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

argv2.c

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:     // print arguments
20:     printf("\n");
21:     for (int i = 0; i < argc; i++)
22:     {
23:         for (int j = 0, n = strlen(argv[i]); j < n; j++)
24:             printf("%c\n", argv[i][j]);
25:         printf("\n");
26:     }
27: }
```

ascii1.c

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(void)
18: {
19:     // display mapping for uppercase letters
20:     for (int i = 65; i < 65 + 26; i++)
21:         printf("%c: %d\n", (char) i, i);
22:
23:     // separate uppercase from lowercase
24:     printf("\n");
25:
26:     // display mapping for lowercase letters
27:     for (int i = 97; i < 97 + 26; i++)
28:         printf("%c: %d\n", (char) i, i);
29: }
30:
```

ascii2.c

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(void)
18: {
19:     // display mapping for uppercase letters
20:     for (int i = 65; i < 65 + 26; i++)
21:         printf("%c  %d    %3d  %c\n", (char) i, i, i + 32, (char) (i + 32));
22: }
23:
```

ascii3.c

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(void)
18: {
19:     // display mapping for uppercase letters
20:     for (char c = 'A'; c <= 'Z'; c = (char) ((int) c + 1))
21:         printf("%c: %d\n", c, (int) c);
22: }
```

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(void)
17: {
18:     // print top row of numbers
19:     printf("\n   ");
20:     for (int i = 1; i <= 10; i++)
21:         printf("%d  ", i);
22:     printf("\n");
23:
24:     // print rows of holes, with letters in leftmost column
25:     for (int i = 0; i < 10; i++)
26:     {
27:         printf("%c  ", 'A' + i);
28:         for (int j = 1; j <= 10; j++)
29:             printf("o  ");
30:         printf("\n");
31:     }
32:     printf("\n");
33: }
```

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(void)
19: {
20:     // ask user for number
21:     printf("How many bottles will there be? ");
22:     int n = GetInt();
23:
24:     // exit upon invalid input
25:     if (n < 1)
26:     {
27:         printf("Sorry, that makes no sense.\n");
28:         return 1;
29:     }
30:
31:     // sing the annoying song
32:     printf("\n");
33:     for (int i = n; i > 0; i--)
34:     {
35:         printf("%d bottle(s) of beer on the wall,\n", i);
36:         printf("%d bottle(s) of beer,\n", i);
37:         printf("Take one down, pass it around,\n");
38:         printf("%d bottle(s) of beer on the wall.\n\n", i - 1);
39:     }
40:
41:     // exit when song is over
42:     printf("Wow, that's annoying.\n");
43:     return 0;
44: }
```

buggy1.c 1/1
lectures/2/src/

```
 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(void)
15: {
16:     for (int i = 0; i <= 10; i++)
17:         printf("*");
18: }
```

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(void)
15: {
16:     for (int i = 0; i <= 10; i++)
17:         printf("*");
18:         printf("\n");
19: }
```

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 a, int b);
16:
17:
18: int
19: main(void)
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:  * Swap arguments' values.
36:  */
37:
38: void
39: swap(int a, int b)
40: {
41:     int tmp = a;
42:     a = b;
43:     b = tmp;
44: }
```

buggy4.c 1/1
lectures/2/src/

```
 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(void);
16:
17:
18: int
19: main(void)
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:  * Tries to increment x.
32:  */
33:
34: void
35: increment(void)
36: {
37:     x++;
38: }
```

buggy5.c 1/1
lectures/2/src/

```
 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(void);
19:
20:
21: int
22: main(void)
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:  * Increments x.
38:  */
39:
40: void
41: increment(void)
42: {
43:     int x = 10;
44:     x++;
45: }
```

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(void)
23: {
24:     float grades[QUIZZES];
25:
26:     // ask user for scores
27:     printf("\nWhat were your quiz scores?\n\n");
28:     for (int i = 0; i < QUIZZES; i++)
29:     {
30:         printf("Quiz #%d of %d: ", i+1, QUIZZES);
31:         grades[i] = GetFloat();
32:     }
33:
34:     // print scores
35:     for (int i = 0; i < 3; i++)
36:         printf("%.2f\n", grades[i]);
37: }
```

```
 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(void)
19: {
20:     // get line of text
21:     string s = GetString();
22:
23:     // capitalize text
24:     for (int i = 0, n = strlen(s); i < n; i++)
25:     {
26:         if (s[i] >= 'a' && s[i] <= 'z')
27:             printf("%c", s[i] - ('a' - 'A'));
28:         else
29:             printf("%c", s[i]);
30:     }
31:     printf("\n");
32: }
```



```
 1: /****************************************************************************
 2:  * cs50.c
 3:  *
 4:  * version 1.1.5
 5:  *
 6:  * Computer Science 50
 7:  * Glenn Holloway
 8:  * David J. Malan
 9:  *
10:  * Definitions for CS50's library.
11:  * Based on Eric Roberts' genlib.c and simpio.c.
12:  *
13:  * The latest version of this file can be found at
14:  * http://cs50.net/pub/releases/cs50/cs50.c.
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: #include <stdio.h>
25: #include <stdlib.h>
26: #include <string.h>
27:
28: #include "cs50.h"
29:
30:
31: /*
32:  * Default capacity of buffer for standard input.
33:  */
34:
35: #define CAPACITY 128
36:
37:
38: /*
39:  * Reads a line of text from standard input and returns the equivalent
40:  * char; if text does not represent a char, user is prompted to retry.
41:  * Leading and trailing whitespace is ignored.  If line can't be read,
42:  * returns CHAR_MAX.
43:  */
44:
45: char
46: GetChar(void)
47: {
48:     // try to get a char from user
49:     while (true)
50:     {
51:         // get line of text, returning CHAR_MAX on failure
52:         string line = GetString();
53:         if (line == NULL)
54:             return CHAR_MAX;
55:
56:         // return a char if only a char (possibly with
57:         // leading and/or trailing whitespace) was provided
58:         char c1, c2;
59:         if (sscanf(line, " %c %c", &c1, &c2) == 1)
60:         {
61:             free(line);
62:             return c1;
63:         }
64:         else
```

```
65:         {
66:             free(line);
67:             printf("Retry: ");
68:         }
69:     }
70: }
71:
72:
73: /*
74:  * Reads a line of text from standard input and returns the equivalent
75:  * double as precisely as possible; if text does not represent a
76:  * double, user is prompted to retry.  Leading and trailing whitespace
77:  * is ignored.  For simplicity, overflow and underflow are not detected.
78:  * If line can't be read, returns DBL_MAX.
79:  */
80:
81: double
82: GetDouble(void)
83: {
84:     // try to get a double from user
85:     while (true)
86:     {
87:         // get line of text, returning DBL_MAX on failure
88:         string line = GetString();
89:         if (line == NULL)
90:             return DBL_MAX;
91:
92:         // return a double if only a double (possibly with
93:         // leading and/or trailing whitespace) was provided
94:         double d; char c;
95:         if (sscanf(line, " %lf %c", &d, &c) == 1)
96:         {
97:             free(line);
98:             return d;
99:         }
100:        else
101:        {
102:            free(line);
103:            printf("Retry: ");
104:        }
105:    }
106: }
107:
108:
109: /*
110:  * Reads a line of text from standard input and returns the equivalent
111:  * float as precisely as possible; if text does not represent a float,
112:  * user is prompted to retry.  Leading and trailing whitespace is ignored.
113:  * For simplicity, overflow and underflow are not detected.  If line can't
114:  * be read, returns FLT_MAX.
115:  */
116:
117: float
118: GetFloat(void)
119: {
120:     // try to get a float from user
121:     while (true)
122:     {
123:         // get line of text, returning FLT_MAX on failure
124:         string line = GetString();
125:         if (line == NULL)
126:             return FLT_MAX;
127:
128:         // return a float if only a float (possibly with
```



```
129:         // leading and/or trailing whitespace) was provided
130:         char c; float f;
131:         if (sscanf(line, " %f %c", &f, &c) == 1)
132:         {
133:             free(line);
134:             return f;
135:         }
136:         else
137:         {
138:             free(line);
139:             printf("Retry: ");
140:         }
141:     }
142: }
143:
144:
145: /*
146:  * Reads a line of text from standard input and returns it as an
147:  * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
148:  * does not represent such an int, user is prompted to retry.  Leading
149:  * and trailing whitespace is ignored.  For simplicity, overflow is not
150:  * detected.  If line can't be read, returns INT_MAX.
151:  */
152:
153: int
154: GetInt(void)
155: {
156:     // try to get an int from user
157:     while (true)
158:     {
159:         // get line of text, returning INT_MAX on failure
160:         string line = GetString();
161:         if (line == NULL)
162:             return INT_MAX;
163:
164:         // return an int if only an int (possibly with
165:         // leading and/or trailing whitespace) was provided
166:         int n; char c;
167:         if (sscanf(line, " %d %c", &n, &c) == 1)
168:         {
169:             free(line);
170:             return n;
171:         }
172:         else
173:         {
174:             free(line);
175:             printf("Retry: ");
176:         }
177:     }
178: }
179:
180:
181: /*
182:  * Reads a line of text from standard input and returns an equivalent
183:  * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
184:  * does not represent such a long long, user is prompted to retry.
185:  * Leading and trailing whitespace is ignored.  For simplicity, overflow
186:  * is not detected.  If line can't be read, returns LLONG_MAX.
187:  */
188:
189: long long
190: GetLongLong(void)
191: {
192:     // try to get a long long from user
```

```
193:     while (true)
194:     {
195:         // get line of text, returning LLONG_MAX on failure
196:         string line = GetString();
197:         if (line == NULL)
198:             return LLONG_MAX;
199:
200:         // return a long long if only a long long (possibly with
201:         // leading and/or trailing whitespace) was provided
202:         long long n; char c;
203:         if (sscanf(line, " %lld %c", &n, &c) == 1)
204:         {
205:             free(line);
206:             return n;
207:         }
208:         else
209:         {
210:             free(line);
211:             printf("Retry: ");
212:         }
213:     }
214: }
215:
216:
217: /*
218:  * Reads a line of text from standard input and returns it as a string,
219:  * sans trailing newline character.  (Ergo, if user inputs only "\n",
220:  * returns "" not NULL.)  Leading and trailing whitespace is not ignored.
221:  * Returns NULL upon error or no input whatsoever (i.e., just EOF).
222:  */
223:
224: string
225: GetString(void)
226: {
227:     // growable buffer for chars
228:     string buffer = NULL;
229:
230:     // capacity of buffer
231:     unsigned int capacity = 0;
232:
233:     // number of chars actually in buffer
234:     unsigned int n = 0;
235:
236:     // character read or EOF
237:     int c;
238:
239:     // iteratively get chars from standard input
240:     while ((c = fgetc(stdin)) != '\n' && c != EOF)
241:     {
242:         // grow buffer if necessary
243:         if (n + 1 > capacity)
244:         {
245:             // determine new capacity: start at CAPACITY then double
246:             if (capacity == 0)
247:                 capacity = CAPACITY;
248:             else if (capacity <= (UINT_MAX / 2))
249:                 capacity *= 2;
250:             else
251:             {
252:                 free(buffer);
253:                 return NULL;
254:             }
255:
256:             // extend buffer's capacity
```



```
257:             string temp = realloc(buffer, capacity * sizeof(char));
258:             if (temp == NULL)
259:             {
260:                 free(buffer);
261:                 return NULL;
262:             }
263:             buffer = temp;
264:         }
265:
266:         // append current character to buffer
267:         buffer[n++] = c;
268:     }
269:
270:     // return NULL if user provided no input
271:     if (n == 0 && c == EOF)
272:         return NULL;
273:
274:     // minimize buffer
275:     string minimal = malloc((n + 1) * sizeof(char));
276:     strncpy(minimal, buffer, n);
277:     free(buffer);
278:
279:     // terminate string
280:     minimal[n] = '\0';
281:
282:     // return string
283:     return minimal;
284: }
```

```
/*****************************************************************************
 * cs50.h
 *
 * version 1.1.5
 *
 * Computer Science 50
 * Glenn Holloway
 * David J. Malan
 *
 * Declarations for CS50's library.
 * Based on Eric Roberts' genlib.h and simpio.h.
 *
 * The latest version of this file can be found at
 * http://cs50.net/pub/releases/cs50/cs50.h.
 *
 * 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
 *****************************************************************************/

#ifndef _CS50_H
#define _CS50_H

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


/*
 * Borrow the standard library's data type for Boolean variables whose
 * values must be (true|false).
 */

#include <stdbool.h>


/*
 * Our own data type for string variables.
 */

typedef char *string;


/*
 * 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);


/*
 * 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);


/*
 * 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(void);


/*
 * 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(void);


/*
 * 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(void);


/*
 * 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(void);


#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(void);
20:
21:
22: int
23: main(void)
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:  * Increments x.
39:  */
40:
41: void
42: increment(void)
43: {
44:     x++;
45: }
```

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: ?>
```

hai.pl 1/1

lectures/2/src/

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

return1.c 1/1

lectures/2/src/

```
 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 a);
17:
18:
19: int
20: main(void)
21: {
22:     int x = 2;
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:  * Returns argument plus one.
33:  */
34:
35: int
36: increment(int a)
37: {
38:     return a + 1;
39: }
```

```
 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 a);
17:
18:
19: int
20: main(void)
21: {
22:     int x = 2;
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:  * Cubes argument.
33:  */
34:
35: int
36: cube(int a)
37: {
38:     return a * a * a;
39: }
```