

bar.c
lectures/weeks/5/src/

1/1

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

copy2.c
lectures/weeks/5/src/

1/1

```
0: ****
1: * copy2.c
2:
3: * Computer Science 50
4: * David J. Malan
5:
6: * Copies a string.
7:
8: * Demonstrates strings as pointers to arrays.
9: ****
10:
11: #include <cs50.h>
12: #include <ctype.h>
13: #include <stdio.h>
14: #include <stdlib.h>
15: #include <string.h>
16:
17:
18: int
19: main(int argc, char * argv[])
20: {
21:     // get line of text
22:     printf("Say something: ");
23:     char *s1 = GetString();
24:     if (s1 == NULL)
25:         return 1;
26:
27:     // allocate enough space for copy
28:     char *s2 = malloc(strlen(s1) * sizeof(char) + 1);
29:     if (s2 == NULL)
30:         return 1;
31:
32:     // copy string
33:     if (s2 != NULL)
34:     {
35:         int n = strlen(s1);
36:         for (int i = 0; i < n; i++)
37:             s2[i] = s1[i];
38:         s2[n] = '\0';
39:     }
40:
41:     // change copy
42:     printf("Capitalizing copy...\n");
43:     if (strlen(s2) > 0)
44:         s2[0] = toupper(s2[0]);
45:
46:     // print original and copy
47:     printf("Original: %s\n", s1);
48:     printf("Copy:      %s\n", s2);
49:
50:     // free memory
51:     free(s1);
52:     free(s2);
53: }
```

```
0: ****
1: * cs50.c
2: *
3: * version 1.1.2
4: *
5: * Computer Science 50
6: * Glenn Holloway
7: * David J. Malan
8: *
9: * Definitions for CS 50's library.
10: * Based on Eric Roberts' genlib.c and simpio.c.
11: *
12: * The latest version of this file can be found at
13: * http://cs50.net/pub/releases/cs50/cs50.c.
14: *
15: * To compile as a static library on your own system:
16: * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
17: * % ar rcs libcs50.a cs50.o
18: * % rm -f cs50.o
19: * % cp cs50.h /usr/local/include
20: * % cp libcs50.a /usr/local/lib
21: ****
22:
23: #include <stdio.h>
24: #include <stdlib.h>
25: #include <string.h>
26:
27: #include "cs50.h"
28:
29:
30: /*
31: * Default capacity of buffer for standard input.
32: */
33:
34: #define CAPACITY 128
35:
36:
37: /*
38: * char
39: * GetChar()
40: *
41: * Reads a line of text from standard input and returns the equivalent
42: * char; if text does not represent a char, user is prompted to retry.
43: * Leading and trailing whitespace is ignored. If line can't be read,
44: * returns CHAR_MAX.
45: */
46:
47: char
48: GetChar()
49: {
50:     // try to get a char from user
51:     while (true)
52:     {
53:         // get line of text, returning CHAR_MAX on failure
54:         string line = GetString();
55:         if (line == NULL)
56:             return CHAR_MAX;
57:
58:         // return a char if only a char (possibly with
59:         // leading and/or trailing whitespace) was provided
60:         char c1, c2;
61:         if (sscanf(line, " %c %c", &c1, &c2) == 1)
62:         {
63:             free(line);
64:             return c1;
65:         }
66:         else
67:         {
68:             free(line);
69:             printf("Retry: ");
70:         }
71:     }
72: }
73:
74: /*
75: * double
76: * GetDouble()
77: *
78: * Reads a line of text from standard input and returns the equivalent
79: * double as precisely as possible; if text does not represent a
80: * double, user is prompted to retry. Leading and trailing whitespace
81: * is ignored. For simplicity, overflow and underflow are not detected.
82: * If line can't be read, returns DBL_MAX.
83: */
84:
85:
86: double
87: GetDouble()
88: {
89:     // try to get a double from user
90:     while (true)
91:     {
92:         // get line of text, returning DBL_MAX on failure
93:         string line = GetString();
94:         if (line == NULL)
95:             return DBL_MAX;
96:
97:         // return a double if only a double (possibly with
98:         // leading and/or trailing whitespace) was provided
99:         double d; char c;
100:        if (sscanf(line, " %lf %c", &d, &c) == 1)
101:        {
102:            free(line);
103:            return d;
104:        }
105:        else
106:        {
107:            free(line);
108:            printf("Retry: ");
109:        }
110:    }
111: }
112:
113:
114: /*
115: * float
116: * GetFloat()
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
126: GetFloat()
127: {
```

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

```
192: /*
193: * long long
194: * GetLongLong()
195: *
196: * Reads a line of text from standard input and returns an equivalent
197: * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
198: * does not represent such a long long, user is prompted to retry.
199: * Leading and trailing whitespace is ignored. For simplicity, overflow
200: * is not detected. If line can't be read, returns LLONG_MAX.
201: */
202:
203: long long
204: GetLongLong()
205: {
206:     // try to get a long long from user
207:     while (true)
208:     {
209:         // get line of text, returning LLONG_MAX on failure
210:         string line = GetString();
211:         if (line == NULL)
212:             return LLONG_MAX;
213:
214:         // return a long long if only a long long (possibly with
215:         // leading and/or trailing whitespace) was provided
216:         long long n; char c;
217:         if (sscanf(line, " %lld %c", &n, &c) == 1)
218:         {
219:             free(line);
220:             return n;
221:         }
222:         else
223:         {
224:             free(line);
225:             printf("Retry: ");
226:         }
227:     }
228: }
229:
230:
231: /*
232: * string
233: * GetString()
234: *
235: * Reads a line of text from standard input and returns it as a string,
236: * sans trailing newline character. (Ergo, if user inputs only "\n",
237: * returns "" not NULL.) Leading and trailing whitespace is not ignored.
238: * Returns NULL upon error or no input whatsoever (i.e., just EOF).
239: */
240:
241: string
242: GetString()
243: {
244:     // growable buffer for chars
245:     string buffer = NULL;
246:
247:     // capacity of buffer
248:     unsigned int capacity = 0;
249:
250:     // number of chars actually in buffer
251:     unsigned int n = 0;
252:
253:     // character read or EOF
254:     int c;
255:
```

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

```
0: ****
1: * cs50.h
2: *
3: * version 1.1.2
4: *
5: * Computer Science 50
6: * Glenn Holloway
7: * David J. Malan
8: *
9: * Declarations for CS 50's library.
10: * Based on Eric Roberts' genlib.h and simpio.h.
11: *
12: * The latest version of this file can be found at
13: * http://cs50.net/pub/releases/cs50/cs50.h.
14: *
15: * To compile as a static library on your own system:
16: * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
17: * % ar rcs libcs50.a cs50.o
18: * % rm -f cs50.o
19: * % cp cs50.h /usr/local/include
20: * % cp libcs50.a /usr/local/lib
21: ****
22:
23: #ifndef _CS50_H
24: #define _CS50_H
25:
26: #include <float.h>
27: #include <limits.h>
28:
29:
30: /*
31: * bool
32: *
33: * Borrow the standard library's data type for Boolean variables whose
34: * values must be (true|false).
35: */
36:
37: #include <stdbool.h>
38:
39:
40: /*
41: * string
42: *
43: * Our own data type for string variables.
44: */
45:
46: typedef char *string;
47:
48:
49: /*
50: * char
51: * GetChar()
52: *
53: * Reads a line of text from standard input and returns the equivalent
54: * char; if text does not represent a char, user is prompted to retry.
55: * Leading and trailing whitespace is ignored. If line can't be read,
56: * returns CHAR_MAX.
57: */
58:
59: char GetChar();
60:
61:
62: /*
63: * double
```

```
64: * GetDouble()
65: *
66: * Reads a line of text from standard input and returns the equivalent
67: * double as precisely as possible; if text does not represent a
68: * double, user is prompted to retry. Leading and trailing whitespace
69: * is ignored. For simplicity, overflow and underflow are not detected.
70: * If line can't be read, returns DBL_MAX.
71: */
72:
73: double GetDouble();
74:
75:
76: /*
77: * float
78: * GetFloat()
79: *
80: * Reads a line of text from standard input and returns the equivalent
81: * float as precisely as possible; if text does not represent a float,
82: * user is prompted to retry. Leading and trailing whitespace is ignored.
83: * For simplicity, overflow and underflow are not detected. If line can't
84: * be read, returns FLT_MAX.
85: */
86:
87: float GetFloat();
88:
89:
90: /*
91: * int
92: * GetInt()
93: *
94: * Reads a line of text from standard input and returns it as an
95: * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
96: * does not represent such an int, user is prompted to retry. Leading
97: * and trailing whitespace is ignored. For simplicity, overflow is not
98: * detected. If line can't be read, returns INT_MAX.
99: */
100:
101: int GetInt();
102:
103:
104: /*
105: * long long
106: * GetLongLong()
107: *
108: * Reads a line of text from standard input and returns an equivalent
109: * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
110: * does not represent such a long long, user is prompted to retry.
111: * Leading and trailing whitespace is ignored. For simplicity, overflow
112: * is not detected. If line can't be read, returns LLONG_MAX.
113: */
114:
115: long long GetLongLong();
116:
117:
118: /*
119: * string
120: * GetString()
121: *
122: * Reads a line of text from standard input and returns it as a string,
123: * sans trailing newline character. (Ergo, if user inputs only "\n",
124: * returns "" not NULL.) Leading and trailing whitespace is not ignored.
125: * Returns NULL upon error or no input whatsoever (i.e., just EOF).
126: */
127:
```

```
128: string GetString();
129:
130:
131:
132: #endif
133:
```

scanf1.c
lectures/weeks/5/src/

1/1

```
0: ****
1: *  scanf1.c
2: *
3: * Computer Science 50
4: * David J. Malan
5: *
6: * Reads a number from the user into an int.
7: *
8: * Demonstrates scanf and address-of operator.
9: ****
10:
11: #include <stdio.h>
12:
13:
14: int
15: main(int argc, char *argv[])
16: {
17:     int x;
18:     printf("Number please: ");
19:     scanf("%d", &x);
20:     printf("Thanks for the %d!\n", x);
21: }
```

scanf2.c
lectures/weeks/5/src/

1/1

```
0: ****
1: *  scanf2.c
2: *
3: * Computer Science 50
4: * David J. Malan
5: *
6: * Reads a string from the user into memory it shouldn't.
7: *
8: * Demonstrates possible attack!
9: ****
10:
11: #include <stdio.h>
12:
13:
14: int
15: main(int argc, char *argv[])
16: {
17:     char *buffer;
18:     printf("String please: ");
19:     scanf("%s", buffer);
20:     printf("Thanks for the \"%s\"!\n", buffer);
21: }
```

scanf3.c
lectures/weeks/5/src/

1/1

```
0: ****
1: *  scanf3.c
2: *
3: * Computer Science 50
4: * David J. Malan
5: *
6: * Reads a string from the user into an array (dangerously).
7: *
8: * Demonstrates potential buffer overflow!
9: ****
10:
11: #include <stdio.h>
12:
13:
14: int
15: main(int argc, char *argv[])
16: {
17:     char buffer[16];
18:     printf("String please: ");
19:     scanf("%s", buffer);
20:     printf("Thanks for the \"%s\"!\n", buffer);
21: }
```

structs1.c
lectures/weeks/5/src/

1/1

```
0: ****
1: *  structs1.c
2: *
3: * Computer Science 50
4: * David J. Malan
5: *
6: * Demonstrates use of structs.
7: ****
8:
9: #include <cs50.h>
10: #include <stdio.h>
11: #include <stdlib.h>
12: #include <string.h>
13:
14: #include "structs.h"
15:
16:
17: // class size
18: #define STUDENTS 3
19:
20:
21: int
22: main(int argc, char *argv[])
23: {
24:     // declare class
25:     student class[STUDENTS];
26:
27:     // populate class with user's input
28:     for (int i = 0; i < STUDENTS; i++)
29:     {
30:         printf("Student's ID: ");
31:         class[i].id = GetInt();
32:
33:         printf("Student's name: ");
34:         class[i].name = GetString();
35:
36:         printf("Student's house: ");
37:         class[i].house = GetString();
38:         printf("\n");
39:     }
40:
41:     // now print anyone in Mather
42:     for (int i = 0; i < STUDENTS; i++)
43:         if (strcmp(class[i].house, "Mather") == 0)
44:             printf("%s is in Mather!\n", class[i].name);
45:
46:     // free memory
47:     for (int i = 0; i < STUDENTS; i++)
48:     {
49:         free(class[i].name);
50:         free(class[i].house);
51:     }
52: }
```

structs2.c
lectures/weeks/5/src/

1/2

```
0: ****
1: * structs.c
2: *
3: * Computer Science 50
4: * David J. Malan
5: *
6: * Demonstrates use of structs.
7: ****
8:
9: #include <cs50.h>
10: #include <stdio.h>
11: #include <stdlib.h>
12: #include <string.h>
13:
14: #include "structs.h"
15:
16:
17: // class size
18: #define STUDENTS 3
19:
20:
21: int
22: main(int argc, char *argv[])
23: {
24:     // declare class
25:     student class[STUDENTS];
26:
27:     // populate class with user's input
28:     for (int i = 0; i < STUDENTS; i++)
29:     {
30:         printf("Student's ID: ");
31:         class[i].id = GetInt();
32:
33:         printf("Student's name: ");
34:         class[i].name = GetString();
35:
36:         printf("Student's house: ");
37:         class[i].house = GetString();
38:         printf("\n");
39:     }
40:
41:     // now print anyone in Mather
42:     for (int i = 0; i < STUDENTS; i++)
43:         if (strcmp(class[i].house, "Mather") == 0)
44:             printf("%s is in Mather!\n\n", class[i].name);
45:
46:     // let's save these students to disk
47:     FILE *fp = fopen("database", "w");
48:     if (fp != NULL)
49:     {
50:         for (int i = 0; i < STUDENTS; i++)
51:         {
52:             fprintf(fp, "%d\n", class[i].id);
53:             fprintf(fp, "%s\n", class[i].name);
54:             fprintf(fp, "%s\n", class[i].house);
55:         }
56:     fclose(fp);
57: }
58:
59: // free memory
60: for (int i = 0; i < STUDENTS; i++)
61: {
62:     free(class[i].name);
63:     free(class[i].house);
```

structs2.c
lectures/weeks/5/src/

2/2

```
64:     }
65: }
66:
```

```
0: /*****
1:  * structs.h
2:  *
3:  * Computer Science 50
4:  * David J. Malan
5:  *
6:  * Defines a student for structs{1,2,3}.c.
7:  *****/
8:
9:
10: // structure representing a student
11: typedef struct
12: {
13:     int id;
14:     char *name;
15:     char *house;
16: }
17: student;
18:
```