```
 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: }
```

```
 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: }
```

```
 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:  * 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: int
16: main(int argc, char * argv[])
17: {
18:     int i;
19:
20:     /* display mapping for uppercase letters */
21:     for (i = 65; i < 65 + 26; i++)
22:         printf("%c: %d\n", (char) i, i);
23:
24:     /* separate uppercase from lowercase */
25:     printf("\n");
26:
27:     /* display mapping for lowercase letters */
28:     for (i = 97; i < 97 + 26; i++)
29:         printf("%c: %d\n", (char) i, i);
30: }
31:
```

```
 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: int
16: main(int argc, char * argv[])
17: {
18:     int i;
19:
20:     /* display mapping for uppercase letters */
21:     for (i = 65; i < 65 + 26; i++)
22:         printf("%c  %d    %3d  %c\n", (char) i, i, i + 32, (char) (i + 32));
23: }
24:
```

```
 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: int
16: main(int argc, char * argv[])
17: {
18:     char c;
19:
20:     /* display mapping for uppercase letters */
21:     for (c = 'A'; c <= 'Z'; c = (char) ((int) c + 1))
22:         printf("%c: %d\n", c, (int) c);
23: }
```

```
 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: int
15: main(int argc, char * argv[])
16: {
17:     int i, j;
18:
19:     /* print top row of numbers */
20:     printf("\n   ");
21:     for (i = 1; i <= 10; i++)
22:         printf("%d  ", i);
23:     printf("\n");
24:
25:     /* print rows of holes, with letters in leftmost column */
26:     for (i = 0; i < 10; i++)
27:     {
28:         printf("%c  ", 'A' + i);
29:         for (j = 1; j <= 10; j++)
30:             printf("o  ");
31:         printf("\n");
32:     }
33:     printf("\n");
34: }
```

```
 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: int
17: main(int argc, char * argv[])
18: {
19:     int i, n;
20:
21:     /* ask user for number */
22:     printf("How many bottles will there be? ");
23:     n = GetInt();
24:
25:     /* exit upon invalid input */
26:     if (n < 1)
27:     {
28:         printf("Sorry, that makes no sense.\n");
29:         return 1;
30:     }
31:
32:     /* sing the annoying song */
33:     printf("\n");
34:     for (i = n; i > 0; i--)
35:     {
36:         printf("%d bottle(s) of beer on the wall,\n", i);
37:         printf("%d bottle(s) of beer,\n", i);
38:         printf("Take one down, pass it around,\n");
39:         printf("%d bottle(s) of beer on the wall.\n\n", i - 1);
40:     }
41:
42:     /* exit when song is over */
43:     printf("Wow, that’s annoying.\n");
44:     return 0;
45: }
```

```
 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: int
17: main(int argc, char * argv[])
18: {
19:     int n;
20:
21:     /* ask user for number */
22:     printf("How many bottles will there be? ");
23:     n = GetInt();
24:
25:     /* exit upon invalid input */
26:     if (n < 1)
27:     {
28:         printf("Sorry, that makes no sense.\n");
29:         return 1;
30:     }
31:
32:     /* sing the annoying song */
33:     printf("\n");
34:     while (n > 0)
35:     {
36:         printf("%d bottle(s) of beer on the wall,\n", n);
37:         printf("%d bottle(s) of beer,\n", n);
38:         printf("Take one down, pass it around,\n");
39:         printf("%d bottle(s) of beer on the wall.\n\n", n - 1);
40:         n--;
41:     }
42:
43:     /* exit when song is over */
44:     printf("Wow, that’s annoying.\n");
45:     return 0;
46: }
```

```
 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: int
16: main(int argc, char * argv[])
17: {
18:     int i, n;
19:     string s1, s2;
20:
21:     /* ask user for number */
22:     printf("How many bottles will there be? ");
23:     n = GetInt();
24:
25:     /* exit upon invalid input */
26:     if (n < 1)
27:     {
28:         printf("Sorry, that makes no sense.\n");
29:         return 1;
30:     }
31:
32:     /* sing the annoying song */
33:     printf("\n");
34:     for (i = n; i > 0; i--)
35:     {
36:         /* use proper grammar */
37:         s1 = (i == 1) ? "bottle" : "bottles";
38:         s2 = (i == 2) ? "bottle" : "bottles";
39:
40:         /* sing verses */
41:         printf("%d %s of beer on the wall,\n", i, s1);
42:         printf("%d %s of beer,\n", i, s1);
43:         printf("Take one down, pass it around,\n");
44:         printf("%d %s of beer on the wall.\n\n", i - 1, s2);
45:     }
46:
47:     /* exit when song is over */
48:     printf("Wow, that’s annoying.\n");
49:     return 0;
50: }
```

```
 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: }
```

```
 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: }
```

```
 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: }
```

```
 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: /* 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 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:     /* print scores */
35:     for (i = 0; i < 10; 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(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: }
```

```
 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: }
```

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

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

```
1: (print "hello, world")
```

```
1: <?
2:     echo "hello, world\n";
3: ?>
```

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

```
    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 prohibitied 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 | 0x20000 | 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 & 0xFFFFFFFE;
173:   term.c_cflag = (term.c_cflag & 0xFFFC6CFF) | 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, TIOCSDTR);
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(printf("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(printf("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(printf("IOServiceOpen failed\n"));
  211:     return;
  212:   }
  213:
  214:   result = IOConnectCallScalarMethod(conn, 0, 0, 0, 0, 0);
  215:   if (result == 0)
  216:     DEBUGLOG(printf("Baseband reset.\n"));
  217:   else
  218:     DEBUGLOG(printf("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(printf("Boot mode: %02X\n", ver->bootmode));
  241:   DEBUGLOG(printf("Major: %d, Minor: %d\n", ver->major, ver->minor));
  242:   DEBUGLOG(printf("Version: %s\n", ver->version));
  243: }
  244:
  245: void CFIStage1_2(int fd)
  246: {
  247:   CFIStage1Req req;
  248:   req.cmd.cls = 0x2;
  249:   req.cmd.opcode = BBCFISTAGE1;
  250:   req.cmd.param_len = 0;
  251:   req.checksum = Checksum ((CmdHeader*)&req);
  252:   DEBUGLOG(printf("Sending CFIStage1 Request\n"));
  253:   SendCmd(fd, &req, sizeof (CFIStage1Req));
  254:   DEBUGLOG(printf("Receiving CFIStage1 response\n"));
```

```
255:   if (!ReadResp(fd)) {
256:     DEBUGLOG(fprintf(stderr, "Failed to receive CFIStage1 response\n"));
257:     exit(1);
258:   }
259:   CFIStage1Ack * cfi1resp = (CFIStage1Ack *) readbuf;
260:   cfi1resp->cmd.opcode = BBCFISTAGE2;
261:   cfi1resp->checksum = Checksum((CmdHeader*)cfi1resp);
262:   SendCmd(fd, cfi1resp, sizeof(CFIStage1Ack));
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 = BBBEGINSECPACK;
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(printf("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 = BBENDSECPACK;
309:   req.cmd.param_len = 0x2;
310:   req.unknown = 0;
311:   req.checksum = Checksum((CmdHeader*)&req);
312:   SendCmd(fd, &req, sizeof (EndSecpackReq));
313:   DEBUGLOG(printf("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 = BBERASE;
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(printf("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 = BBERASESTATUS;
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(printf("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 = BBREAD;
362:   req.cmd.param_len = 0x2;
363:   req.size = size;
364:   req.checksum = Checksum((CmdHeader*)&req);
365:
366:   DEBUGLOG(printf("\nSending read request:\n"));
367:   SendCmd(fd, &req, sizeof(ReadReq));
368:
369:   DEBUGLOG(printf("Receiving read response\n"));
370:   return ReadResp(fd);
371: }
372:
373: void Seek(int fd, unsigned int addr)
374: {
375:   DEBUGLOG(printf("Sending seek command for addr %p\n", addr));
376:   SeekReq req;
377:   req.cmd.cls = 0x2;
378:   req.cmd.opcode = BBSEEK;
379:   req.cmd.param_len = 0x4;
380:   req.addr = addr;
381:   req.checksum = Checksum((CmdHeader*)&req);
382:   SendCmd(fd, &req, sizeof (SeekReq));
```

```
383:   DEBUGLOG(printf("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 = 0x400000; // 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(printf("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(printf("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\n"\
  555:          "Credits: Daeken, Darkmen, guest184, gray, iZsh, pytey, roxfan, Sam, un
s, Zappaz, Zf\n\n" \
  556:          "* Leet Hax not for commercial uses\n"\
  557:          "  Punishment: Monkeys coming out of your ass Bruce Almighty style.\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 fwsize = 0;
576:   fw = ReadBL(argv[2], &fwsize);
577:
578:   RestartBaseband();
579:   fd = InitConn(115200);
580:
581:   GetVersion(fd);
582:   CFIStage1_2(fd);
583:   SendBeginSecpack(fd, secpack);
584:   SendErase(fd, 0xA0020000, 0xA03bfffe);
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, fwsize, true);
590:   SendEndSecpack(fd);
591:   ValidateFW(fd, fw);
592:   printf("Completed.\nEnjoy!\n");
593:   free(fw);
594:
595:   return 0;
596: }
```

```
 1: /*
 2:   Copyright 2007 Daeken && iZsh
 3:
 4:   No iPhones were harm and no money wasted in the process...
 5:   Credits: Daeken, iZsh, roxfan and an Anonymous contributor
 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 prohibitied 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: #ifndef __PACKETS_H__
25: #define __PACKETS_H__
26:
27: #pragma pack(1)
28:
29: enum OPCODE
30: {
31:   BBSETBAUDRATE = 0x82,
32:   BBCFISTAGE1 = 0x84,
33:   BBCFISTAGE2 = 0x85,
34:   BB102 = 0x102,
35:   BBBEGINSECPACK = 0x204,
36:   BBENDSECPACK = 0x205,
37:   BBGETFLASHID = 0x801,
38:   BBSEEK = 0x802,
39:   BBREAD = 0x803,
40:   BBWRITE = 0x804,
41:   BBERASE = 0x805,
42:   BBERASESTATUS = 0x806,
43:   BBNOP1 = 0x807,
44:   BBWRITEPAGE = 0x808,
45:   BBNOP2 = 0x809,
46: };
47:
48: typedef struct {
49:   unsigned char opcode; // 0x0b
50:   unsigned int bootmode;
51:   unsigned int major;
52:   unsigned int minor;
53:   char version[0x33];
54: } VersionAck;
55:
56: typedef struct {
57:   unsigned short int cls;
58:   unsigned short int opcode;
59:   unsigned short int param_len;
60: } CmdHeader;
61:
62: // Baudrate
63: typedef struct {
64:   CmdHeader cmd;
```

```
  unsigned int baud;
  unsigned int checksum;
} BaudReq;

// CFIStage
typedef struct {
  CmdHeader cmd;
  unsigned int checksum;
} CFIStage1Req;

typedef struct {
  CmdHeader cmd;
  unsigned char data[0x100];
  unsigned int checksum;
} CFIStage1Ack;

// Secpack

typedef struct {
  CmdHeader cmd;
  unsigned char data[0x800];
  unsigned int checksum;
} BeginSecpackReq;

typedef struct {
  CmdHeader cmd;
  unsigned short unknown1;
  unsigned int unknown2;
  unsigned int checksum;
} BeginSecpackAck;

typedef struct {
  CmdHeader cmd;
  unsigned short unknown;
  unsigned int checksum;
} EndSecpackReq;

typedef struct {
  CmdHeader cmd;
  unsigned short unknown;
  unsigned int checksum;
} EndSecpackAck;

// FlashID
typedef struct {
  CmdHeader cmd;
  unsigned int checksum;
} GetFlashIDReq;

typedef struct {
  CmdHeader cmd;
  unsigned int null;
  unsigned int major;
  unsigned int minor;
  char string[8];
  unsigned int checksum;
} GetFlashIDAck;

// Seek
// The addr seems to be checked against a white list
typedef struct {
  CmdHeader cmd;
  unsigned int addr; // Virtual Address to map from
  unsigned int checksum;
```

```
  129: } SeekReq;
  130:
  131: typedef struct {
  132:   CmdHeader cmd;
  133:   unsigned short unkown1;
  134:   unsigned int unnown2;
  135:   unsigned int checksum;
  136: } SeekAck;
  137:
  138: // Read
  139: typedef struct {
  140:   CmdHeader cmd;
  141:   unsigned short int size;
  142:   unsigned int checksum;
  143: } ReadReq;
  144:
  145: typedef struct {
  146:   CmdHeader cmd;
  147:   unsigned char first_char;
  148: } ReadAck;
  149:
  150: // Write
  151: // Don’t forget the checksum at the end of the buffer
  152: // "Write one page" is a wrapper around Write to write one page (0x800 byte)
  153: typedef struct {
  154:   CmdHeader cmd;
  155:   unsigned char first_char;
  156: } WriteReq;
  157:
  158: typedef struct {
  159:   CmdHeader cmd;
  160:   unsigned short int unknown; // seems to be xor of the bytes written or somethi
ng close to it
  161:   unsigned int checksum;
  162: } WriteAck;
  163:
  164: // Erase
  165: // This command seems to have a backdoor:
  166: // if low_addr == 0xA03D0000 then size is hardcoded to 0x20000
  167: // else if low_addr == 0xA07D0000 then size is hardcoded to 0x22000
  168: // What is interesting is that because the NOR is 4M long:
  169: // 0xA07D0000 - 4M == 0xA03D0000
  170: // so... same address, one fold cycled, and erase an extra 0x2000
  171: // mmm... factory reset? Who want to risk his phone? (assuming you
  172: // can go around the secpack)
  173: // Also from the NOR dump, it looks like this range (0xA03D0000 -> 0xA03F2000) i
s
  174: // hidden by the read command (would need to disassemble the low level
  175: // read command to be sure)
  176: typedef struct {
  177:   CmdHeader cmd;
  178:   unsigned int low_addr;
  179:   unsigned int high_addr;
  180:   unsigned int checksum;
  181: } EraseReq;
  182:
  183: typedef struct {
  184:   CmdHeader cmd;
  185:   unsigned short unknown1;
  186:   unsigned int checksum;
  187: } EraseAck;
  188:
  189: // Erase status query
  190: // Take the EraseAck and just change the opcode
```

```
191: typedef struct {
192:   CmdHeader cmd;
193:   unsigned short int unknown1;
194:   unsigned int checksum;
195: } EraseStatusReq;
196:
197: // Operation performed when done == 1; loop until you get it
198: typedef struct {
199:   CmdHeader cmd;
200:   unsigned char done;
201:   unsigned int unknown2; // looks like a VA addr
202:   unsigned char unknown3;
203:   unsigned int checksum;
204: } EraseStatusAck;
205:
206: #endif
```

```
 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: }
```