

bar.c1/1

lectures/4/src/

```
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: }
```

compare1.c1/1

lectures/4/src/

```
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: }
```

compare2.c

1/1

lectures/4/src/

```
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: }
```

copy1.c

1/1

lectures/4/src/

```
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: }
```

copy2.c

lectures/4/src/

1/1

```

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: }

```

cs50.c

lectures/4/src/

1/5

```

1: /*****
2:  * cs50.c
3:  *
4:  * version 1.1.6
5:  *
6:  * Computer Science 50
7:  * Glenn Holloway
8:  * David J. Malan
9:  *
10: * Definitions for the CS50 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://www.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:  * Reads a line of text from standard input and returns the equivalent
74:  * double as precisely as possible; if text does not represent a
75:  * double, user is prompted to retry. Leading and trailing whitespace
76:  * is ignored. For simplicity, overflow and underflow are not detected.
77:  * If line can't be read, returns DBL_MAX.
78:  */
79:
80: double
81: GetDouble(void)
82: {
83:     // try to get a double from user
84:     while (true)
85:     {
86:         // get line of text, returning DBL_MAX on failure
87:         string line = GetString();
88:         if (line == NULL)
89:             return DBL_MAX;
90:
91:         // return a double if only a double (possibly with
92:         // leading and/or trailing whitespace) was provided
93:         double d; char c;
94:         if (sscanf(line, "%lf %c", &d, &c) == 1)
95:         {
96:             free(line);
97:             return d;
98:         }
99:         else
100:         {
101:             free(line);
102:             printf("Retry: ");
103:         }
104:     }
105: }
106: }
107:
108: /*
109:  * Reads a line of text from standard input and returns the equivalent
110:  * float as precisely as possible; if text does not represent a float,
111:  * user is prompted to retry. Leading and trailing whitespace is ignored.
112:  * For simplicity, overflow and underflow are not detected. If line can't
113:  * be read, returns FLT_MAX.
114:  */
115:
116: float
117: GetFloat(void)
118: {
119:     // try to get a float from user
120:     while (true)
121:     {
122:         // get line of text, returning FLT_MAX on failure
123:         string line = GetString();
124:         if (line == NULL)
125:             return FLT_MAX;
126:
127:         // 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:  * Reads a line of text from standard input and returns it as an
146:  * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
147:  * does not represent such an int, user is prompted to retry. Leading
148:  * and trailing whitespace is ignored. For simplicity, overflow is not
149:  * detected. If line can't be read, returns INT_MAX.
150:  */
151:
152: int
153: GetInt(void)
154: {
155:     // try to get an int from user
156:     while (true)
157:     {
158:         // get line of text, returning INT_MAX on failure
159:         string line = GetString();
160:         if (line == NULL)
161:             return INT_MAX;
162:
163:         // return an int if only an int (possibly with
164:         // leading and/or trailing whitespace) was provided
165:         int n; char c;
166:         if (sscanf(line, "%d %c", &n, &c) == 1)
167:         {
168:             free(line);
169:             return n;
170:         }
171:         else
172:         {
173:             free(line);
174:             printf("Retry: ");
175:         }
176:     }
177: }
178: }
179:
180: /*
181:  * Reads a line of text from standard input and returns an equivalent
182:  * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
183:  * does not represent such a long long, user is prompted to retry.
184:  * Leading and trailing whitespace is ignored. For simplicity, overflow
185:  * is not detected. If line can't be read, returns LLONG_MAX.
186:  */
187:
188: long long
189: GetLongLong(void)
190: {
191:     // 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: }
```

```

1: /*****
2:  * cs50.h
3:  *
4:  * version 1.1.6
5:  *
6:  * Computer Science 50
7:  * Glenn Holloway
8:  * David J. Malan
9:  *
10: * Declarations for the CS50 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://www.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:  * Borrow the standard library's data type for Boolean variables whose
33:  * values must be (true|false).
34:  */
35:
36: #include <stdbool.h>
37:
38:
39: /*
40:  * Our own data type for string variables.
41:  */
42:
43: typedef char *string;
44:
45:
46: /*
47:  * Reads a line of text from standard input and returns the equivalent
48:  * char; if text does not represent a char, user is prompted to retry.
49:  * Leading and trailing whitespace is ignored. If line can't be read,
50:  * returns CHAR_MAX.
51:  */
52:
53: char
54: GetChar(void);
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:
69: /*
70:  * Reads a line of text from standard input and returns the equivalent
71:  * float as precisely as possible; if text does not represent a float,
72:  * user is prompted to retry. Leading and trailing whitespace is ignored.
73:  * For simplicity, overflow and underflow are not detected. If line can't
74:  * be read, returns FLT_MAX.
75:  */
76:
77: float
78: GetFloat(void);
79:
80:
81: /*
82:  * Reads a line of text from standard input and returns it as an
83:  * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
84:  * does not represent such an int, user is prompted to retry. Leading
85:  * and trailing whitespace is ignored. For simplicity, overflow is not
86:  * detected. If line can't be read, returns INT_MAX.
87:  */
88:
89: int
90: GetInt(void);
91:
92:
93: /*
94:  * Reads a line of text from standard input and returns an equivalent
95:  * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
96:  * does not represent such a long long, user is prompted to retry.
97:  * Leading and trailing whitespace is ignored. For simplicity, overflow
98:  * is not detected. If line can't be read, returns LLONG_MAX.
99:  */
100:
101: long long
102: GetLongLong(void);
103:
104:
105: /*
106:  * Reads a line of text from standard input and returns it as a string,
107:  * sans trailing newline character. (Ergo, if user inputs only "\n",
108:  * returns "" not NULL.) Leading and trailing whitespace is not ignored.
109:  * Returns NULL upon error or no input whatsoever (i.e., just EOF).
110:  */
111:
112: string GetString(void);
113:
114:
115:
116: #endif

```

```

1: /*
2:
3:   iUnlock v42.PROPER -- Copyright 2007 The dev team
4:
5:   Credits: Daeken, Darkmen, guest184, gray, iZsh, pytey, roxfan, Sam, uns, Zappa
z, zf
6:
7:   All code, information or data [from now on "data"] available
8:   from the "iPhone dev team" [1] or any other project linked from
9:   this or other pages is owned by the creator who created the data.
10:  The copyright, license right, distribution right and any other
11:  rights lies with the creator.
12:
13:  It is prohibited to use the data without the written agreement
14:  of the creator. This included using ideas in other projects
15:  (commercial or not commercial).
16:
17:  Where data was created by more than 1 creator a written agreement
18:  from each of the creators has to be obtained.
19:
20:  Punishment: Monkeys coming out of your ass Bruce Almighty style.
21:
22:  [1] http://iphone.fiveforty.net/wiki/index.php?title=Main_Page
23: */
24: #include <stdio.h>
25: #include <stdlib.h>
26: #include <unistd.h>
27: #include <string.h>
28: #include <fcntl.h>
29: #include <termios.h>
30: #include <errno.h>
31: #include <time.h>
32: #include "IOKit/IOKitLib.h"
33:
34: #include "packets.h"
35:
36: #define ever ;;
37: #define LOG stdout
38: #define SPEED 750000
39:
40: #pragma pack(1)
41:
42: #define BUFSIZE (65536+100)
43: unsigned char readbuf[BUFSIZE];
44:
45: struct termios term;
46:
47: // #define DEBUG_ENABLED 1
48:
49: #ifndef DEBUG_ENABLED
50: #define DEBUGLOG(x)
51: #else
52: #define DEBUGLOG(x) x
53: #endif
54:
55: #define UINT(x) *((unsigned int *) (x))
56:
57: const char * RE = "Why the hell are you reversing this app?! We said we were \"
58: "going to release the sources...";
59:
60: void HexDumpLine(unsigned char *buf, int remainder, int offset)
61: {
62:   int i = 0;
63:   char c = 0;

```

```

64:
65:   // Print the hex part
66:   fprintf(LOG, "%08x | ", offset);
67:   for (i = 0; i < 16; ++i) {
68:     if (i < remainder)
69:       fprintf(LOG, "%02x%s", buf[i], (i == 7) ? " " : "");
70:     else
71:       fprintf(LOG, " %s", (i == 7) ? " " : "");
72:   }
73:   // Print the ascii part
74:   fprintf(LOG, " | ");
75:   for (i = 0; i < 16 && i < remainder; ++i) {
76:     c = buf[i];
77:     if (c >= 0x20 && c <= 0x7e)
78:       fprintf(LOG, "%c%s", c, (i == 7) ? " " : "");
79:     else
80:       fprintf(LOG, ".%s", (i == 7) ? " " : "");
81:   }
82:
83:   fprintf(LOG, "\n");
84: }
85:
86: void HexDump(unsigned char *buf, int size)
87: {
88:   int i = 0;
89:
90:   for (i = 0; i < size; i += 16)
91:     HexDumpLine(buf + i, size - i, i);
92:   fprintf(LOG, "%08x\n", size);
93: }
94:
95: int Checksum(CmdHeader * packet)
96: {
97:   int sum = 0x00030000;
98:   sum += packet->opcode;
99:   sum += packet->param_len;
100:
101:   int len = packet->param_len;
102:   unsigned char * buf = ((unsigned char *) packet) + sizeof (CmdHeader);
103:   int i = 0;
104:
105:   for (i = 0; i < len; ++i)
106:     sum += buf[i];
107:   return sum;
108: }
109:
110: void SendCmd(int fd, void *buf, size_t size)
111: {
112:   DEBUGLOG(fprintf(LOG, "Sending:\n"));
113:   DEBUGLOG(HexDump((unsigned char *) buf, size));
114:
115:   if (write(fd, buf, size) == -1) {
116:     fprintf(stderr, "Shit. %s\n", strerror(errno));
117:     exit(1);
118:   }
119: }
120:
121: #define sendBytes(fd, args...) {\
122:   unsigned char sendbuf[] = {args}; \
123:   SendCmd(fd, sendbuf, sizeof(sendbuf)); \
124: }
125:
126: int ReadResp(int fd)
127: {

```

```

128: int len = 0;
129: struct timeval timeout;
130: int nfds = fd + 1;
131: fd_set readfds;
132:
133: FD_ZERO(&readfds);
134: FD_SET(fd, &readfds);
135:
136: // Wait a second
137: timeout.tv_sec = 0;
138: timeout.tv_usec = 500000;
139:
140: while (select(nfds, &readfds, NULL, NULL, &timeout) > 0)
141:     len += read(fd, readbuf + len, BUFSIZE - len);
142:
143: if (len > 0) {
144:     DEBUGLOG(fprintf(LOG, "Read:\n"));
145:     DEBUGLOG(HexDump(readbuf, len));
146: }
147: return len;
148: }
149:
150: int InitConn(int speed)
151: {
152:     int fd = open("/dev/tty.baseband", O_RDWR | O_20000 | O_NOCTTY);
153:     unsigned int blahnull = 0;
154:     unsigned int handshake = TIOCM_DTR | TIOCM_RTS | TIOCM_CTS | TIOCM_DSR;
155:
156:     if (fd == -1) {
157:         fprintf(stderr, "%i(%s)\n", errno, strerror(errno));
158:         exit(1);
159:     }
160:
161:     ioctl(fd, 0x2000740D);
162:     fcntl(fd, 4, 0);
163:     tcgetattr(fd, &term);
164:
165:     ioctl(fd, 0x8004540A, &blahnull);
166:     cfsetspeed(&term, speed);
167:     cfmakeraw(&term);
168:     term.c_cc[VMIN] = 0;
169:     term.c_cc[VTIME] = 5;
170:
171:     term.c_iflag = (term.c_iflag & 0xFFFFF0CD) | 5;
172:     term.c_oflag = term.c_oflag & 0xFFFFF0FE;
173:     term.c_cflag = (term.c_cflag & 0xFFFFC6CFF) | 0x3CB00;
174:     term.c_lflag = term.c_lflag & 0xFFFFFA77;
175:
176:     term.c_cflag = (term.c_cflag & ~CSIZE) | CS8;
177:     term.c_cflag &= ~PARENB;
178:     term.c_lflag &= ~ECHO;
179:
180:     tcsetattr(fd, TCSANOW, &term);
181:
182:     ioctl(fd, TIOCSDTTR);
183:     ioctl(fd, TIOCCDTR);
184:     ioctl(fd, TIOCMSET, &handshake);
185:
186:     return fd;
187: }
188:
189: void RestartBaseband()
190: {
191:     kern_return_t result;

```

```

192: mach_port_t masterPort;
193:
194: result = IOMasterPort(MACH_PORT_NULL, &masterPort);
195: if (result) {
196:     DEBUGLOG(sprintf("IOMasterPort failed\n"));
197:     return;
198: }
199:
200: CFMutableDictionaryRef matchingDict = IOServiceMatching("AppleBaseband");
201: io_service_t service = IOServiceGetMatchingService(kIOMasterPortDefault, match
ingDict);
202: if (!service) {
203:     DEBUGLOG(sprintf("IOServiceGetMatchingService failed\n"));
204:     return;
205: }
206:
207: io_connect_t conn;
208: result = IOServiceOpen(service, mach_task_self(), 0, &conn);
209: if (result) {
210:     DEBUGLOG(sprintf("IOServiceOpen failed\n"));
211:     return;
212: }
213:
214: result = IOConnectCallScalarMethod(conn, 0, 0, 0, 0, 0);
215: if (result == 0)
216:     DEBUGLOG(sprintf("Baseband reset.\n"));
217: else
218:     DEBUGLOG(sprintf("Baseband reset failed\n"));
219: IOServiceClose(conn);
220: }
221:
222: void SendGetVersion(int fd)
223: {
224:     sendBytes(fd, 0x60, 0x0D);
225: }
226:
227: void GetVersion(int fd)
228: {
229:     SendGetVersion(fd);
230:
231:     for (ever) {
232:         if (ReadResp(fd) != 0) {
233:             if (readbuf[0] == 0x0b)
234:                 break;
235:         }
236:         SendGetVersion(fd);
237:     }
238:
239:     VersionAck *ver = (VersionAck *) readbuf;
240:     DEBUGLOG(sprintf("Boot mode: %02X\n", ver->bootmode));
241:     DEBUGLOG(sprintf("Major: %d, Minor: %d\n", ver->major, ver->minor));
242:     DEBUGLOG(sprintf("Version: %s\n", ver->version));
243: }
244:
245: void CFISTagel_2(int fd)
246: {
247:     CFISTagelReq req;
248:     req.cmd.cls = 0x2;
249:     req.cmd.opcode = BBCFISTAGEL;
250:     req.cmd.param_len = 0;
251:     req.checksum = Checksum ((CmdHeader*)&req);
252:     DEBUGLOG(sprintf("Sending CFISTagel Request\n"));
253:     SendCmd(fd, &req, sizeof (CFISTagelReq));
254:     DEBUGLOG(sprintf("Receiving CFISTagel response\n"));

```



```

255: if (!ReadResp(fd)) {
256:     DEBUGLOG(fprintf(stderr, "Failed to receive CFISTag1 response\n"));
257:     exit(1);
258: }
259: CFISTag1Ack * cfilresp = (CFISTag1Ack *) readbuf;
260: cfilresp->cmd.opcode = BB CFISTAGE2;
261: cfilresp->checksum = Checksum((CmdHeader*)cfilresp);
262: SendCmd(fd, cfilresp, sizeof(CFISTag1Ack));
263: if (!ReadResp(fd)) {
264:     DEBUGLOG(fprintf(stderr, "Failed to receive CFISTage2 response\n"));
265:     exit(1);
266: }
267: }
268:
269: void ReadSecpack(const char * FilePath, void * Buffer)
270: {
271:     FILE * fp = fopen(FilePath, "rb");
272:     if (fp == NULL) {
273:         perror(FilePath);
274:         exit(1);
275:     }
276:
277:     fseek(fp, 0x1a4L, SEEK_SET);
278:     if (fread(Buffer, 1, 0x800, fp) != 0x800) {
279:         fprintf(stderr, "Error while reading the secpack content\n");
280:         free(Buffer);
281:         exit(1);
282:     }
283:     fclose(fp);
284: }
285:
286: void SendBeginSecpack(int fd, void * Secpack)
287: {
288:     printf("Sending Begin Secpack command\n");
289:
290:     BeginSecpackReq req;
291:     req.cmd.cls = 0x2;
292:     req.cmd.opcode = BB BEGINSECPACK;
293:     req.cmd.param_len = 0x800;
294:     memcpy(&req.data, Secpack, 0x800);
295:     req.checksum = Checksum((CmdHeader*)&req);
296:     SendCmd(fd, &req, sizeof (BeginSecpackReq));
297:     // Wait for the answer
298:     DEBUGLOG(sprintf("Reading answer\n"));
299:     while (!ReadResp(fd)) ;
300: }
301:
302: void SendEndSecpack(int fd)
303: {
304:     printf("Sending End Secpack command\n");
305:
306:     EndSecpackReq req;
307:     req.cmd.cls = 0x2;
308:     req.cmd.opcode = BB ENDSECPACK;
309:     req.cmd.param_len = 0x2;
310:     req.unknown = 0;
311:     req.checksum = Checksum((CmdHeader*)&req);
312:     SendCmd(fd, &req, sizeof (EndSecpackReq));
313:     DEBUGLOG(sprintf("Reading answer\n"));
314:     ReadResp(fd);
315: }
316:
317: void SendErase(int fd, int BeginAddr, int EndAddr)
318: {

```

```

319:     printf("Sending Erase command\n");
320:
321:     EraseReq req;
322:     req.cmd.cls = 0x2;
323:     req.cmd.opcode = BB ERASE;
324:     req.cmd.param_len = 8;
325:     req.low_addr = BeginAddr;
326:     req.high_addr = EndAddr;
327:     req.checksum = Checksum((CmdHeader*)&req);
328:     SendCmd(fd, &req, sizeof (EraseReq));
329:     sleep(1); // Give it some time
330:     DEBUGLOG(sprintf("Reading answer\n"));
331:     if (!ReadResp(fd)) {
332:         fprintf(stderr, "Ooops, something was wrong while erasing\n");
333:         exit(1);
334:     }
335:
336:     printf("Waiting For Erase Completion...\n");
337:
338:     EraseAck * eraseack = (EraseAck *) readbuf;
339:     EraseStatusReq statusreq;
340:     statusreq.cmd.cls = 0x2;
341:     statusreq.cmd.opcode = BB ERASESTATUS;
342:     statusreq.cmd.param_len = 2;
343:     statusreq.unknown1 = eraseack->unknown1;
344:     statusreq.checksum = Checksum((CmdHeader*)&statusreq);
345:
346:     EraseStatusAck * erasestatusack = (EraseStatusAck *) readbuf;
347:     do {
348:         SendCmd(fd, &statusreq, sizeof(EraseStatusReq));
349:         DEBUGLOG(sprintf("Reading answer\n"));
350:         ReadResp(fd);
351:         erasestatusack = (EraseStatusAck *) readbuf;
352:     } while (erasestatusack->done != 1);
353:
354: }
355:
356: int ReadAddr(int fd, unsigned short int size)
357: {
358:     ReadReq req;
359:
360:     req.cmd.cls = 0x2;
361:     req.cmd.opcode = BB READ;
362:     req.cmd.param_len = 0x2;
363:     req.size = size;
364:     req.checksum = Checksum((CmdHeader*)&req);
365:
366:     DEBUGLOG(sprintf("\nSending read request:\n"));
367:     SendCmd(fd, &req, sizeof(ReadReq));
368:
369:     DEBUGLOG(sprintf("Receiving read response\n"));
370:     return ReadResp(fd);
371: }
372:
373: void Seek(int fd, unsigned int addr)
374: {
375:     DEBUGLOG(sprintf("Sending seek command for addr %p\n", addr));
376:     SeekReq req;
377:     req.cmd.cls = 0x2;
378:     req.cmd.opcode = BB SEEK;
379:     req.cmd.param_len = 0x4;
380:     req.addr = addr;
381:     req.checksum = Checksum((CmdHeader*)&req);
382:     SendCmd(fd, &req, sizeof (SeekReq));

```

```

383:  DEBUGLOG(sprintf("Reading answer\n"));
384:  ReadResp(fd);
385: }
386:
387: void DumpReadBufToFile(FILE * fp)
388: {
389:  ReadAck * packet = (ReadAck*)readbuf;
390:  int len = packet->cmd.param_len;
391:  unsigned char * buf = &packet->first_char;
392:  fwrite(buf, len, 1, fp);
393: }
394:
395: void Dump(int fd, FILE * fp)
396: {
397:  unsigned int addr = 0xa0000000;
398:  unsigned int nor_size = 0x4000000; // the NOR is 4M (32Mbit)
399:  unsigned int page_size = 0x800;
400:  int i = 0;
401:
402:  Seek(fd, addr);
403:  for (i = 0; i < nor_size; i += page_size) {
404:    DEBUGLOG(sprintf("Addr: %p\n", addr + i));
405:    ReadAddr(fd, page_size);
406:    DumpReadBufToFile(fp);
407:  }
408: }
409:
410: void * ReadBL(const char * FilePath, int * Size)
411: {
412:  FILE * fp = fopen(FilePath, "rb");
413:  if (fp == NULL) {
414:    perror(FilePath);
415:    exit(1);
416:  }
417:
418:  fseek(fp, 0, SEEK_END);
419:  int size = ftell(fp);
420:  fseek(fp, 0, SEEK_SET);
421:
422:  void * buffer = malloc(size);
423:
424:  if (fread(buffer, 1, size, fp) != size) {
425:    fprintf(stderr, "Error while reading the BL content\n");
426:    free(buffer);
427:    exit(1);
428:  }
429:  fclose(fp);
430:  *Size = size;
431:  return buffer;
432: }
433:
434: void * ReadFW(const char * FilePath, int Size)
435: {
436:  FILE * fp = fopen(FilePath, "rb");
437:  if (fp == NULL) {
438:    perror(FilePath);
439:    exit(1);
440:  }
441:
442:  void * buffer = malloc(Size);
443:  fseek(fp, 0x9a4L + 0x20000, SEEK_SET);
444:
445:  if (fread(buffer, 1, Size, fp) != Size) {
446:    fprintf(stderr, "Error while reading the FW content\n");

```

```

447:  free(buffer);
448:  exit(1);
449: }
450:  fclose(fp);
451:  return buffer;
452: }
453:
454:
455: void SendWriteOnePage(int fd, unsigned char * Buffer, int Size)
456: {
457:  int size_to_write = Size > 0x800 ? 0x800 : Size;
458:
459:  // Header, buffer, checksum
460:  int req_size = sizeof(CmdHeader) + size_to_write + 4;
461:  WriteReq * req = malloc(req_size);
462:
463:  req->cmd.cls = 0x2;
464:  req->cmd.opcode = BBWRITE;
465:  req->cmd.param_len = size_to_write;
466:  memset(&req->first_char, 0, size_to_write);
467:  memcpy(&req->first_char, Buffer, size_to_write);
468:  *(unsigned int *)(&req->first_char + size_to_write) = Checksum((CmdHeader*)req
);
469:
470:  SendCmd(fd, req, req_size);
471:  DEBUGLOG(sprintf("Reading answer\n"));
472:  if (!ReadResp(fd)) {
473:    free(req);
474:    fprintf(stderr, "Ooops, something was wrong while Writing\n");
475:    exit(1);
476:  }
477:  free(req);
478: }
479:
480: void SendWrite(int fd, unsigned char * Buffer, int Size, int Debug)
481: {
482:  if (Debug) printf("Sending Write command\n");
483:  int cur_size = Size;
484:  int step = Size / 20;
485:  int last_progress = 0;
486:
487:  while (cur_size > 0) {
488:    int progress = (Size - cur_size) / step;
489:    if (Debug && progress >= last_progress) {
490:      printf("%.2d%%\n", progress * 5);
491:      last_progress = progress;
492:    }
493:    SendWriteOnePage(fd, Buffer, cur_size);
494:    cur_size -= 0x800;
495:    Buffer += 0x800;
496:  }
497: }
498:
499: // In progress ;)
500: void PatchingFW(unsigned char * Buffer)
501: {
502:  printf("Patching FW\n");
503:
504:  if (Buffer[213740] != 0x04
505:      || Buffer[213741] != 0x00
506:      || Buffer[213742] != 0xa0
507:      || Buffer[213743] != 0xe1)
508:  {
509:    printf("Error in patch\n");

```

```

510:     exit(1);
511: }
512: Buffer[213740] = 0x00;
513: Buffer[213741] = 0x00;
514: Buffer[213742] = 0xa0;
515: Buffer[213743] = 0xe3;
516:
517: memset(Buffer + 0x410, 0, 3);
518: memset(Buffer + 0x800, 0, 16 * 10);
519: memset(Buffer + 0xBFC, 0, 16 * 8);
520: memset(Buffer + 0xFFC, 0, 16 * 8);
521: }
522:
523: void ValidateFW(int fd, unsigned char * Buffer)
524: {
525:     printf("Validating the write command\n");
526:     Seek(fd, 0xA0020000);
527:     ReadAddr(fd, 0x800);
528:
529:     ReadAck * packet = (ReadAck*)readbuf;
530:     unsigned char * buf = &packet->first_char;
531:
532:     if (memcmp(buf, Buffer, 0x800)) {
533:         printf("FW differences found\n");
534:     } else {
535:         printf("FW are equal!\n");
536:     }
537: }
538:
539: //void usage(char *prog)
540: //{
541: //    fprintf(stderr, "Usage: %s <fls file> [bl]\n", prog);
542: //    exit(1);
543: //}
544:
545: void usage(char *prog)
546: {
547:     fprintf(stderr, "Usage: %s <fls file> <NOR file>\n", prog);
548:     exit(1);
549: }
550:
551:
552: void credit(void)
553: {
554:     printf("iUnlock v42.PROPER -- Copyright 2007 The dev team\n\n\n\
555: s, Zappaz, Zf\n\n\n\
556:     * Leet Hax not for commercial uses\n\n\
557:     " Punishment: Monkeys coming out of your ass Bruce Almighty style.\n\n\n\
558:     );
559: }
560:
561: int main(int argc, char **argv)
562: {
563:     const char * hehe = RE;
564:     int fd;
565:
566:     credit();
567:
568:     if (argc != 3)
569:         usage(argv[0]);
570:
571:     void * secpack = malloc(0x800);

```

```

572:     ReadSecpack(argv[1], secpack);
573:
574:     void * fw = NULL;
575:     int fwsz = 0;
576:     fw = ReadBL(argv[2], &fwsz);
577:
578:     RestartBaseband();
579:     fd = InitConn(115200);
580:
581:     GetVersion(fd);
582:     CFIStagel_2(fd);
583:     SendBeginSecpack(fd, secpack);
584:     SendErase(fd, 0xA0020000, 0xA03bffff);
585:     Seek(fd, 0xA0020000 - 0x400);
586:     unsigned char foo[0x400];
587:     memset(foo, 0, 0x400);
588:     SendWrite(fd, foo, 0x400, false);
589:     SendWrite(fd, fw, fwsz, true);
590:     SendEndSecpack(fd);
591:     ValidateFW(fd, fw);
592:     printf("Completed.\nEnjoy!\n");
593:     free(fw);
594:
595:     return 0;
596: }

```

```
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: }
```

structs.h

lectures/4/src/

1/1

```
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;
```

structs1.c

lectures/4/src/

1/1

```
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:  * structs.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:     // let's save these students to disk
48:     FILE *fp = fopen("database", "w");
49:     if (fp != NULL)
50:     {
51:         for (int i = 0; i < STUDENTS; i++)
52:         {
53:             fprintf(fp, "%d\n", class[i].id);
54:             fprintf(fp, "%s\n", class[i].name);
55:             fprintf(fp, "%s\n", class[i].house);
56:         }
57:         fclose(fp);
58:     }
59:
60:     // free memory
61:     for (int i = 0; i < STUDENTS; i++)
62:     {
63:         free(class[i].name);
64:         free(class[i].house);
```

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

```
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: }
```