Kyler Little

```
/********* super.c code ***************/
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ext2fs/ext2_fs.h>
#include "constants.h"

GD    *gp;
SUPER *sp;
INODE *ip;
DIR   *dp;

#define BLKSIZE 1024

/****************** in <ext2fs/ext2_fs.h>******************************
struct ext2_super_block {
  u32  s_inodes_count;       // total number of inodes
  u32  s_blocks_count;       // total number of blocks
  u32  s_r_blocks_count;
  u32  s_free_blocks_count;  // current number of free blocks
  u32  s_free_inodes_count;  // current number of free inodes
  u32  s_first_data_block;   // first data block in this group
  u32  s_log_block_size;     // 0 for 1KB block size
  u32  s_log_frag_size;
  u32  s_blocks_per_group;   // 8192 blocks per group
  u32  s_frags_per_group;
  u32  s_inodes_per_group;
  u32  s_mtime;
  u32  s_wtime;
  u16  s_mnt_count;          // number of times mounted
  u16  s_max_mnt_count;      // mount limit
  u16  s_magic;              // 0xEF53
  // A FEW MORE non-essential fields
};
**********************************************************************/

char buf[BLKSIZE];
int fd;

int get_block(int fd, int blk, char buf[ ])
{
    lseek(fd, (long)blk*BLKSIZE, 0);
    read(fd, buf, BLKSIZE);
}

int super()
{
    // read SUPER block
    get_block(fd, 1, buf);
    sp = (SUPER *)buf;

    // check for EXT2 magic number:
    printf("s_magic = %x\n", sp->s_magic);
    if (sp->s_magic != 0xEF53) {
        printf("NOT an EXT2 FS\n");
        exit(1);
  }

  printf("EXT2 FS OK\n");

  printf("s_inodes_count = %d\n", sp->s_inodes_count);
  printf("s_blocks_count = %d\n", sp->s_blocks_count);

  printf("s_free_inodes_count = %d\n", sp->s_free_inodes_count);
  printf("s_free_blocks_count = %d\n", sp->s_free_blocks_count);
  printf("s_first_data_blcok = %d\n", sp->s_first_data_block);

  printf("s_log_block_size = %d\n", sp->s_log_block_size);
  printf("s_blocks_per_group = %d\n", sp->s_blocks_per_group);
  printf("s_inodes_per_group = %d\n", sp->s_inodes_per_group);

  printf("s_mnt_count = %d\n", sp->s_mnt_count);
  printf("s_max_mnt_count = %d\n", sp->s_max_mnt_count);

  printf("s_magic = %x\n", sp->s_magic);

  printf("s_mtime = %s", ctime(&sp->s_mtime));
  printf("s_wtime = %s", ctime(&sp->s_wtime));
}

char *disk = "../mydisk";

int main(int argc, char *argv[ ])
{
    if (argc > 1)
    disk = argv[1];
    fd = open(disk, O_RDONLY);
    if (fd < 0){
        printf("open failed\n");
        exit(1);
    }
    super();
}
```

super.C output:

```
s_magic = ef53
EXT2 FS OK
s_inodes_count = 184
s_blocks_count = 1440
s_free_inodes_count = 155
s_free_blocks_count = 1389
s_first_data_blcok = 1
s_log_block_size = 0
s_blocks_per_group = 8192
s_inodes_per_group = 184
s_mnt_count = 1
s_max_mnt_count = -1
s_magic = ef53
s_mtime = Wed Oct 24 20:20:58 2018
s_wtime = Wed Oct 24 20:20:58 2018
```

```
/********* gd.c code ***************/
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ext2fs/ext2_fs.h>
#include "constants.h"

GD    *gp;
SUPER *sp;
INODE *ip;
DIR   *dp;

#define BLKSIZE 1024

/****************** in <ext2fs/ext2_fs.h>******************************
struct ext2_group_desc
{
        u32     bg_block_bitmap;      // Blocks bitmap block
        u32     bg_inode_bitmap;      // Inodes bitmap block
        u32     bg_inode_table;               // Inodes table block
        u16     bg_free_blocks_count;         // Free blocks count
        u16     bg_free_inodes_count;         // Free inodes count
        u16     bg_used_dirs_count;   // Directories count
        u16     bg_flags;
        u32     bg_exclude_bitmap_lo;         // Exclude bitmap for snapshots
        u16     bg_block_bitmap_csum_lo;// crc32c(s_uuid+grp_num+bitmap) LSB
        u16     bg_inode_bitmap_csum_lo;// crc32c(s_uuid+grp_num+bitmap) LSB
        u16     bg_itable_unused;     // Unused inodes count
        u16     bg_checksum;                  // crc16(s_uuid+group_num+group_desc)
};
**********************************************************************/

char buf[BLKSIZE];
int fd;

int get_block(int fd, int blk, char buf[ ])
{
   lseek(fd, (long)blk*BLKSIZE, 0);
   read(fd, buf, BLKSIZE);
}

int gd()
{
   // read GROUP DESCRIPTOR block
   get_block(fd, 2, buf);
   gp = (GD *)buf;

   if (!gp) {
      printf("No GROUP DESCRIPTOR block!\n");
      exit(1);
   }
   printf("EXT2 FS OK\n");

   printf("bg_block_bitmap = %d\n", gp->bg_block_bitmap);
   printf("bg_inode_bitmap = %d\n", gp->bg_inode_bitmap);

   printf("bg_inode_table = %d\n", gp->bg_inode_table);

   printf("bg_free_inodes_count = %d\n", gp->bg_free_inodes_count);
   printf("bg_free_blocks_count = %d\n", gp->bg_free_blocks_count);
   printf("bg_used_dirs_count = %d\n", gp->bg_used_dirs_count);

   printf("bg_flags = %d\n", gp->bg_flags);
   printf("bg_exclude_bitmap_lo = %d\n", gp->bg_exclude_bitmap_lo);
   printf("bg_inode_bitmap_csum_lo = %d\n", gp->bg_inode_bitmap_csum_lo);
   printf("bg_block_bitmap_csum_lo = %d\n", gp->bg_block_bitmap_csum_lo);

   printf("bg_itable_unused = %d\n", gp->bg_itable_unused);
   printf("bg_checksum = %d\n", gp->bg_checksum);
}

char *disk = "../mydisk";

int main(int argc, char *argv[ ])
{
    if (argc > 1)
    disk = argv[1];
    fd = open(disk, O_RDONLY);
    if (fd < 0){
        printf("open failed\n");
        exit(1);
    }
    gd();
}
```

gd.c output:

```
EXT2 FS OK
bg_block_bitmap = 8
bg_inode_bitmap = 9
bg_inode_table = 10
bg_free_inodes_count = 155
bg_free_blocks_count = 1389
bg_used_dirs_count = 6
bg_flags = 4
bg_exclude_bitmap_lo = 0
bg_inode_bitmap_csum_lo = 0
bg_block_bitmap_csum_lo = 0
bg_itable_unused = 0
bg_checksum = 0
```

```
//imap

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ext2fs/ext2_fs.h>

// define shorter TYPES, save typing efforts
typedef struct ext2_group_desc  GD;
typedef struct ext2_super_block SUPER;
typedef struct ext2_inode       INODE;
typedef struct ext2_dir_entry_2 DIR;    // need this for new version of e2fs

#define BLKSIZE 1024

GD    *gp;
SUPER *sp;
INODE *ip;
DIR   *dp;

char buf[BLKSIZE];
int fd;

int get_block(int fd, int blk, char buf[ ])
{
  lseek(fd, (long)blk*BLKSIZE, 0);
   read(fd, buf, BLKSIZE);
}

int tst_bit(char *buf, int bit)
{
  int i, j;
  i = bit / 8;  j = bit % 8;
  if (buf[i] & (1 << j))
     return 1;
  return 0;
}

int imap()
{
  char buf[BLKSIZE];
  int  imap, ninodes;
  int  i;

  // read SUPER block
  get_block(fd, 1, buf);
  sp = (SUPER *)buf;

  ninodes = sp->s_inodes_count;
  printf("ninodes = %d\n", ninodes);

  // read Group Descriptor 0
  get_block(fd, 2, buf);
  gp = (GD *)buf;

  imap = gp->bg_inode_bitmap;
  printf("imap = %d\n", imap);

  // read inode_bitmap block
  get_block(fd, imap, buf);

  for (i=0; i < ninodes; i++){
    (tst_bit(buf, i)) ?     putchar('1') : putchar('0');
    if (i && (i % 8)==0)
       printf(" ");
  }
  printf("\n");
}
```

```
char *disk = "mydisk";

int main(int argc, char *argv[ ])
{
  if (argc > 1)
    disk = argv[1];

  fd = open(disk, O_RDONLY);
  if (fd < 0){
    printf("open %s failed\n", disk);
    exit(1);
  }

  imap();
}
```

imap.c output:

```
ninodes = 184
imap = 9
111111111 11111111 11111110 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000000
```

```
//bmap



#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <ext2fs/ext2_fs.h>


typedef struct ext2_group_desc  GD;
typedef struct ext2_super_block SUPER;
typedef struct ext2_inode       INODE;
typedef struct ext2_dir_entry_2 DIR;

#define BLKSIZE 1024

GD    *gp;
SUPER *sp;
INODE *ip;
DIR   *dp;

char buf[BLKSIZE];
int fd;

int get_block(int fd, int blk, char buf[])
{
    lseek(fd, (long)blk*BLKSIZE,0);
    read(fd, buf, BLKSIZE);
}

int tst_bit(char *buf, int bit)
{
  int i, j;
  i = bit / 8;  j = bit % 8;
  if (buf[i] & (1 << j))
     return 1;
  return 0;
}

int bmap()
{
    int bmap, i, nblocks;
    get_block(fd,1,buf);
    sp=(SUPER *)buf;

    nblocks=sp->s_blocks_count;
    printf("nblocks = %d\n", nblocks);

    get_block(fd,2,buf);
    gp=(GD *)buf;

    bmap=gp->bg_block_bitmap;
    printf("bmap = %d\n", bmap);

    get_block(fd,bmap,buf);

    for (i=0; i < nblocks; i++){
    putchar(tst_bit(buf,i)+48);
    if(i && (i%8)==0)
        putchar(' ');
    if(i && (i%32) ==0)
        putchar('\n');
  }
  putchar('\n');
}

char *disk = "mydisk";
```

```
int main(int argc, char *argv[ ])
{
  if (argc > 1)
    disk = argv[1];

  fd = open(disk, O_RDONLY);
  if (fd < 0){
    printf("open %s failed\n", disk);
    exit(1);
  }

  bmap();
}
```

bmap.c output:

```
nblocks = 1440
bmap = 8
111111111 11111111 11111111 11111111
11111111 11111111 11110000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 0001101
```

```
/********* inode.c: print information in / INODE (INODE #2) *********/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ext2fs/ext2_fs.h>
#include "constants.h"

GD    *gp;
SUPER *sp;
INODE *ip;
DIR   *dp;

int fd;
int iblock;

int get_block(int fd, int blk, char buf[ ])
{
    lseek(fd,(long)blk*BLKSIZE, 0);
    read(fd, buf, BLKSIZE);
}

int inode()
{
    char buf[BLKSIZE];

    // read GD
    get_block(fd, 2, buf);
    gp = (GD *)buf;
    /****************
     printf("%8d %8d %8d %8d %8d %8d\n",
        gp->bg_block_bitmap,
        gp->bg_inode_bitmap,
        gp->bg_inode_table,
        gp->bg_free_blocks_count,
        gp->bg_free_inodes_count,
        gp->bg_used_dirs_count);
    ****************/
    iblock = gp->bg_inode_table;   // get inode start block#
    printf("inode_block=%d\n", iblock);

    // get inode start block
    get_block(fd, iblock, buf);

    ip = (INODE *)buf + 1;          // ip points at 2nd INODE

    printf("mode=0x%4x\n", ip->i_mode);
    printf("uid=%d  gid=%d\n", ip->i_uid, ip->i_gid);
    printf("size=%d\n", ip->i_size);
    printf("time=%s", ctime(&ip->i_ctime));
    printf("link=%d\n", ip->i_links_count);
    printf("i_block[0]=%d\n", ip->i_block[0]);

    /*****************************
     u16  i_mode;         // same as st_imode in stat() syscall
    u16  i_uid;                    // ownerID
    u32  i_size;                   // file size in bytes
    u32  i_atime;                    // time fields
    u32  i_ctime;
    u32  i_mtime;
    u32  i_dtime;
    u16  i_gid;                    // groupID
    u16  i_links_count;                // link count
    u32  i_blocks;                   // IGNORE
    u32  i_flags;                    // IGNORE
    u32  i_reserved1;                 // IGNORE
    u32  i_block[15];                 // IMPORTANT, but later
    ***************************/
}
```

```
char *disk = "../mydisk";
int main(int argc, char *argv[])
{
    if (argc > 1)
        disk = argv[1];

    fd = open(disk, O_RDONLY);
    if (fd < 0){
        printf("open %s failed\n", disk);
        exit(1);
    }

    inode();
}
```

inode.c output:

```
inode_block=10
mode=0x41ed
uid=0  gid=0
size=1024
time=Wed Oct 24 20:20:58 2018
link=7
i_block[0]=33
```

```
/********* dir.c: print all entries under '/' directory *********/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ext2fs/ext2_fs.h>
#include <string.h>
#include "constants.h"

GD    *gp;
SUPER *sp;
INODE *ip;
DIR   *dp;

int fd;
int iblock;

int search(INODE *ip, char *name) {
    int i = 0;
    char dbuf[BLKSIZE], *cp;

    for (i = 0; i < NUM_DIRECT_BLKS; i++) {
        if (ip->i_block[i] == 0) return 0;   // NOT FOUND

        // Otherwise, search for name in dir; read direct block into dbuf
        get_block(fd, ip->i_block[i], dbuf);

        dp = (DIR *)dbuf;
        cp = dbuf;

        while (cp < dbuf + BLKSIZE) {
            // if name matches with dir's name, return inode number
            if (!strncmp(dp->name, name, dp->name_len)) {
                return dp->inode;
            }
            cp += dp->rec_len;
            dp = (DIR *)cp;
        }
    }
}

int get_block(int fd, int blk, char buf[ ])
{
    lseek(fd,(long)blk*BLKSIZE, 0);
    read(fd, buf, BLKSIZE);
}

int dir()
{
    char buf[BLKSIZE], dbuf[BLKSIZE], *cp, temp[MAX_FILENAME_LEN];

    // read GD
    get_block(fd, 2, buf);
    gp = (GD *)buf;

    iblock = gp->bg_inode_table;   // get inode start block#
    printf("inode_block = %d\n", iblock);

    // get inode start block
    get_block(fd, iblock, buf);

    // get root inode #2
    ip = (INODE *)buf + 1;         // ip points at 2nd INODE

    // int ino = search(ip, "dir2");
    // if (ino) printf("ino: %d\n", ino);

    // ip = (INODE *)buf + 1;         // ip points at 2nd INODE

    int i;
    for (i = 0; i<NUM_DIRECT_BLKS; i++) {
        if (ip->i_block[i] == 0) {
            break;
        }
        // Note: ip->i_block[0-11] will yield a pointer to a direct block
        printf("i_block[%d] = %d\n", i, ip->i_block[i]);

        // Read direct block into dbuf
        get_block(fd, ip->i_block[i], dbuf);
        printf(" ino      rec_len    name_len    name\n");

        dp = (DIR *)dbuf;
        cp = dbuf;

        while (cp < dbuf + BLKSIZE) {
            strncpy(temp, dp->name, dp->name_len);
            temp[dp->name_len] = 0;

            printf("%4d    %6d      %6d       %s\n", dp->inode, dp->rec_len, dp->name_len, temp);
            cp += dp->rec_len;
            dp = (DIR *)cp;
        }
    }
}

char *disk = "mydisk";
int main(int argc, char *argv[])
{
    if (argc > 1)
        disk = argv[1];

    fd = open(disk, O_RDONLY);
    if (fd < 0){
        printf("open %s failed\n", disk);
        exit(1);
    }

    dir();
}
```

dir.c output:

```
inode_block = 10
i_block[0] = 33
 ino      rec_len    name_len    name
   2        12          1       .
   2        12          2       ..
  11        20         10       lost+found
  12        12          4       dir1
  13        12          4       dir2
  14        12          4       dir3
  15        12          4       dir4
  16        16          5       file1
  17        16          5       file2
  18        16          5       file3
  19       884          5       file4
```

```
//ialloc

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ext2fs/ext2_fs.h>

// define shorter TYPES, save typing efforts
typedef struct ext2_group_desc  GD;
typedef struct ext2_super_block SUPER;
typedef struct ext2_inode       INODE;
typedef struct ext2_dir_entry_2 DIR;    // need this for new version of e2fs

#define BLKSIZE 1024

GD    *gp;
SUPER *sp;
INODE *ip;
DIR   *dp;

/********** globals *************/
int fd;
int imap, bmap;  // IMAP and BMAP block number
int ninodes, nblocks, nfreeInodes, nfreeBlocks;

int get_block(int fd, int blk, char buf[ ])
{
  lseek(fd, (long)blk*BLKSIZE, 0);
  read(fd, buf, BLKSIZE);
}

int put_block(int fd, int blk, char buf[ ])
{
  lseek(fd, (long)blk*BLKSIZE, 0);
  write(fd, buf, BLKSIZE);
}

int tst_bit(char *buf, int bit)
{
  int i, j;
  i = bit/8; j=bit%8;
  if (buf[i] & (1 << j))
     return 1;
  return 0;
}

int set_bit(char *buf, int bit)
{
  int i, j;
  i = bit/8; j=bit%8;
  buf[i] |= (1 << j);
}

int clr_bit(char *buf, int bit)
{
  int i, j;
  i = bit/8; j=bit%8;
  buf[i] &= ~(1 << j);
}

int decFreeInodes(int dev)
{
  char buf[BLKSIZE];

  // dec free inodes count in SUPER and GD
  get_block(dev, 1, buf);
  sp = (SUPER *)buf;
  sp->s_free_inodes_count--;
  put_block(dev, 1, buf);
  get_block(dev, 2, buf);
  gp = (GD *)buf;
  gp->bg_free_inodes_count--;
  put_block(dev, 2, buf);
}

int ialloc(int dev)
{
  int  i;
  char buf[BLKSIZE];

  // read inode_bitmap block
  get_block(dev, imap, buf);

  for (i=0; i < ninodes; i++){
    if (tst_bit(buf, i)==0){
       set_bit(buf,i);
       decFreeInodes(dev);

       put_block(dev, imap, buf);

       return i+1;
    }
  }
  printf("ialloc(): no more free inodes\n");
  return 0;
}

char *disk = "mydisk";

int main(int argc, char *argv[ ])
{
  int i, ino;
  char buf[BLKSIZE];

  if (argc > 1)
    disk = argv[1];

  fd = open(disk, O_RDWR);
  if (fd < 0){
    printf("open %s failed\n", disk);
    exit(1);
  }

  // read SUPER block
  get_block(fd, 1, buf);
  sp = (SUPER *)buf;

  ninodes = sp->s_inodes_count;
  nblocks = sp->s_blocks_count;
  nfreeInodes = sp->s_free_inodes_count;
  nfreeBlocks = sp->s_free_blocks_count;
  printf("ninodes=%d nblocks=%d nfreeInodes=%d nfreeBlocks=%d\n",
                 ninodes, nblocks, nfreeInodes, nfreeBlocks);

  // read Group Descriptor 0
  get_block(fd, 2, buf);
  gp = (GD *)buf;

  imap = gp->bg_inode_bitmap;
  printf("imap = %d\n", imap);
  getchar();

  for (i=0; i < 5; i++){
    ino = ialloc(fd);
    printf("allocated ino = %d\n", ino);
  }
}
```

ialloc.c output:

```
ninodes=184 nblocks=1440 nfreeInodes=155 nfreeBlocks=1389
imap = 9
```

```
//balloc

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <ext2fs/ext2_fs.h>

// define shorter TYPES, save typing efforts
typedef struct ext2_group_desc  GD;
typedef struct ext2_super_block SUPER;
typedef struct ext2_inode       INODE;
typedef struct ext2_dir_entry_2 DIR;    // need this for new version of e2fs

#define BLKSIZE 1024

GD    *gp;
SUPER *sp;
INODE *ip;
DIR   *dp;

/********** globals *************/
int fd;
int imap, bmap;  // IMAP and BMAP block number
int ninodes, nblocks, nfreeInodes, nfreeBlocks;

int get_block(int fd, int blk, char buf[ ])
{
  lseek(fd, (long)blk*BLKSIZE, 0);
  read(fd, buf, BLKSIZE);
}

int put_block(int fd, int blk, char buf[ ])
{
  lseek(fd, (long)blk*BLKSIZE, 0);
  write(fd, buf, BLKSIZE);
}

int tst_bit(char *buf, int bit)
{
  int i, j;
  i = bit/8; j=bit%8;
  if (buf[i] & (1 << j))
     return 1;
  return 0;
}

int set_bit(char *buf, int bit)
{
  int i, j;
  i = bit/8; j=bit%8;
  buf[i] |= (1 << j);
}

int clr_bit(char *buf, int bit)
{
  int i, j;
  i = bit/8; j=bit%8;
  buf[i] &= ~(1 << j);
}

int decFreeInodes(int dev)
{
  char buf[BLKSIZE];

  // dec free inodes count in SUPER and GD
  get_block(dev, 1, buf);
  sp = (SUPER *)buf;
  sp->s_free_inodes_count--;
  put_block(dev, 1, buf);
  get_block(dev, 2, buf);
  gp = (GD *)buf;
  gp->bg_free_inodes_count--;
  put_block(dev, 2, buf);
}

int balloc(int dev)
{
  int  i;
  char buf[BLKSIZE];

  // read inode_bitmap block
  get_block(dev, bmap, buf);

  for (i=0; i < nblocks; i++){
    if (tst_bit(buf, i)==0){
       set_bit(buf,i);
       decFreeInodes(dev);

       put_block(dev, bmap, buf);

       return i+1;
    }
  }
  printf("balloc(): no more free blocks\n");
  return 0;
}

char *disk = "mydisk";

int main(int argc, char *argv[ ])
{
  int i, bno;
  char buf[BLKSIZE];

  if (argc > 1)
    disk = argv[1];

  fd = open(disk, O_RDWR);
  if (fd < 0){
    printf("open %s failed\n", disk);
    exit(1);
  }

  // read SUPER block
  get_block(fd, 1, buf);
  sp = (SUPER *)buf;

  ninodes = sp->s_inodes_count;
  nblocks = sp->s_blocks_count;
  nfreeInodes = sp->s_free_inodes_count;
  nfreeBlocks = sp->s_free_blocks_count;
  printf("ninodes=%d nblocks=%d nfreeInodes=%d nfreeBlocks=%d\n",
                 ninodes, nblocks, nfreeInodes, nfreeBlocks);

  // read Group Descriptor 0
  get_block(fd, 2, buf);
  gp = (GD *)buf;

  bmap = gp->bg_block_bitmap;
  printf("bmap = %d\n", bmap);
  getchar();

  for (i=0; i < 5; i++){
    bno = balloc(fd);
    printf("allocated bno = %d\n", bno);
  }
}
```

balloc.c output:

```
ninodes=184 nblocks=1440 nfreeInodes=155 nfreeBlocks=1389
bmap = 8
```

Header file that contains all necessary functions

```
#ifndef CONSTANTS
#define CONSTANTS

/* Debugging Mode */
#define DEBUG_MODE 0

/* IO */
#define BLKSIZE 1024

/* Define shorter TYPES, save typing efforts */
typedef struct ext2_group_desc  GD;
typedef struct ext2_super_block SUPER;
typedef struct ext2_inode       INODE;
typedef struct ext2_dir_entry_2 DIR;    // need this for new version of e2fs

/* inode Constants */
#define NUM_DIRECT_BLKS 12

/* Macro -- String Lengths */
#define MAX_FILENAME_LEN 256

#endif
```