Introduction

We were asked to implement the ls command that lists the content of a directory in the terminal.
The variants we were asked to implement include: with directory argument and without, with the specific flags -n,-s,-l, -i and without.
We were also asked to implement out own filter and comparison functions that are required for the scandir function that was supposed to be used to implement the solution.
Our implementation was required to hide the hidden files that begin with the character ".";

Design

-Functional design considerations and design decisions

-stage1.c&h

The function basicDir in the stage1.h file is the main function for the stage1 process. It follows the example in the C programming manual and prints the content of a directory, if it is not hidden and was stored in the dirent pointer array by the scandir function. It also entails a function that returns an error to stderr if the scandir function failed.

-stage2.c&h

The main method is split into two parts by an if/else statement. When the arguments entered by a user only require stage 1 to be executed it calls stage 1, else it calls moves on calling stage2. Stage 2 has 4 global boolean variables that are representing each compulsory flag. I found this to be the best way as it "cheap" to check boolean values and to store them/change them. Due to that is made it easier to handle the random order of flags entered by the user. The variable pathIndex has been declared to store the index at which the directory, hence the path has been entered by the user though the terminal(index in argv). I have set it to -1 , because if no directory has been given by the user, it won't interfere with the iteration in storeFlags.

-abcsort

This function returns an integer, that is interpreted as a boolean. It gets this value by passing the file names of both dirent structs to the strcasecmp

function.

-filter

This filter is supposed to be passed on as an argument to the scandir function. It returns an integer, that is interpreted as a boolean and returns false or 0 when an entry starts with the character '.' .

-sizesort

This sorting function will be passed on onto scandir, making sure the entires are sorted correctly when an -S flag is present in the arguments. I create a char array rather than a pointer as it was required to give the pointer a certain size and mallocing is not necessary when we know that the maximum number of characters in a file name is 255. I then create stat struct variables and compare their sizes and return a negative value if the second value is bigger, else it returns a positive value.

-The design/structure of the code

There has been inserted a main method for both stage 1 and 2 in order to test both executables individually and independently. The function prototypes are mostly contained in the header, and where the function is used in more than one file , the function has been declared and defined(implemented) in the header file.

Some necessary library files have been included in the header files and some in the main c file, depending on how many functions are using the library. I also chose to separate the components of the output, using tabs. I found that tabs keep the spacing between different components consistent and hence easier to look at.

The stat struct pointer contArray stores the all the stat structs that store the information about each individual entry in a directory. I found it to be easier, as I can just iterate through the loop to print information about each entry instead of creating an output on the fly.

Testing

a. Fulfilment of Stage1

To test stage 1 you can either run stage1..

```
bash-4.3$ ./stage1 ls
example_folder
Makefile
Makefile~
stage1
stage1.c
stage1.h
stage2
stage2.c
stage2.h
bash-4.3$ ./stage1 ls example_folder
a.out
report.odt
bash-4.3$
```

..or run stage 2

```
bash-4.3$ ./stage2 ls
-------------------------------------------------
filename: example_folder

-------------------------------------------------
filename: Makefile

-------------------------------------------------
filename: Makefile~

-------------------------------------------------
filename: stage1

-------------------------------------------------
filename: stage1.c

-------------------------------------------------
filename: stage1.h

-------------------------------------------------
filename: stage2

-------------------------------------------------
filename: stage2.c

-------------------------------------------------
filename: stage2.h
```

```
bash-4.3$ ./stage2 ls example_folder
a.out
report.odt
DONE
```

In both it can be seen that it produces a line by line listing of the contents of the directory in an ascending alphabetical order.

b. fulfilment of Stage 2

1. Flag without directory

```
bash-4.3$ ./stage2 ls -l
-------------------------------------------------
drwxr-xr-x      5 bytes Wed Apr 13 19:20:07 2016
username:cak8   groupname:students      filename: example_folder

-------------------------------------------------
-rw-r--r--      157 bytes       Wed Apr 13 18:06:40 2016
username:cak8   groupname:students      filename: Makefile

-------------------------------------------------
-rw-r--r--      157 bytes       Wed Apr 13 16:18:22 2016
username:cak8   groupname:students      filename: Makefile~

-------------------------------------------------
-rwxr-xr-x      11272 bytes     Wed Apr 13 17:36:05 2016
username:cak8   groupname:students      filename: stage1

-------------------------------------------------
```

2. Flag with directory

```
bash-4.3$ ./stage2 ls -l example_folder
-------------------------------------------------
-rwxr-xr-x      21632 bytes     Wed Apr 13 16:02:15 2016
username:cak8   groupname:students      filename: a.out

-------------------------------------------------
-rw-r--r--      30381 bytes     Wed Apr 13 19:20:08 2016
username:cak8   groupname:students      filename: report.odt

DONE
```

3. Multiple flags with directory

```
bash-4.3$ ./stage2 ls -l example_folder -i -n
-----------------------------------------------
149430  -rwxr-xr-x      21632 bytes     Wed Apr 13 16:02:15 2016
userId:18285    groupId:10030   filename: a.out

-----------------------------------------------
76658   -rw-r--r--      30381 bytes     Wed Apr 13 19:20:08 2016
userId:18285    groupId:10030   filename: report.odt

DONE
```

4. Multiple flags without directory

```
bash-4.3$ ./stage2 ls -l -i -n
-----------------------------------------------
149461  drwxr-xr-x      5 bytes Wed Apr 13 19:37:15 2016
userId:18285    groupId:10030   filename: example_folder

-----------------------------------------------
149465  -rw-r--r--      157 bytes       Wed Apr 13 18:06:40 2016
userId:18285    groupId:10030   filename: Makefile

-----------------------------------------------
91399   -rw-r--r--      157 bytes       Wed Apr 13 16:18:22 2016
userId:18285    groupId:10030   filename: Makefile~

-----------------------------------------------
149459  -rwxr-xr-x      11272 bytes     Wed Apr 13 17:36:05 2016
userId:18285    groupId:10030   filename: stage1

-----------------------------------------------
```

5. Flags with varying cases(uppercase, lowercase)

```
bash-4.3$ ./stage2 ls -L -n -s
-------------------------------------------------
-rwxr-xr-x      20143 bytes     Wed Apr 13 19:30:31 2016
userId:18285    groupId:10030   filename: stage2

-------------------------------------------------
-rwxr-xr-x      11272 bytes     Wed Apr 13 17:36:05 2016
userId:18285    groupId:10030   filename: stage1

-------------------------------------------------
-rw-r--r--      5845 bytes      Wed Apr 13 19:32:55 2016
userId:18285    groupId:10030   filename: stage2.c

-------------------------------------------------
-rw-r--r--      1598 bytes      Wed Apr 13 19:32:26 2016
userId:18285    groupId:10030   filename: stage1.h

-------------------------------------------------
-rw-r--r--      597 bytes       Wed Apr 13 17:35:42 2016
userId:18285    groupId:10030   filename: stage2.h
```

Evaluation and Conclusion

My program satisfies all the requirements of stage 1 and 2. It allows to give multiple flags as arguments in a way that allows varied orders of inputs. Hence arguments could come in the order: flag, directory, flag or flag flag directory. As this is the final practical of the module cs2002, I can say that I have understood why C is necessary for low level manipulation in programming (such as system programming) and hence why it is so fragile.

How to run the program

-Stage1 can be run by giving only the arguments ls followed by the directory
example: "./stage1 ls ../abc/d"
-Stage2 can be run giving the ls argument followed by any of the required flags, hence -s,-n,-l,-i and the directory
example: "./stage2 ls -n -i ../abc/d "