```c
/**
 * adder.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Adds two numbers.
 *
 * Demonstrates use of CS50's library.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // ask user for input
    printf("Give me an integer: ");
    int x = GetInt();
    printf("Give me another integer: ");
    int y = GetInt();

    // do the math
    printf("The sum of %i and %i is %i!\n", x, y, x + y);
}
```

```c
/**************************************************************************
 * buggy-0.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Should print 10 asterisks but doesn't!
 * Can you find the bug?
 **************************************************************************/

#include <stdio.h>

int main(void)
{
    for (int i = 0; i <= 10; i++)
        printf("*");
}
```

```c
/**********************************************************************
 * buggy-1.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Should print 10 asterisks, one per line, but doesn't!
 * Can you find the bug?
 **********************************************************************/

#include <stdio.h>

int main(void)
{
    for (int i = 0; i <= 10; i++)
        printf("*");
        printf("\n");
}
```

```c
/**
 * conditions-0.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Tells user if his or her input is positive or negative (somewhat
 * inaccurately).
 *
 * Demonstrates use of if-else construct.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // ask user for an integer
    printf("I'd like an integer please: ");
    int n = GetInt();

    // analyze user's input (somewhat inaccurately)
    if (n > 0)
    {
        printf("You picked a positive number!\n");
    }
    else
    {
        printf("You picked a negative number!\n");
    }
}
```

```c
/**
 * conditions-1.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Tells user if his or her input is positive or negative.
 *
 * Demonstrates use of if-else if-else construct.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // ask user for an integer
    printf("I'd like an integer please: ");
    int n = GetInt();

    // analyze user's input
    if (n > 0)
    {
        printf("You picked a positive number!\n");
    }
    else if (n == 0)
    {
        printf("You picked zero!\n");
    }
    else
    {
        printf("You picked a negative number!\n");
    }
}
```

```c
/**
 * f2c.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Converts Fahrenheit to Celsius.
 *
 * Demonstrates arithmetic.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // ask user user for temperature in Fahrenheit
    printf("Temperature in F: ");
    float f = GetFloat();

    // convert F to C
    float c = 5.0 / 9.0 * (f - 32.0);

    // display result to one decimal place
    printf("%.1f\n", c);
}
```

```c
/**
 * hello-0.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Says hello to the world.
 *
 * Demonstrates use of printf.
 */

#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

```c
/**
 * hello-1.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Says hello to just David.
 *
 * Demonstrates use of CS50's library.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string name = "David";
    printf("hello, %s\n", name);
}
```

```c
/**
 * hello-2.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Says hello to whomever.
 *
 * Demonstrates use of CS50's library and standard input.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    printf("State your name: ");
    string name = GetString();
    printf("hello, %s\n", name);
}
```

```
/*

  iUnlock v42.PROPER -- Copyright 2007 The dev team

  Credits: Daeken, Darkmen, guest184, gray, iZsh, pytey, roxfan, Sam, uns, Zappaz, Zf

  All code, information or data [from now on "data"] available
  from the "iPhone dev team" [1] or any other project linked from
  this or other pages is owned by the creator who created the data.
  The copyright, license right, distribution right and any other
  rights lies with the creator.

  It is prohibitied to use the data without the written agreement
  of the creator. This included using ideas in other projects
  (commercial or not commercial).

  Where data was created by more than 1 creator a written agreement
  from each of the creators has to be obtained.

  Punishment: Monkeys coming out of your ass Bruce Almighty style.

  [1] http://iphone.fiveforty.net/wiki/index.php?title=Main_Page
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <time.h>
#include "IOKit/IOKitLib.h"

#include "packets.h"

#define ever ;;
#define LOG stdout
#define SPEED 750000

#pragma pack(1)

#define BUFSIZE (65536+100)
unsigned char readbuf[BUFSIZE];

struct termios term;

//#define DEBUG_ENABLED 1

```

```
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. {
```

```
  int sum = 0x00030000;
  sum += packet->opcode;
  sum += packet->param_len;

  int len = packet->param_len;
  unsigned char * buf = ((unsigned char *)packet) + sizeof (CmdHeader);
  int i = 0;

  for (i = 0; i < len; ++i)
    sum += buf[i];
  return sum;
}

void SendCmd(int fd, void *buf, size_t size)
{
  DEBUGLOG(fprintf(LOG, "Sending:\n"));
  DEBUGLOG(HexDump((unsigned char*)buf, size));

  if(write(fd, buf, size) == -1) {
    fprintf(stderr, "Shit. %s\n", strerror(errno));
    exit(1);
  }
}

#define sendBytes(fd, args...) {\
  unsigned char sendbuf[] = {args}; \
  SendCmd(fd, sendbuf, sizeof(sendbuf)); \
}

int ReadResp(int fd)
{
  int len = 0;
  struct timeval timeout;
  int nfds = fd + 1;
  fd_set readfds;

  FD_ZERO(&readfds);
  FD_SET(fd, &readfds);

  // Wait a second
  timeout.tv_sec = 0;
  timeout.tv_usec = 500000;

  while (select(nfds, &readfds, NULL, NULL, &timeout) > 0)
    len += read(fd, readbuf + len, BUFSIZE - len);

  if (len > 0) {
    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, matchingDict);
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);
```

```
}

void DumpReadBufToFile(FILE * fp)
{
  ReadAck * packet = (ReadAck*)readbuf;
  int len = packet->cmd.param_len;
  unsigned char * buf = &packet->first_char;
  fwrite(buf, len, 1, fp);
}

void Dump(int fd, FILE * fp)
{
  unsigned int addr = 0xa0000000;
  unsigned int nor_size = 0x400000; // the NOR is 4M (32Mbit)
  unsigned int page_size = 0x800;
  int i = 0;

  Seek(fd, addr);
  for (i = 0; i < nor_size; i += page_size) {
    DEBUGLOG(printf("Addr: %p\n", addr + i));
    ReadAddr(fd, page_size);
    DumpReadBufToFile(fp);
  }
}

void * ReadBL(const char * FilePath, int * Size)
{
  FILE * fp = fopen(FilePath, "rb");
  if (fp == NULL) {
    perror(FilePath);
    exit(1);
  }

  fseek(fp, 0, SEEK_END);
  int size = ftell(fp);
  fseek(fp, 0, SEEK_SET);

  void * buffer = malloc(size);

  if (fread(buffer, 1, size, fp) != size) {
    fprintf(stderr, "Error while reading the BL content\n");
    free(buffer);
    exit(1);
  }
  fclose(fp);
  *Size = size;
  return buffer;
}
```

```
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)
```

```
{
  if (Debug) printf("Sending Write command\n");
  int cur_size = Size;
  int step = Size / 20;
  int last_progress = 0;

  while (cur_size > 0) {
    int progress = (Size - cur_size) / step;
    if (Debug && progress >= last_progress) {
      printf("%.2d%%\n", progress * 5);
      last_progress = progress;
    }
    SendWriteOnePage(fd, Buffer, cur_size);
    cur_size -= 0x800;
    Buffer += 0x800;
  }
}

// In progress ;)
void PatchingFW(unsigned char * Buffer)
{
  printf("Patching FW\n");

  if (Buffer[213740] != 0x04
    || Buffer[213741] != 0x00
    || Buffer[213742] != 0xa0
    || Buffer[213743] != 0xe1)
  {
    printf("Error in patch\n");
    exit(1);
  }
  Buffer[213740] = 0x00;
  Buffer[213741] = 0x00;
  Buffer[213742] = 0xa0;
  Buffer[213743] = 0xe3;

  memset(Buffer + 0x410, 0, 3);
  memset(Buffer + 0x800, 0, 16 * 10);
  memset(Buffer + 0xBFC, 0, 16 * 8);
  memset(Buffer + 0xFFC, 0, 16 * 8);
}

void ValidateFW(int fd, unsigned char * Buffer)
{
  printf("Validating the write command\n");
  Seek(fd, 0xA0020000);
  ReadAddr(fd, 0x800);

```

```
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.           "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.    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. }
```

```c
/**
 * nonswitch.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Assesses the size of user's input.
 *
 * Demonstrates use of Boolean ANDing.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // ask user for an integer
    printf("Give me an integer between 1 and 10: ");
    int n = GetInt();

    // judge user's input
    if (n >= 1 && n <= 3)
    {
        printf("You picked a small number.\n");
    }
    else if (n >= 4 && n <= 6)
    {
        printf("You picked a medium number.\n");
    }
    else if (n >= 7 && n <= 10)
    {
        printf("You picked a big number.\n");
    }
    else
    {
        printf("You picked an invalid number.\n");
    }
}
```

```c
/**
 * positive.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Demands that user provide a positive integer.
 *
 * Demonstrates use of do-while.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // loop until user provides a positive integer
    int n;
    do
    {
        printf("Please give me a positive int: ");
        n = GetInt();
    }
    while (n < 1);
    printf("Thanks for the positive int!\n", n);
}
```

```c
/**************************************************************************
 * return.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Cubes a variable.
 *
 * Demonstrates use of parameter and return value.
 **************************************************************************/

#include <stdio.h>

// function prototype
int cube(int a);

int main(void)
{
    int x = 2;
    printf("x is now %i\n", x);
    printf("Cubing...\n");
    x = cube(x);
    printf("Cubed!\n");
    printf("x is now %i\n", x);
}

/**
 * Cubes argument.
 */
int cube(int n)
{
    return n * n * n;
}
```

```c
/**
 * sizeof.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Reports the sizes of C's data types.
 *
 * Demonstrates use of sizeof.
 */

#include <stdio.h>

int main(void)
{
    // some sample variables
    char c;
    double d;
    float f;
    int i;

    // report the sizes of variables' types
    printf("char: %i\n", sizeof(c));
    printf("double: %i\n", sizeof(d));
    printf("float: %i\n", sizeof(f));
    printf("int: %i\n", sizeof(i));
}
```

```c
/**
 * switch.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Assesses the size of user's input.
 *
 * Demonstrates use of a switch.
 */

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // ask user for an integer
    printf("Give me an integer between 1 and 10: ");
    int n = GetInt();

    // judge user's input
    switch (n)
    {
        case 1:
        case 2:
        case 3:
            printf("You picked a small number.\n");
            break;

        case 4:
        case 5:
        case 6:
            printf("You picked a medium number.\n");
            break;

        case 7:
        case 8:
        case 9:
        case 10:
            printf("You picked a big number.\n");
            break;

        default:
           printf("You picked an invalid number.\n");
           break;
    }
}
```