Reads a line of text from standard input and returns the equivalent
 55:  * char; if text does not represent a char, user is prompted to retry.
 56:  * Leading and trailing whitespace is ignored.  If line can't be read,
 57:  * returns CHAR_MAX.
 58:  */
 59:
 60: char GetChar();
 61:
 62:
 63: /*
 64:  * double
```

```
 * GetDouble()
 *
 * 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();


/*
 * float
 * GetFloat()
 *
 * 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();


/*
 * int
 * GetInt()
 *
 * 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();


/*
 * long long
 * GetLongLong()
 *
 * 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();


/*
 * string
 * GetString()
 *
 * 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();


#endif

```

global.c 1/1

lectures/2/src/

```
 1: /***************************************************************************
 2:  * global.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Increments variables.
 8:  *
 9:  * Demonstrates use of global variable and issue of scope.
10:  ***************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: // global variable
16: int x;
17:
18: // function prototype
19: void increment();
20:
21:
22: int
23: main(int argc, char *argv[])
24: {
25:     printf("x is now %d\n", x);
26:     printf("Initializing...\n");
27:     x = 1;
28:     printf("Initialized!\n");
29:     printf("x is now %d\n", x);
30:     printf("Incrementing...\n");
31:     increment();
32:     printf("Incremented!\n");
33:     printf("x is now %d\n", x);
34: }
35:
36:
37: /*
38:  * void
39:  * increment()
40:  *
41:  * Increments x.
42:  */
43:
44: void
45: increment()
46: {
47:     x++;
48: }
```

hai.cc 1/1

lectures/2/src/

```
1: #include <iostream>
2:
3: using namespace std;
4:
5: int
6: main(int argc, char * argv[])
7: {
8:     cout << "O hai, world!" << endl;
9: }
```

hai.lisp 1/1
lectures/2/src/

```
1: (print "O hai, world!")
```

hai.php 1/1
lectures/2/src/

```
1: <?
2:     echo "O hai, world!\n";
3: ?>
```

```
1: MAIN:
2: {
3:     print "O hai, world!\n";
4: }
```



```
 1: /**************************************************************************
 2:  * return1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Increments a variable.
 8:  *
 9:  * Demonstrates use of parameter and return value.
10:  **************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: // function prototype
16: int increment(int);
17:
18:
19: int
20: main(int argc, char *argv[])
21: {
22:     int x = 1;
23:     printf("x is now %d\n", x);
24:     printf("Incrementing...\n");
25:     x = increment(x);
26:     printf("Incremented!\n");
27:     printf("x is now %d\n", x);
28: }
29:
30:
31: /*
32:  * int
33:  * increment(int a)
34:  *
35:  * Returns argument plus one.
36:  */
37:
38: int
39: increment(int a)
40: {
41:     return a + 1;
42: }
```

```
 1: /***************************************************************************
 2:  * return2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Cubes a variable.
 8:  *
 9:  * Demonstrates use of parameter and return value.
10:  ***************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: // function prototype
16: int cube(int);
17:
18:
19: int
20: main(int argc, char *argv[])
21: {
22:     int x = 1;
23:     printf("x is now %d\n", x);
24:     printf("Cubing...\n");
25:     x = cube(x);
26:     printf("Cubed!\n");
27:     printf("x is now %d\n", x);
28: }
29:
30:
31: /*
32:  * int
33:  * cube(int a)
34:  *
35:  * Cubes argument.
36:  */
37:
38: int
39: cube(int a)
40: {
41:     return a * a * a;
42: }
```



```
 1: /***************************************************************************
 2:  * string1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Prints a given string one character per line.
 8:  *
 9:  * Demonstrates strings as arrays of chars and use of strlen.
10:  ***************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <string.h>
15:
16:
17: int
18: main(int argc, char *argv[])
19: {
20:     char c;
21:     int i;
22:     string s;
23:
24:     // get line of text
25:     s = GetString();
26:
27:     // print string, one character per line
28:     if (s != NULL)
29:     {
30:         for (i = 0; i < strlen(s); i++)
31:         {
32:             c = s[i];
33:             printf("%c\n", c);
34:         }
35:     }
36: }
```

```
 1: /****************************************************************************
 2:  * string2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Prints a given string one character per line.
 8:  *
 9:  * Demonstrates strings as arrays of chars with slight optimization.
10:  ****************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <string.h>
15:
16:
17: int
18: main(int argc, char *argv[])
19: {
20:     int i, n;
21:     string s;
22:
23:     // get line of text
24:     s = GetString();
25:
26:     // print string, one character per line
27:     if (s != NULL)
28:     {
29:         for (i = 0, n = strlen(s); i < n; i++)
30:         {
31:             printf("%c\n", s[i]);
32:         }
33:     }
34: }
```