Assignment 1: File Splitter

Due date: *Friday, October 25, 2019 at 22:00*

> *This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own.*

In a file called `splitter.c` write a C program that splits one large file into multiple smaller files. The program also re-assembles (merges) the split files back into the original large file. This has many use-cases, like when you exceed the maximum file size when sending an e-mail.

The program runs in two modes: *split* and *merge*. In split mode, the program splits the original file into smaller files, and also generates an index file that will be used for merging. In merge mode, the program reads the index file, and merges the smaller files into the larger file.

Split Mode. In split mode, the user provides two arguments: *filename* is the name of the original file; *max-size* is the maximum file size. The program splits the file *filename* into multiple parts, each with a maximum size of *max-size* bytes. The last part may be smaller than *max-size*. The program writes each part to a file named *filename*`.part`N, that is, the original file name, followed by the extension `.part`N, where N is a counter starting from 1. The program also creates an index file *filename*`.index`. The index file contains a list of the split parts, as well as a checksum for each part, as described below.

Merge Mode. In merge mode, the user provides the program with two arguments: the index file *index-filename*, and the path to write the merged file, *merged-filename*. The program reads the index file, verifies the checksum for each part, and merges all the parts, saving them to a new file *merged-filename*.

Index File Format. The index file lists all of the file parts. On each line is the path to the file part, followed by a white space, and then the checksum in hexadecimal. For example:

video.mp4.index

```
video.mp4.part1 0a
video.mp4.part2 12
video.mp4.part3 e8
video.mp4.part4 51
```

Checksum. The program computes the checksum of each part as the exclusive-or (XOR) of all the bytes in the part. The checksum is therefore an 8-bit number (i.e., a *char,* or better yet, an *unsigned char*) and is represented in hexadecimal. For example, if the part has the following 5 bytes:

foo.txt.part4

```
hello
```

the resulting checksum would be `0x68 ^ 0x65 ^ 0x6c ^ 0x6c ^ 0x6f` $=$ `0x62`. And the index would contain:

foo.txt.index

```
...
foo.txt.part4 62
...
```

Error Handling. If the user provides invalid arguments, if the file is invalid (e.g., it does not exist or it is not readable), or if there is other invalid input, the program exits with a non-zero return code. Furthermore, in merge mode, if the checksum of a part does not match the checksum in the index file, the program prints the number of the part, followed by a newline (exactly that, and no other output). If there are no errors, the program exits with return code 0.

Program Usage Your program must take the following command-line arguments. The first argument is the mode, either `split` or `merge`. If the mode is `split`, then the following command-line arguments are the original file name *filename* and the maximum part size *max-size*. If the mode is `merge`, then the following arguments are the name of the index file *index-filename* and the name of the output file *merged-filename*. Therefore, the usage is as follows:

`splitter split` *filename max-size*

`splitter merge` *index-filename merged-filename*

Assumptions and limits You can assume that file names do not contain spaces. You can also assume that a file can be split into at most 1024 parts.

Example

```
$ ./splitter split ciao.mp4 5120000
$ cat ciao.mp4.index
ciao.mp4.part1 98
ciao.mp4.part2 2a
ciao.mp4.part3 45
ciao.mp4.part4 f1
$ ./splitter merge ciao.mp4.index merged.mp4
$ md5sum ciao.mp4 merged.mp4
cb7dd4b18e23c9e21b055b50c4f4caec  ciao.mp4
cb7dd4b18e23c9e21b055b50c4f4caec  merged.mp4
```

Requirements

To grade your implementation, we will run automated tests. To ensure you receive a full grade, your program must behave exactly as specified here. Below is a check-list of the most important requirements:

- *clean compilation:* your code must compile without warnings (`-Wall`).
- *usage and arguments:* your program must use the exact command-line arguments specified above.
- *index file:* your program must generate an index file exactly as specified above.
- *correctness:* your program must correctly merge an identical copy of the original file.

Submission Instructions

Submit one source file named `splitter.c`. Add comments to your code to explain sections of the code that might not be clear. You may also add comments at the beginning of the source file to refer to any and all external sources of information you may have used, including code, suggestions, and comments from other students. If your implementation has limitations and errors you are aware of and were unable to fix, list those as well in the initial comments.

You may use an integrated development environment (IDE) of your choice. However, *do not submit any IDE-specific file,* such as project description files. Also, *make absolutely sure that the file you submit can be compiled and tested with a simple invocation of the standard C compiler.*

Submit your solution through the iCorsi system.