

```
1. /*****
2.  * adder.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Adds two numbers.
8.  *
9.  * Demonstrates use of CS50's library.
10. *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14.
15. int
16. main(void)
17. {
18.     // ask user for input
19.     printf("Give me an integer: ");
20.     int x = GetInt();
21.     printf("Give me another integer: ");
22.     int y = GetInt();
23.
24.     // do the math
25.     printf("The sum of %d and %d is %d!\n", x, y, x + y);
26. }
```

```
1.  /*****
2.   * argv1.c
3.   *
4.   * Computer Science 50
5.   * David J. Malan
6.   *
7.   * Prints command-line arguments, one per line.
8.   *
9.   * Demonstrates use of argv.
10.  *****/
11.
12. #include <stdio.h>
13.
14.
15. int
16. main(int argc, char *argv[])
17. {
18.     // print arguments
19.     printf("\n");
20.     for (int i = 0; i < argc; i++)
21.         printf("%s\n", argv[i]);
22.     printf("\n");
23. }
```

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

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

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

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

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

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



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

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

```
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 b);
18.
19.
20. int
21. main(void)
22. {
23.     // ask user for number
24.     printf("How many bottles will there be? ");
25.     int n = GetInt();
26.
27.     // exit upon invalid input
28.     if (n < 1)
29.     {
30.         printf("Sorry, that makes no sense.\n");
31.         return 1;
32.     }
33.
34.     // sing the annoying song
35.     printf("\n");
36.     while (n)
37.         chorus(n--);
38.
39.     // exit when song is over
40.     printf("Wow, that's annoying.\n");
41.     return 0;
42. }
43.
44.
45. /*
46.  * Sings about specified number of bottles.
47.  */
48.
```

```
49. void
50. chorus(int b)
51. {
52.     // use proper grammar
53.     string s1 = (b == 1) ? "bottle" : "bottles";
54.     string s2 = (b == 2) ? "bottle" : "bottles";
55.
56.     // sing verses
57.     printf("%d %s of beer on the wall,\n", b, s1);
58.     printf("%d %s of beer,\n", b, s1);
59.     printf("Take one down, pass it around,\n");
60.     printf("%d %s of beer on the wall.\n\n", b - 1, s2);
61. }
```

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

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

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

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



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

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

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

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

```
1. /* http://www.ioccc.org/years.html */
2.
3. #include "stdio.h"
4. #define e 3
5. #define g (e/e)
6. #define h ((g+e)/2)
7. #define f (e-g-h)
8. #define j (e*e-g)
9. #define k (j-h)
10. #define l(x) tab2[x]/h
11. #define m(n,a) ((n&(a))==(a))
12.
13. long tab1[]={ 989L,5L,26L,0L,88319L,123L,0L,9367L };
14. int tab2[]={ 4,6,10,14,22,26,34,38,46,58,62,74,82,86 };
15.
16. main(m1,s) char *s; {
17.     int a,b,c,d,o[k],n=(int)s;
18.     if(m1==1){ char b[2*j+f-g]; main(l(h+e)+h+e,b); printf(b); }
19.     else switch(m1==h){
20.     case f:
21.         a=(b=(c=(d=g)<<g)<<g)<<g);
22.         return(m(n,a|c)|m(n,b)|m(n,a|d)|m(n,c|d));
23.     case h:
24.         for(a=f;a<j;++a)if(tab1[a]&&!(tab1[a]%((long)l(n))))return(a);
25.     case g:
26.         if(n<h)return(g);
27.         if(n<j){n-=g;c='D';o[f]=h;o[g]=f;}
28.         else{c='\r'-'b';n-=j-g;o[f]=o[g]=g;}
29.         if((b=n)>=e)for(b=g<<g;b<n;++b)o[b]=o[b-h]+o[b-g]+c;
30.         return(o[b-g]%n+k-h);
31.     default:
32.         if(m1==e) main(m1-g+e+h,s+g); else *(s+g)=f;
33.         for(*s=a=f;a<e;) *s=(s<<e)|main(h+a++,(char *)m1);
34.     }
35. }
36.
```

```
1. /*
2.
3.   iUnlock v42.PROPER -- Copyright 2007 The dev team
4.
5.   Credits: Daeken, Darkmen, guest184, gray, iZsh, pytey, roxfan, Sam, uns, Zappaz, 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 | 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 & 0xFFFFFFFF;
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, 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(sprintf("IOMasterPort failed\n"));
197.     return;
198. }
199.
200. CFMutableDictionaryRef matchingDict = IOServiceMatching("AppleBaseband");
201. io_service_t service = IOServiceGetMatchingService(kIOMasterPortDefault, matchingDict);
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(printf("Major: %d, Minor: %d\n", ver->major, ver->minor));
242.  DEBUGLOG(printf("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(printf("Sending CFIStagel Request\n"));
253.      SendCmd(fd, &req, sizeof (CFIStagelReq));
254.      DEBUGLOG(printf("Receiving CFIStagel response\n"));
255.      if (!ReadResp(fd)) {
256.          DEBUGLOG(fprintf(stderr, "Failed to receive CFIStagel response\n"));
257.          exit(1);
258.      }
259.      CFIStagelAck * cfilresp = (CFIStagelAck *) readbuf;
260.      cfilresp->cmd.opcode = BBCFISTAGE2;
261.      cfilresp->checksum = Checksum((CmdHeader*)cfilresp);
262.      SendCmd(fd, cfilresp, sizeof(CFIStagelAck));
263.      if (!ReadResp(fd)) {
264.          DEBUGLOG(fprintf(stderr, "Failed to receive CFIStagel2 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 = BBEGINSECPACK;
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.     sprintf("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(sprintf("Reading answer\n"));
314.     ReadResp(fd);
315. }
316.
317. void SendErase(int fd, int BeginAddr, int EndAddr)
318. {
319.     sprintf("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(sprintf("Reading answer\n"));
331.     if (!ReadResp(fd)) {
332.         fprintf(stderr, "Oops, something was wrong while erasing\n");
333.         exit(1);
334.     }
335.
336.     sprintf("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(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 = BBREAD;
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 = 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(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 = 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(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, "Oops, 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" \
555.           "Credits: Daeken, Darkmen, guest184, gray, iZsh, pytey, roxfan, Sam, uns, 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. CFISStage1_2(fd);  
583. SendBeginSecpack(fd, secpack);  
584. SendErase(fd, 0xA0020000, 0xA03bffffe);  
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.  * math1.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Computes a total but does nothing with it.
8.  *
9.  * Demonstrates use of variables.
10. *****/
11.
12. #include <stdio.h>
13.
14. int
15. main(void)
16. {
17.     int x = 1;
18.     int y = 2;
19.     int z = x + y;
20. }
```

```
1.  /*****
2.   * math2.c
3.   *
4.   * Computer Science 50
5.   * David J. Malan
6.   *
7.   * Computes and prints an integral total.
8.   *
9.   * Demonstrates use of a format string.
10.  *****/
11.
12.  #include <stdio.h>
13.
14.  int
15.  main(void)
16.  {
17.      int x = 1;
18.      int y = 2;
19.      int z = x + y;
20.      printf("%d", z);
21.  }
```

```
1. /*****
2.  * math3.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Computes and prints a floating-point total.
8.  *
9.  * Demonstrates loss of precision.
10. *****/
11.
12. #include <stdio.h>
13.
14. int
15. main(void)
16. {
17.     float answer = 17 / 13;
18.     printf("%.2f\n", answer);
19. }
```

```
1. /*****
2.  * math4.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Computes and prints a floating-point total.
8.  *
9.  * Demonstrates use of floating-point math.
10. *****/
11.
12. #include <stdio.h>
13.
14. int
15. main(void)
16. {
17.     float answer = 17 / 13.0;
18.     printf("%.2f\n", answer);
19. }
```

```
1. /*****
2.  * math5.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Computes and prints a floating-point total.
8.  *
9.  * Demonstrates use of casting.
10. *****/
11.
12. #include <stdio.h>
13.
14. int
15. main(void)
16. {
17.     float answer = 17 / (float) 13;
18.     printf("%.2f\n", answer);
19. }
```

```
1. /*****
2.  * noswitch.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Assesses the size of user's input.
8.  *
9.  * Demonstrates use of Boolean ANDing.
10. *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14.
15. int
16. main(void)
17. {
18.     // ask user for an integer
19.     printf("Give me an integer between 1 and 10: ");
20.     int n = GetInt();
21.
22.     // judge user's input
23.     if (n >= 1 && n <= 3)
24.         printf("You picked a small number.\n");
25.     else if (n >= 4 && n <= 6)
26.         printf("You picked a medium number.\n");
27.     else if (n >= 7 && n <= 10)
28.         printf("You picked a big number.\n");
29.     else
30.         printf("You picked an invalid number.\n");
31. }
```



```
1. /*****
2.  * positive1.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Demands that user provide a positive number.
8.  *
9.  * Demonstrates use of do-while.
10. *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14.
15. int
16. main(void)
17. {
18.     // loop until user provides a positive integer
19.     int n;
20.     do
21.     {
22.         printf("I demand that you give me a positive integer: ");
23.         n = GetInt();
24.     }
25.     while (n < 1);
26.     printf("Thanks for the %d!\n", n);
27. }
```

```
1. /*****
2.  * positive2.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Demands that user provide a positive number.
8.  *
9.  * Demonstrates use of bool.
10. *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14.
15. int
16. main(void)
17. {
18.     // loop until user provides a positive integer
19.     bool thankful = false;
20.     do
21.     {
22.         printf("I demand that you give me a positive integer: ");
23.         if (GetInt() > 0)
24.             thankful = true;
25.     }
26.     while (thankful == false);
27.     printf("Thanks for the positive integer!\n");
28. }
```

```
1. /*****
2.  * positive3.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Demands that user provide a positive number.
8.  *
9.  * Demonstrates use of !.
10. *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14.
15. int
16. main(void)
17. {
18.     // loop until user provides a positive integer
19.     bool thankful = false;
20.     do
21.     {
22.         printf("I demand that you give me a positive integer: ");
23.         if (GetInt() > 0)
24.             thankful = true;
25.     }
26.     while (!thankful);
27.     printf("Thanks for the positive integer!\n");
28. }
```

```
1. /*****
2.  * progress1.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Simulates a progress bar.
8.  *
9.  * Demonstrates sleep.
10. *****/
11.
12. #include <stdio.h>
13. #include <unistd.h>
14.
15. int
16. main(void)
17. {
18.     // simulate progress from 0% to 100%
19.     for (int i = 0; i <= 100; i++)
20.     {
21.         printf("Percent complete: %d%%\n", i);
22.         sleep(1);
23.     }
24.     printf("\n");
25. }
```

```
1. /*****
2.  * progress2.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Simulates a better progress bar.
8.  *
9.  * Demonstrates \r, fflush, and sleep.
10. *****/
11.
12. #include <stdio.h>
13. #include <unistd.h>
14.
15. int
16. main(void)
17. {
18.     // simulate progress from 0% to 100%
19.     for (int i = 0; i <= 100; i++)
20.     {
21.         printf("\rPercent complete: %d%%", i);
22.         fflush(stdout);
23.         sleep(1);
24.     }
25.     printf("\n");
26. }
```

```
1. /*****
2.  * progress3.c
3.  *
4.  * Computer Science 50
5.  * David J. Malan
6.  *
7.  * Simulates a better progress bar.
8.  *
9.  * Demonstrates a while loop.
10. *****/
11.
12. #include <stdio.h>
13. #include <unistd.h>
14.
15. int
16. main(void)
17. {
18.     int i = 0;
19.
20.     /* simulate progress from 0% to 100% */
21.     while (i <= 100)
22.     {
23.         printf("\rPercent complete: %d%%", i);
24.         fflush(stdout);
25.         sleep(1);
26.         i++;
27.     }
28.     printf("\n");
29. }
```

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

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



```
1.  /*****
2.   * sizeof.c
3.   *
4.   * Computer Science 50
5.   * David J. Malan
6.   *
7.   * Reports the sizes of C's data types.
8.   *
9.   * Demonstrates use of sizeof.
10.  *****/
11.
12. #include <stdio.h>
13.
14. int
15. main(void)
16. {
17.     // some sample variables
18.     char c;
19.     double d;
20.     float f;
21.     int i;
22.
23.     // report the sizes of variables' types
24.     printf("char: %d\n", sizeof(c));
25.     printf("double: %d\n", sizeof(d));
26.     printf("float: %d\n", sizeof(f));
27.     printf("int: %d\n", sizeof(i));
28. }
```