```
/*****************************************************************************
 * bar.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Offers opportunities to play with pointers with GDB.
 *****************************************************************************/

#include <stdio.h>


int foo(int n);
void bar(int m);

int
main(void)
{
    int a;
    char * s = "hello, world";
    printf("%s\n", &s[7]);
    a = 5;
    foo(a);
    return 0;
}

int
foo(int n)
{
    int b;
    b = n;
    b *= 2;
    bar(b);
    return b;
}

void
bar(int m)
{
    printf("Hi, I'm bar!\n");
}
```



```
/*****************************************************************************
 * compare1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Tries (and fails) to compare two strings.
 *
 * Demonstrates strings as pointers to arrays.
 *****************************************************************************/

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


int
main(void)
{
    // get line of text
    printf("Say something: ");
    string s1 = GetString();

    // get another line of text
    printf("Say something: ");
    string s2 = GetString();

    // try (and fail) to compare strings
    if (s1 == s2)
        printf("You typed the same thing!\n");
    else
        printf("You typed different things!\n");
}
```

```
 1: /*************************************************************************
 2:  * compare2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Compares two strings.
 8:  *
 9:  * Demonstrates strings as pointers to arrays.
10:  *************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <string.h>
15:
16:
17: int
18: main(void)
19: {
20:     // get line of text
21:     printf("Say something: ");
22:     char *s1 = GetString();
23:
24:     // get another line of text
25:     printf("Say something: ");
26:     char *s2 = GetString();
27:
28:     // try to compare strings
29:     if (s1 != NULL && s2 != NULL)
30:     {
31:         if (strcmp(s1, s2) == 0)
32:             printf("You typed the same thing!\n");
33:         else
34:             printf("You typed different things!\n");
35:     }
36: }
```



```
 1: /*************************************************************************
 2:  * copy1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Tries and fails to copy two strings.
 8:  *
 9:  * Demonstrates strings as pointers to arrays.
10:  *************************************************************************/
11:
12: #include <cs50.h>
13: #include <ctype.h>
14: #include <stdio.h>
15: #include <stdlib.h>
16: #include <string.h>
17:
18:
19: int
20: main(void)
21: {
22:     // get line of text
23:     printf("Say something: ");
24:     char *s1 = GetString();
25:     if (s1 == NULL)
26:         return 1;
27:
28:     // try (and fail) to copy string
29:     char *s2 = s1;
30:
31:     // change "copy"
32:     printf("Capitalizing copy...\n");
33:     if (strlen(s2) > 0)
34:         s2[0] = toupper(s2[0]);
35:
36:     // print original and "copy"
37:     printf("Original: %s\n", s1);
38:     printf("Copy:     %s\n", s2);
39:
40:     // free memory
41:     free(s1);
42: }
```

```
/*****************************************************************************
 * copy2.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Copies a string.
 *
 * Demonstrates strings as pointers to arrays.
 *****************************************************************************/

#include <cs50.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


int
main(void)
{
    // get line of text
    printf("Say something: ");
    char *s1 = GetString();
    if (s1 == NULL)
        return 1;

    // allocate enough space for copy
    char *s2 = malloc((strlen(s1) + 1) * sizeof(char));
    if (s2 == NULL)
        return 1;

    // copy string
    int n = strlen(s1);
    for (int i = 0; i < n; i++)
        s2[i] = s1[i];
    s2[n] = '\0';

    // change copy
    printf("Capitalizing copy...\n");
    if (strlen(s2) > 0)
        s2[0] = toupper(s2[0]);

    // print original and copy
    printf("Original: %s\n", s1);
    printf("Copy:     %s\n", s2);

    // free memory
    free(s1);
    free(s2);
}
```



```
/*****************************************************************************
 * cs50.c
 *
 * version 1.1.6
 *
 * Computer Science 50
 * Glenn Holloway
 * David J. Malan
 *
 * Definitions for the CS50 Library.
 * Based on Eric Roberts' genlib.c and simpio.c.
 *
 * The latest version of this file can be found at
 * http://www.cs50.net/pub/releases/cs50/cs50.c.
 *
 * To compile as a static library on your own system:
 * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
 * % ar rcs libcs50.a cs50.o
 * % rm -f cs50.o
 * % cp cs50.h /usr/local/include
 * % cp libcs50.a /usr/local/lib
 *****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "cs50.h"


/*
 * Default capacity of buffer for standard input.
 */

#define CAPACITY 128


/*
 * Reads a line of text from standard input and returns the equivalent
 * char; if text does not represent a char, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  If line can't be read,
 * returns CHAR_MAX.
 */

char
GetChar(void)
{
    // try to get a char from user
    while (true)
    {
        // get line of text, returning CHAR_MAX on failure
        string line = GetString();
        if (line == NULL)
            return CHAR_MAX;

        // return a char if only a char (possibly with
        // leading and/or trailing whitespace) was provided
        char c1, c2;
        if (sscanf(line, " %c %c", &c1, &c2) == 1)
        {
            free(line);
            return c1;
        }
        else
```

```
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns the equivalent
 * double as precisely as possible; if text does not represent a
 * double, user is prompted to retry.  Leading and trailing whitespace
 * is ignored.  For simplicity, overflow and underflow are not detected.
 * If line can't be read, returns DBL_MAX.
 */

double
GetDouble(void)
{
    // try to get a double from user
    while (true)
    {
        // get line of text, returning DBL_MAX on failure
        string line = GetString();
        if (line == NULL)
            return DBL_MAX;

        // return a double if only a double (possibly with
        // leading and/or trailing whitespace) was provided
        double d; char c;
        if (sscanf(line, " %lf %c", &d, &c) == 1)
        {
            free(line);
            return d;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns the equivalent
 * float as precisely as possible; if text does not represent a float,
 * user is prompted to retry.  Leading and trailing whitespace is ignored.
 * For simplicity, overflow and underflow are not detected.  If line can't
 * be read, returns FLT_MAX.
 */

float
GetFloat(void)
{
    // try to get a float from user
    while (true)
    {
        // get line of text, returning FLT_MAX on failure
        string line = GetString();
        if (line == NULL)
            return FLT_MAX;

        // return a float if only a float (possibly with
```



```
        // leading and/or trailing whitespace) was provided
        char c; float f;
        if (sscanf(line, " %f %c", &f, &c) == 1)
        {
            free(line);
            return f;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns it as an
 * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
 * does not represent such an int, user is prompted to retry.  Leading
 * and trailing whitespace is ignored.  For simplicity, overflow is not
 * detected.  If line can't be read, returns INT_MAX.
 */

int
GetInt(void)
{
    // try to get an int from user
    while (true)
    {
        // get line of text, returning INT_MAX on failure
        string line = GetString();
        if (line == NULL)
            return INT_MAX;

        // return an int if only an int (possibly with
        // leading and/or trailing whitespace) was provided
        int n; char c;
        if (sscanf(line, " %d %c", &n, &c) == 1)
        {
            free(line);
            return n;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns an equivalent
 * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
 * does not represent such a long long, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  For simplicity, overflow
 * is not detected.  If line can't be read, returns LLONG_MAX.
 */

long long
GetLongLong(void)
{
    // try to get a long long from user
```

```
    while (true)
    {
        // get line of text, returning LLONG_MAX on failure
        string line = GetString();
        if (line == NULL)
            return LLONG_MAX;

        // return a long long if only a long long (possibly with
        // leading and/or trailing whitespace) was provided
        long long n; char c;
        if (sscanf(line, " %lld %c", &n, &c) == 1)
        {
            free(line);
            return n;
        }
        else
        {
            free(line);
            printf("Retry: ");
        }
    }
}


/*
 * Reads a line of text from standard input and returns it as a string,
 * sans trailing newline character.  (Ergo, if user inputs only "\n",
 * returns "" not NULL.)  Leading and trailing whitespace is not ignored.
 * Returns NULL upon error or no input whatsoever (i.e., just EOF).
 */

string
GetString(void)
{
    // growable buffer for chars
    string buffer = NULL;

    // capacity of buffer
    unsigned int capacity = 0;

    // number of chars actually in buffer
    unsigned int n = 0;

    // character read or EOF
    int c;

    // iteratively get chars from standard input
    while ((c = fgetc(stdin)) != '\n' && c != EOF)
    {
        // grow buffer if necessary
        if (n + 1 > capacity)
        {
            // determine new capacity: start at CAPACITY then double
            if (capacity == 0)
                capacity = CAPACITY;
            else if (capacity <= (UINT_MAX / 2))
                capacity *= 2;
            else
            {
                free(buffer);
                return NULL;
            }

            // extend buffer's capacity
```



```
            string temp = realloc(buffer, capacity * sizeof(char));
            if (temp == NULL)
            {
                free(buffer);
                return NULL;
            }
            buffer = temp;
        }

        // append current character to buffer
        buffer[n++] = c;
    }

    // return NULL if user provided no input
    if (n == 0 && c == EOF)
        return NULL;

    // minimize buffer
    string minimal = malloc((n + 1) * sizeof(char));
    strncpy(minimal, buffer, n);
    free(buffer);

    // terminate string
    minimal[n] = '\0';

    // return string
    return minimal;
}
```

```
/*****************************************************************************
 * cs50.h
 *
 * version 1.1.6
 *
 * Computer Science 50
 * Glenn Holloway
 * David J. Malan
 *
 * Declarations for the CS50 Library.
 * Based on Eric Roberts' genlib.h and simpio.h.
 *
 * The latest version of this file can be found at
 * http://www.cs50.net/pub/releases/cs50/cs50.h.
 *
 * To compile as a static library on your own system:
 * % gcc -c -ggdb -std=c99 cs50.c -o cs50.o
 * % ar rcs libcs50.a cs50.o
 * % rm -f cs50.o
 * % cp cs50.h /usr/local/include
 * % cp libcs50.a /usr/local/lib
 *****************************************************************************/

#ifndef _CS50_H
#define _CS50_H

#include <float.h>
#include <limits.h>


/*
 * Borrow the standard library's data type for Boolean variables whose
 * values must be (true|false).
 */

#include <stdbool.h>


/*
 * Our own data type for string variables.
 */

typedef char *string;


/*
 * Reads a line of text from standard input and returns the equivalent
 * char; if text does not represent a char, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  If line can't be read,
 * returns CHAR_MAX.
 */

char
GetChar(void);


/*
 * Reads a line of text from standard input and returns the equivalent
 * double as precisely as possible; if text does not represent a
 * double, user is prompted to retry.  Leading and trailing whitespace
 * is ignored.  For simplicity, overflow and underflow are not detected.
 * If line can't be read, returns DBL_MAX.
 */

```



```
double
GetDouble(void);


/*
 * Reads a line of text from standard input and returns the equivalent
 * float as precisely as possible; if text does not represent a float,
 * user is prompted to retry.  Leading and trailing whitespace is ignored.
 * For simplicity, overflow and underflow are not detected.  If line can't
 * be read, returns FLT_MAX.
 */

float
GetFloat(void);


/*
 * Reads a line of text from standard input and returns it as an
 * int in the range of [-2^31 + 1, 2^31 - 2], if possible; if text
 * does not represent such an int, user is prompted to retry.  Leading
 * and trailing whitespace is ignored.  For simplicity, overflow is not
 * detected.  If line can't be read, returns INT_MAX.
 */

int
GetInt(void);


/*
 * Reads a line of text from standard input and returns an equivalent
 * long long in the range [-2^63 + 1, 2^63 - 2], if possible; if text
 * does not represent such a long long, user is prompted to retry.
 * Leading and trailing whitespace is ignored.  For simplicity, overflow
 * is not detected.  If line can't be read, returns LLONG_MAX.
 */

long long
GetLongLong(void);


/*
 * Reads a line of text from standard input and returns it as a string,
 * sans trailing newline character.  (Ergo, if user inputs only "\n",
 * returns "" not NULL.)  Leading and trailing whitespace is not ignored.
 * Returns NULL upon error or no input whatsoever (i.e., just EOF).
 */

string GetString(void);



#endif
```

```
/*

  iUnlock v42.PROPER -- Copyright 2007 The dev team

  Credits: Daeken, Darkmen, guest184, gray, iZsh, pytey, roxfan, Sam, uns, Zappa
z, 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

#ifndef DEBUG_ENABLED
#define DEBUGLOG(x)
#else
#define DEBUGLOG(x) x
#endif

#define UINT(x) *((unsigned int *) (x))

const char * RE = "Why the hell are you reversing this app?! We said we were "\
"going to release the sources...";

void HexDumpLine(unsigned char *buf, int remainder, int offset)
{
  int i = 0;
  char c = 0;
```



```

  // Print the hex part
  fprintf(LOG, "%08x | ", offset);
  for (i = 0; i < 16; ++i) {
    if (i < remainder)
      fprintf(LOG, "%02x%s", buf[i], (i == 7) ? "  " : " ");
    else
      fprintf(LOG, "  %s", (i == 7) ? "  " : " ");
  }
  // Print the ascii part
  fprintf(LOG, " | ");
  for (i = 0; i < 16 && i < remainder; ++i) {
    c = buf[i];
    if (c >= 0x20 && c <= 0x7e)
      fprintf(LOG, "%c%s", c, (i == 7) ? " " : "");
    else
      fprintf(LOG, ".%s", (i == 7) ? " " : "");
  }

  fprintf(LOG, "\n");
}

void HexDump(unsigned char *buf, int size)
{
  int i = 0;

  for (i = 0; i < size; i += 16)
    HexDumpLine(buf + i, size - i, i);
  fprintf(LOG, "%08x\n", size);
}

int Checksum(CmdHeader * packet)
{
  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"));
    DEBUGLOG(HexDump(readbuf, len));
  }
  return len;
}

int InitConn(int speed)
{
  int fd = open("/dev/tty.baseband", O_RDWR | 0x20000 | O_NOCTTY);
  unsigned int blahnull = 0;
  unsigned int handshake = TIOCM_DTR | TIOCM_RTS | TIOCM_CTS | TIOCM_DSR;

  if(fd == -1) {
    fprintf(stderr, "%i(%s)\n", errno, strerror(errno));
    exit(1);
  }

  ioctl(fd, 0x2000740D);
  fcntl(fd, 4, 0);
  tcgetattr(fd, &term);

  ioctl(fd, 0x8004540A, &blahnull);
  cfsetspeed(&term, speed);
  cfmakeraw(&term);
  term.c_cc[VMIN] = 0;
  term.c_cc[VTIME] = 5;

  term.c_iflag = (term.c_iflag & 0xFFFFF0CD) | 5;
  term.c_oflag =  term.c_oflag & 0xFFFFFFFE;
  term.c_cflag = (term.c_cflag & 0xFFFC6CFF) | 0x3CB00;
  term.c_lflag =  term.c_lflag & 0xFFFFFA77;

  term.c_cflag = (term.c_cflag & ~CSIZE) | CS8;
  term.c_cflag &= ~PARENB;
  term.c_lflag &= ~ECHO;

  tcsetattr(fd, TCSANOW, &term);

  ioctl(fd, TIOCSDTR);
  ioctl(fd, TIOCCDTR);
  ioctl(fd, TIOCMSET, &handshake);

  return fd;
}

void RestartBaseband()
{
  kern_return_t   result;
```



```
  mach_port_t     masterPort;

  result = IOMasterPort(MACH_PORT_NULL, &masterPort);
  if (result) {
    DEBUGLOG(printf("IOMasterPort failed\n"));
    return;
  }

  CFMutableDictionaryRef matchingDict = IOServiceMatching("AppleBaseband");
  io_service_t service = IOServiceGetMatchingService(kIOMasterPortDefault, match
ingDict);
  if (!service) {
    DEBUGLOG(printf("IOServiceGetMatchingService failed\n"));
    return;
  }

  io_connect_t conn;
  result = IOServiceOpen(service, mach_task_self(), 0, &conn);
  if (result) {
    DEBUGLOG(printf("IOServiceOpen failed\n"));
    return;
  }

  result = IOConnectCallScalarMethod(conn, 0, 0, 0, 0, 0);
  if (result == 0)
    DEBUGLOG(printf("Baseband reset.\n"));
  else
    DEBUGLOG(printf("Baseband reset failed\n"));
  IOServiceClose(conn);
}

void SendGetVersion(int fd)
{
  sendBytes(fd, 0x60, 0x0D);
}

void GetVersion(int fd)
{
  SendGetVersion(fd);

  for (ever) {
    if(ReadResp(fd) != 0) {
      if(readbuf[0] == 0x0b)
        break;
    }
    SendGetVersion(fd);
  }

  VersionAck *ver = (VersionAck *) readbuf;
  DEBUGLOG(printf("Boot mode: %02X\n", ver->bootmode));
  DEBUGLOG(printf("Major: %d, Minor: %d\n", ver->major, ver->minor));
  DEBUGLOG(printf("Version: %s\n", ver->version));
}

void CFIStage1_2(int fd)
{
  CFIStage1Req req;
  req.cmd.cls = 0x2;
  req.cmd.opcode = BBCFISTAGE1;
  req.cmd.param_len = 0;
  req.checksum = Checksum ((CmdHeader*)&req);
  DEBUGLOG(printf("Sending CFIStage1 Request\n"));
  SendCmd(fd, &req, sizeof (CFIStage1Req));
  DEBUGLOG(printf("Receiving CFIStage1 response\n"));
```

```
  if (!ReadResp(fd)) {
    DEBUGLOG(fprintf(stderr, "Failed to receive CFIStage1 response\n"));
    exit(1);
  }
  CFIStage1Ack * cfi1resp = (CFIStage1Ack *) readbuf;
  cfi1resp->cmd.opcode = BBCFISTAGE2;
  cfi1resp->checksum = Checksum((CmdHeader*)cfi1resp);
  SendCmd(fd, cfi1resp, sizeof(CFIStage1Ack));
  if (!ReadResp(fd)) {
    DEBUGLOG(fprintf(stderr, "Failed to receive CFIStage2 response\n"));
    exit(1);
  }
}

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

  fseek(fp, 0x1a4L, SEEK_SET);
  if (fread(Buffer, 1, 0x800, fp) != 0x800) {
    fprintf(stderr, "Error while reading the secpack content\n");
    free(Buffer);
    exit(1);
  }
  fclose(fp);
}

void SendBeginSecpack(int fd, void * Secpack)
{
  printf("Sending Begin Secpack command\n");

  BeginSecpackReq req;
  req.cmd.cls = 0x2;
  req.cmd.opcode = BBBEGINSECPACK;
  req.cmd.param_len = 0x800;
  memcpy(&req.data, Secpack, 0x800);
  req.checksum = Checksum((CmdHeader*)&req);
  SendCmd(fd, &req, sizeof (BeginSecpackReq));
  // Wait for the answer
  DEBUGLOG(printf("Reading answer\n"));
  while (!ReadResp(fd)) ;
}

void SendEndSecpack(int fd)
{
  printf("Sending End Secpack command\n");

  EndSecpackReq req;
  req.cmd.cls = 0x2;
  req.cmd.opcode = BBENDSECPACK;
  req.cmd.param_len = 0x2;
  req.unknown = 0;
  req.checksum = Checksum((CmdHeader*)&req);
  SendCmd(fd, &req, sizeof (EndSecpackReq));
  DEBUGLOG(printf("Reading answer\n"));
  ReadResp(fd);
}

void SendErase(int fd, int BeginAddr, int EndAddr)
{
```



```
  printf("Sending Erase command\n");

  EraseReq req;
  req.cmd.cls = 0x2;
  req.cmd.opcode = BBERASE;
  req.cmd.param_len = 8;
  req.low_addr = BeginAddr;
  req.high_addr = EndAddr;
  req.checksum = Checksum((CmdHeader*)&req);
  SendCmd(fd, &req, sizeof (EraseReq));
  sleep(1); // Give it some time
  DEBUGLOG(printf("Reading answer\n"));
  if (!ReadResp(fd)) {
    fprintf(stderr, "Ooops, something was wrong while erasing\n");
    exit(1);
  }

  printf("Waiting For Erase Completion...\n");

  EraseAck * eraseack = (EraseAck *) readbuf;
  EraseStatusReq statusreq;
  statusreq.cmd.cls = 0x2;
  statusreq.cmd.opcode = BBERASESTATUS;
  statusreq.cmd.param_len = 2;
  statusreq.unknown1 = eraseack->unknown1;
  statusreq.checksum = Checksum((CmdHeader*)&statusreq);

  EraseStatusAck * erasestatusack = (EraseStatusAck *) readbuf;
  do {
    SendCmd(fd, &statusreq, sizeof(EraseStatusReq));
    DEBUGLOG(printf("Reading answer\n"));
    ReadResp(fd);
    erasestatusack = (EraseStatusAck *) readbuf;
  } while (erasestatusack->done != 1);

}

int ReadAddr(int fd, unsigned short int size)
{
  ReadReq req;

  req.cmd.cls = 0x2;
  req.cmd.opcode = BBREAD;
  req.cmd.param_len = 0x2;
  req.size = size;
  req.checksum = Checksum((CmdHeader*)&req);

  DEBUGLOG(printf("\nSending read request:\n"));
  SendCmd(fd, &req, sizeof(ReadReq));

  DEBUGLOG(printf("Receiving read response\n"));
  return ReadResp(fd);
}

void Seek(int fd, unsigned int addr)
{
  DEBUGLOG(printf("Sending seek command for addr %p\n", addr));
  SeekReq req;
  req.cmd.cls = 0x2;
  req.cmd.opcode = BBSEEK;
  req.cmd.param_len = 0x4;
  req.addr = addr;
  req.checksum = Checksum((CmdHeader*)&req);
  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");
```

```
    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);

  ReadAck * packet = (ReadAck*)readbuf;
  unsigned char * buf = &packet->first_char;

  if (memcmp(buf, Buffer, 0x800)) {
    printf("FW differences found\n");
  } else {
    printf("FW are equal!\n");
  }
}

//void usage(char *prog)
//{
//  fprintf(stderr, "Usage: %s <fls file> [bl]\n", prog);
//  exit(1);
//}

void usage(char *prog)
{
  fprintf(stderr, "Usage: %s <fls file> <NOR file>\n", prog);
  exit(1);
}


void credit(void)
{
  printf("iUnlock v42.PROPER -- Copyright 2007 The dev team\n\n\n"\
         "Credits: Daeken, Darkmen, guest184, gray, iZsh, pytey, roxfan, Sam, uns, Zappaz, Zf\n\n" \
         "* Leet Hax not for commercial uses\n"\
         "  Punishment: Monkeys coming out of your ass Bruce Almighty style.\n\n"
         );
}

int main(int argc, char **argv)
{
  const char * hehe = RE;
  int fd;

  credit();

  if (argc != 3)
    usage(argv[0]);

  void * secpack = malloc(0x800);
```



```
  ReadSecpack(argv[1], secpack);

  void * fw = NULL;
  int fwsize = 0;
  fw = ReadBL(argv[2], &fwsize);

  RestartBaseband();
  fd = InitConn(115200);

  GetVersion(fd);
  CFIStage1_2(fd);
  SendBeginSecpack(fd, secpack);
  SendErase(fd, 0xA0020000, 0xA03bfffe);
  Seek(fd, 0xA0020000 - 0x400);
  unsigned char foo[0x400];
  memset(foo, 0, 0x400);
  SendWrite(fd, foo, 0x400, false);
  SendWrite(fd, fw, fwsize, true);
  SendEndSecpack(fd);
  ValidateFW(fd, fw);
  printf("Completed.\nEnjoy!\n");
  free(fw);

  return 0;
}
```

```
 1: /*****************************************************************************
 2:  * pointers.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Prints a given string one character per line.
 8:  *
 9:  * Demonstrates pointer arithmetic.
10:  *****************************************************************************/
11:
12: #include <cs50.h>
13: #include <stdio.h>
14: #include <stdlib.h>
15: #include <string.h>
16:
17:
18: int
19: main(void)
20: {
21:     // get line of text
22:    char *s = GetString();
23:     if (s == NULL)
24:         return 1;
25:
26:     // print string, one character per line
27:     for (int i = 0, n = strlen(s); i < n; i++)
28:         printf("%c\n", *(s+i));
29:
30:     // free string
31:     free(s);
32: }
```



```
 1: /*****************************************************************************
 2:  * scanf1.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Reads a number from the user into an int.
 8:  *
 9:  * Demonstrates scanf and address-of operator.
10:  *****************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: int
16: main(void)
17: {
18:     int x;
19:     printf("Number please: ");
20:     scanf("%d", &x);
21:     printf("Thanks for the %d!\n", x);
22: }
```

```
 1: /*****************************************************************************
 2:  * scanf2.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Reads a string from the user into memory it shouldn't.
 8:  *
 9:  * Demonstrates possible attack!
10:  *****************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: int
16: main(void)
17: {
18:     char *buffer;
19:     printf("String please: ");
20:     scanf("%s", buffer);
21:     printf("Thanks for the \"%s\"!\n", buffer);
22: }
```



```
 1: /*****************************************************************************
 2:  * scanf3.c
 3:  *
 4:  * Computer Science 50
 5:  * David J. Malan
 6:  *
 7:  * Reads a string from the user into an array (dangerously).
 8:  *
 9:  * Demonstrates potential buffer overflow!
10:  *****************************************************************************/
11:
12: #include <stdio.h>
13:
14:
15: int
16: main(void)
17: {
18:     char buffer[16];
19:     printf("String please: ");
20:     scanf("%s", buffer);
21:     printf("Thanks for the \"%s\"!\n", buffer);
22: }
```

```
/*****************************************************************************
 * structs.h
 *
 * Computer Science 50
 * David J. Malan
 *
 * Defines a student for structs{1,2}.c.
 *****************************************************************************/


// structure representing a student
typedef struct
{
    int id;
    char *name;
    char *house;
}
student;
```



```
/*****************************************************************************
 * structs1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Demonstrates use of structs.
 *****************************************************************************/

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

#include "structs.h"


// class size
#define STUDENTS 3


int
main(void)
{
    // declare class
    student class[STUDENTS];

    // populate class with user's input
    for (int i = 0; i < STUDENTS; i++)
    {
        printf("Student's ID: ");
        class[i].id = GetInt();

        printf("Student's name: ");
        class[i].name = GetString();

        printf("Student's house: ");
        class[i].house = GetString();
        printf("\n");
    }

    // now print anyone in Mather
    for (int i = 0; i < STUDENTS; i++)
        if (strcmp(class[i].house, "Mather") == 0)
            printf("%s is in Mather!\n\n", class[i].name);

    // free memory
    for (int i = 0; i < STUDENTS; i++)
    {
        free(class[i].name);
        free(class[i].house);
    }
}
```

```
/*****************************************************************************
 * structs.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Demonstrates use of structs.
 *****************************************************************************/

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

#include "structs.h"


// class size
#define STUDENTS 3


int
main(void)
{
    // declare class
    student class[STUDENTS];

    // populate class with user's input
    for (int i = 0; i < STUDENTS; i++)
    {
        printf("Student's ID: ");
        class[i].id = GetInt();

        printf("Student's name: ");
        class[i].name = GetString();

        printf("Student's house: ");
        class[i].house = GetString();
        printf("\n");
    }

    // now print anyone in Mather
    for (int i = 0; i < STUDENTS; i++)
        if (strcmp(class[i].house, "Mather") == 0)
            printf("%s is in Mather!\n\n", class[i].name);

    // let's save these students to disk
    FILE *fp = fopen("database", "w");
    if (fp != NULL)
    {
        for (int i = 0; i < STUDENTS; i++)
        {
            fprintf(fp, "%d\n", class[i].id);
            fprintf(fp, "%s\n", class[i].name);
            fprintf(fp, "%s\n", class[i].house);
        }
        fclose(fp);
    }

    // free memory
    for (int i = 0; i < STUDENTS; i++)
    {
        free(class[i].name);
        free(class[i].house);
```



```
    }
}
```

```
/*****************************************************************************
 * swap.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Swaps two variables' values.
 *
 * Demonstrates passing by reference.
 *****************************************************************************/

#include <stdio.h>


// function prototype
void swap(int *a, int *b);


int
main(void)
{
    int x = 1;
    int y = 2;

    printf("x is %d\n", x);
    printf("y is %d\n", y);
    printf("Swapping...\n");
    swap(&x, &y);
    printf("Swapped!\n");
    printf("x is %d\n", x);
    printf("y is %d\n", y);
}


/*
 * Swap arguments' values.
 */

void
swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```