CS 392, Systems Programming: Assignment 2

1. **Objective**
   In this assignment, you will create a C program to sort files of strings, ints, or doubles. Any file given to you will contain only one of the three possible types. The files will be sorted using quicksort with lomuto partitioning. This assignment is intended to cover pointers, pointer conversion, file reading, type conversion, function pointers, and makefiles.

2. **Problem**
   This assignment will require you to submit a single zip file with the following contents:
   - sort.c (the source file that contains the main function)
   - quicksort.h (the header file for the quicksort library we are creating)
   - quicksort.c (the corresponding source file with the implementation of those functions contained within the header
   - makefile (the directions for building the project)

   Download the template file to get started. Do not change any of the function headers when you are coding the implementation.

   Function Pointers
   This assignment involves the use of function pointers that will be passed to the quicksort function to tell it what function to use to compare the elements of the list.

   To support your understanding of function pointers, please read up on function pointers in section 5.11 of K&R as well as here: https://www.geeksforgeeks.org/function-pointer-in-c/

   Command Line Arguments
   This program will be called by using `./sort [-i|-d] filename`
   - The -i flag will specify that the file to be sorted contains integers.
   - The -d flag will specify that the file to be sorted contains doubles.
   - No flag will mean that the file will be sorted as strings.
   - `filename` should be a path to a .txt file where each line is an element to be sorted.

   You must include <getopt.h> in sort.c and use getopt to parse the command line arguments. It works the same way as getopt in bash. This site cover getopt thoroughly: https://azrael.digipen.edu/~mmead/www/mg/getopt/index.html

   Error Checking
   Case 1: No input arguments -- print usage message and return EXIT_FAILURE.

```
$ ./sort
Usage: ./sort [-i|-d] filename
   -i: Specifies the file contains ints.
   -d: Specifies the file contains doubles.
   filename: The file to sort.
   No flags defaults to sorting strings.
```

Case 2: Invalid Flag -- print usage message with error at top and return EXIT_FAILURE.

```
$ ./sort -z
Error: Unknown option '-z' received.
Usage: ./sort [-i|-d] filename
   -i: Specifies the file contains ints.
   -d: Specifies the file contains doubles.
   filename: The file to sort.
   No flags defaults to sorting strings.
```

Case 3: Invalid filename -- print error message and use sterror (man 3 strerror) to provide details after the first period. Return EXIT_FAILURE.

```
$ ./sort -d notfound.txt
Error: Cannot open 'notfound.txt'. No such file or directory.
```

Case 4: No filename -- print error message and return EXIT_FAILURE.

```
$ ./sort -i
Error: No input file specified.
```

Case 5: Multiple filenames -- print error message and return EXIT_FAILURE.

```
$ ./sort possible.txt ohno.txt
Error: Too many files specified.
```

Case 6: Multiple Valid Flags -- print error message and return EXIT_FAILURE.

```
$ ./sort –id ints.txt
Error: Too many flags specified.
```

Case 7: Multiple Flags with Invalid Flag -- print error message for invalid flag and return EXIT_FAILURE.

```
$ ./sort –iz ints.txt
Error: Unknown option '-z' received.
Usage: ./sort [-i|-d] filename
   -i: Specifies the file contains ints.
   -d: Specifies the file contains doubles.
   filename: The file to sort.
   No flags defaults to sorting strings.
```

The contents of the input files will be correct. You may safely assume that each line contains a string, int, or double, and that every line in the file will be of the same data type. There will be at most 1024 lines in a file, and each line will contain up to 64 printable (visible) characters. Every line in the file, including the last line, will end in a newline character.

When the user supplies the correct arguments, the program should sort the supplied data. Seen below, the tinyints file contains integers 1 through 4 in random order, and when sorted with this program, the output is the integers 1 through 4 in non-decreasing order, each on a separate line.

```
$ ./sort -i input/tinyints.txt
1
2
3
4
```

Here is an example of the program sorting strings:

```
$ ./sort input/strings.txt
are
biscuits
club
hello
in
my
world
zebras
```

Doubles should be printed with default precision (6 places to the right of the decimal point).

```
$ ./sort -d input/doubles.txt
1.000000
1.000025
1.001000
3.789500
5.167500
12.444444
600.170000
```

Of course, if the user tries to sort strings with the -i or -d flag, the output is not expected to be correct. The same is true for any mismatch between the flag specified and actual input data.

3. Tips

a) You must use getopt to parse command line arguments.
b) You'll need to become comfortable with void pointers and pointer arithmetic.
c) Use gdb to help find segfaults and valgrind to check for memory leaks.
d) Start this assignment early. The concept is straightforward, but the details might require some time and effort.