CS-392 Systems Programming Spring 2022

Assignment 2

Deadline: see Canvas

1 Objective

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

2 Problem

This assignment will require you to submit a single zip file with the following contents:

(1) `sort.c` (the source file that contains the main function);

(2) `mergesort.c` (the corresponding source file with the implementation of those functions contained within the header);

(3) a makefile.

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 merge sort function to specify how to compare the elements of the list.

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;
- `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. See this site for more details on getopt:

https://azrael.digipen.edu/~mmead/www/mg/getopt/index.html.

Error Checking For command-line arguments, you can use the following cases to test:

(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.
```

(2) Invalid Flag – print usage message with an error message specifying the illegal option (z in the example) at the 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.
```

(3) Invalid filename – print error message and use `stderr` to provide details. Return `EXIT_FAILURE`.

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

(4) No filename – print error message and return `EXIT_FAILURE`.

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

(5) Multiple filenames – print error message and return `EXIT_FAILURE`.

```
$ ./sort -i file.txt anotherfile.txt
Error: Too many files specified.
```

(6) Multiple Valid Flags – print error message and return `EXIT_FAILURE`.

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

```
$ ./sort -iiiiii ints.txt
Error: Too many flags specified.
```

(7) Multiple Flags including Invalid Flag – print error message for invalid flag, usage 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.
```

Note:

- Using flag `-abc` is the same to using `-a -b -c`, so your program should produce the same result using either way;
- The precedence of error types is (from high to low): unknown option, too many flags, too many files. When multiple errors present in the command-line argument, always print corresponding message for the highest error type, and exit with `EXIT_FAILURE` immediately.

3 Files To Complete

3.1 Header File (1 in total)

You're provided with a header file, `mergesort.h`, which contains the following function decalarations:

```
/* Functions to compare a and b */
int int_cmp(const void* a, const void* b);
int dbl_cmp(const void* a, const void* b);

/* Main sorting algorithm you need to implement */
void mergesort(void*  array,
               size_t len,
               size_t elem_sz,
               int (*comp)(const void*, const void*));
```

where `array` is an array pointed by `void*`, because it needs to be able to handle both double and integer arrays. `len` is the length of the array. `elem_sz` is the size of each

element in the array, *i.e.,* how many bytes does each element take. Lastly, `*comp` is a function pointer that can point to either `int_cmp()` or `dbl_cmp()`, depending on the type of data (speficied by the command-line flags).

For the two comparison functions, return 0 if they are equal; return 1 if first number is greater; otherwise return -1.

Do not change this file under any circumstances.

3.2 Implementation Files (3 in total)

(1) `mergesort.c`: Based on the declarations in the header file, you're going to write a `.c` file called `mergesort.c`. In this file, you'll need to include the header file, and implement the functions declared there. Note that you can certainly create more functions here if you need, such as a helper function. The only thing you're not allowed to do, is to create any function that takes a specific type of array. See example below:

```
1 void helper_double(double* left, double* right);  /* Not Allowed! */
2 void helper_int(int* left, double* right);        /* Not Allowed! */
3 void helper(void* left, void* right);             /* Perfect! */
```

Hint: If you wonder, how does the function know the type of the array? The answer is: it doesn't really need to know. Think about how data are stored as binary pattern, and why we need an argument like `elem_sz`.

(2) `sort.c`: In this file, you'll parse the command-line arguments, and include the header file (do not include the source code file). You'll read a file, get all the lines stored in an array, and call `mergesort()` to sort the array. Once it's done, print the array to the screen, one number per line. Make sure there's no memory leak.

(3) `Makefile`: Lastly, write a makefile to compile all your files. You can worry about the makefile at the end. Provide all source code files as inputs to the compiler before spending time on the makefile.

4 Good to Know

(1) The contents of the input files will be correct. You may safely assume that each line contains a human-readable 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 end in a newline character. See `test.txt` for an example;

(2) If you cannot use function pointers or void pointers successfully, submit a program without these features for partial credit (at most 50%). In this case, make it work for double precision inputs and print an appropriate error message if int is selected as the type.

5 Requirements

You must:

(1) have proper comments;

(2) include the header file `mergesort.h` in your implementation;

(3) name all your filenames **exactly** as described above;

(4) use `getopt` to parse command-line arguments;

(5) use dynamic allocation with `malloc()` and `free()`;

(6) use `FILE*` to process files not redirections;

You must NOT:

(1) include any `.c` in your code, such as `#include "mergesort.c"`;

(2) have memory leak;

(3) declare any function that takes specific type of arrays (see description above);

(4) change the file `mergesort.h`.

6 What to Submit

Zip only `Makefile`, `sort.c`, and `mergesort.c`. Do not include the header file.