FAT12

Yonatan Erez – 208987073

It was by far the hardest homework of the course, reading through the fat12 specs and understand where to jump was not easy at all!

Please consider adding some extra time for this homework on the next semesters.

name_parsing.h

Includes the following functions:

```
void split_stored(unsigned char name_ext[11], unsigned char name[8], unsigned char ext[3]);

void stored2username(unsigned char name[8], unsigned char ext[3], unsigned char user_name[13]);

void username2stored(unsigned char user_name[13], unsigned char name[8], unsigned char ext[3]);

bool isEqual(unsigned char name[8], unsigned char ext[3], unsigned char user_name[13]);

unsigned char ToUpperCase(unsigned char x);
```

The function are utilities of the main flow.

name_parsing.c

Includes the implementation of the header file functions.

```
void split_stored(unsigned char name_ext[11], unsigned char name[8], unsigned char ext[3]){
    // splits stored filename into name and extension
```

```
void stored2username(unsigned char name[8], unsigned char ext[3], unsigned char user_name[13]){
    // transform name, extension into a filename
```

```
unsigned char ToUpperCase(unsigned char x)
{   // returns uppercase(x) of x if it is between  a,z, O.W returns x
```

```
int find_dot(unsigned char user_name[13]){
    // returns the index of the first dot, if it does not exist returns -1
```

```
void username2stored(unsigned char user_name[13], unsigned char name[8], unsigned char ext[3]){
    // transform filename into name, extension
```

```
bool isEqual(unsigned char name[8], unsigned char ext[3], unsigned char user_name[13]){
    // returns True iff name, ext is equal to filename
```

msdos_fs.h:

Was not change, includes crucial structures to read entries and sectors.

main.c

Includes the main flow of the code and some additional functions for each part of it.

```
int count_digits(char* str){
    // return the digits amount in str
```

```
void print_size(int size){
    // prints the size of the file with , on each 3 digits
```

```
void print_entry(struct msdos_dir_entry* entry) {
    // prints the entry in the windows dir format
```

```
bool is_valid_entry(struct msdos_dir_entry* entry)
{
    // return True iff entry is valid
```

```
struct msdos_dir_entry** read_root_directory(FILE* fp, int RootDirStart)
{
    // reads the root dir and returns a list o the root directory entries
```

```
int get_first_not_null_index(struct msdos_dir_entry** entries){
    // returns the first index of entry which is not NULL
```

```
int find_min(struct msdos_dir_entry** entries){
    // returns the index of the minimum (lex sorted) entry name
```

```
void print_root_dir(struct msdos_dir_entry** entries) {
    // goes over the root dir and prints each entry
```

```
int find(struct msdos_dir_entry** entries, char* filename){
    // returns the index of entry named filename, if it does not exitst returns -1
```

```
void extract_from_root_dir(FILE* fp, struct fat_boot_sector bs, struct msdos_dir_entry** entries, char*
filename, char* dest){
    // extract the content of file named filename from the root directory into a file named dest
```

```
int main(int argc, char *argv[]) {
    // runs the whole sequece
```

I decided to read the root directory and then execute the required command.

```
int FirstRootDirSecNum = bs.reserved + (bs.fats * bs.fat_length);
int RootDirStart = FirstRootDirSecNum * SECTOR_SIZE;
struct msdos_dir_entry** entries = read_root_directory(fp, RootDirStart);
```

Using the data in the boot sector it can be either found or calculated manually that the root directory begins at the 19th sector of the disk, which means:

$$\text{RootDirStart} = 0x2600.$$

The struct entries will be field with all the entries (out of 16*14 possibilities), if they are empty of invalid, the proper index will be filled with NULL.

When the struct is ready and contains all the root directory entries, we can start executing the required command.

If the command is dir, the function print_root_dir will be executed.
The function counts the number of entries and on each iteration finds the minimal (lexicographically sorted) filename.
The minimal will be printed and then freed from the entries list so it will not be printed on the next iteration.

If the command is cp, the function extract_from_root_dir will be executed.
The function uses the find function which returns the index of the filename exists, and if it doesn't exist find will return -1 and the File does not exist message will be printed. Otherwise, the filename exists and then the function will extract its content into the given destination file.
As instructed, if the filename is a directory, the File doesn't exist message will be printed.

After executing the required command, the main flow releases the allocated entries memory and exist with code 0 (SUCCESS!).