**Assignment 3: myChannels (80 marks)**

**Notes**

1. Failure to follow the instructions may lead to failed test cases and/or a final grade of 0
2. You can do this assignment individually or in a team of two
   1. If you are doing it in a group, only one submission per group is required
3. You may submit multiple times before the deadline
   1. Grade penalties will be imposed for late submissions (see course outline)
4. Plan before coding
5. For this assignment, you can use:

* [standard C libraries](https://en.cppreference.com/w/c/header)
* POSIX APIs
  + POSIX is a standardized operating systems interface based on UNIX
  + You can find a list [here](https://en.wikibooks.org/wiki/C_Programming/POSIX_Reference)

1. All the code in this lab must be done using C **only**
2. Use function-level and inline comments throughout your code
   1. We will not be specifically grading documentation
   2. However, remember that you will not be able to comment on your code unless it is sufficiently documented
   3. Take the time to document your code as you develop it properly
3. Check FAQ before posting questions on Piazza
4. We will carefully analyze the code submitted to look for plagiarism signs, do not do it
5. If you are unsure about what is allowed, please talk to an instructor or a TA

# **Rules**

* Follow the file name as specified in the instructions
* **makefile**
  + Makefile provides the following functionality
  + You will receive 0 if your Makefile fails
* **all**
  + Compile your program (this is the default behavior)
  + Producing an executable file named the same as the C file
* **clean**
  + Deletes the executable file and any intermediate files
  + .o files specifically
* Check your build to ensure that there are no errors

**Goals**

* To understand multi-threaded programming by following
  + To create a dynamic number of threads and correctly distribute tasks among them
  + To write to a shared variable from multiple threads
  + To coordinate progress of threads by checkpointing and locks
* To design and implement buffered file reading operations

**Task**

Develop a multi-threaded program that takes 16-bit integer input from *multiple* channels and formulates a *final output* by adding samples from all channels with *optional* low-pass and/or amplification values.

* Create the input/s channel file with a list of samples, one in each line.
* Each sample should be a 16-bit integer greater than 0.
* To find the mixed output, add the corresponding samples from all the channels.
  + The final output for a given sample is rounded up to the next integer value.
* Each channel may have an **optional** low-pass filter value (alpha)
  + and/or an amplification value (beta)
* The final output for each sample is collected and printed into an output file.

**Low-pass filter value (alpha)**

* **Constraints**
  + It is a float value between 0.0 and 1.0 .
  + The default alpha value is 1.0.
* **Formula**
  + new\_sample\_value = alpha \* sample\_value + (1 - alpha) \* previous\_sample\_value
* **Example**
  + If the samples are 1, 2, 3, 4 and alpha = 0.5
  + new\_sample\_values: 1, 1.5, 2.25, 3.125
  + For the first entry,
    - we used the sample without any modification.
  + For the second entry,
    - new\_value = 0.5 \* 2 + (1 – 0.5) \* 1 = 1 + 0.5 = 1.5
  + For the third entry,
    - new\_value = 0.5 \* 3 + (1 – 0.5) \* 1.5 = 1.5 + 0.75 = 2.25

**Amplification value (beta)**

* **Constraints**
  + It is a float value > 0.0
  + The default beta value is 1.0
* **Formula**
  + It is the multiplicative factor used for a given sample
  + new\_sample\_value = beta \* sample\_value
* **Example**
  + If the samples are 1, 2, 3, 4 and beta = 0.5
  + new\_sample\_values: 0.5, 1, 1.5, 2

**Important**

1. The executable file name should be **myChannels**
2. The alpha computation is first performed on the sample before amplification.
3. The final sample value should be rounded up to an integer value.
4. If integer overflow, the final output value is 65535 (the maximum 16-bit integer value)

**Arguments**

|  |  |
| --- | --- |
| **buffer\_size** | Size of the buffer (in bytes) when reading a file (int) |
| **num\_threads** | Number of threads (int) |
| **metadata\_file\_path** | Absolute Path of the metadata file (char\*) |
| **lock\_config** | Choose a single global lock or granular locks |
| **global\_checkpointing** | Flag used for determining if global checkpointing should be used  int value can be either 0 or 1 |
| **output\_file\_path** | Path of the output file (char\*) |

**Info/Constraints for each argument**

**1. Buffer size**

* Size of buffer in bytes
* You can use binary files or text files
* For binary files, make sure that there are no extra issues related to a partial reading

**2. Number of threads**

* Assume that the number of input files = p \* num\_threads
* Each thread will work with exactly p files, where p is some integer value
  + Example 1:
    - Input directory contains files: f1, f2, f3, f4, f5, f6
    - num\_threads = 2
    - Thread 1 works on files f1, f3, f5
    - Thread 2 works on files f2, f4, f6
  + Example 2:
    - The input directory contains files: f1, f2
    - num\_threads = 2
    - Thread 1 works on file f1
    - Thread 2 works on file f2
* Each thread can only read "k" bytes from a file at a time
  + The entire file should not be read/stored in memory

**3. Metadata file**

* The metadata file contains information about the channel files used as input
* A sample metadata file format is shown below
  + number\_of\_input\_files

channel\_file1\_path

channel\_file1\_low\_pass\_filter\_value

channel\_file1\_amplification

channel\_file2\_path

channel\_file2\_low\_pass\_filter\_value

channel\_file2\_amplification

channel\_file3\_path

channel\_file3\_low\_pass\_filter\_value

channel\_file3\_amplification

* **Example metadata file**
  + 3

input\_files/1.txt

1.0

0.8

input\_files/2.txt

1.0

1.0

input\_files/3.txt

1.0

2.8

* **Channel file**
  + Each Channel file has a list of samples - one on each line
  + Each sample is a 16-bit integer and >0
  + A sample Channel file format is shown below:
    - 23

414

23

45

12451

**4. Lock configurations**

* If lock\_config = 1, a single global lock should be used for accessing the output channel
* If lock\_config = 2, use a different lock for each entry in the output channel
* If lock\_config = 3, use compare\_and\_swap to update the entries in the output channel

**5. Global checkpointing**

* Global checkpointing enforces uniform processing of all files across threads
* If global\_checkpointing = 1, a thread can read the next “k” bytes from a file only after all the threads have finished processing the “k” bytes in their respective files.

**6. Local checkpointing**

* If a single thread works on p files (say, file f1, f2, f3, … fp)
  + then it must adhere to the local checkpointing rules
* If global\_checkpointing = 0,
  + The thread should read only “k” bytes from each file at a time
  + The thread can read the next “k” bytes from the file only after it has finished reading “k” bytes from all the files

Example: Consider thread T1 needs to work on files f1, f2 and f3

* Say, thread T1 has finished processing the first 100 bytes from f1
* It can read the next 100 bytes from f1 only after it has finished processing the first 100 bytes from f2 and f3
* Similarly, when the thread finishes the second 100 bytes from a file, it should process the second 100 bytes for all its files before moving to the third 100 bytes and so on

**7. Output file path:** Path of the output file to store all computations from the channels

**Note:** When implementing your solution, please follow the provided test cases. There are **NO ERRORS** in the computation of the given examples. The test cases specifically emphasize the buffer reading process. In this context, only the \n character should be treated as a byte, while \r should be ignored. The approach you choose for handling this depends on your design. One possible approach is to read the input file and consider only \n as a byte, disregarding \r.

**Example 1**

**Buffer size** = 2

**Input file 1**

1

2

3

**Input file 2**

10

20

30

**Metadata file**

2

/inputs/files/file\_1.txt

1

1

/inputs/files/file\_2.txt

0.5

0.5

**Output**

6

5

5

9

**The above calculation is as follows**

* For input file 1
  + 1\n -> 1
  + 2\n -> 2
  + 3\n -> 3
  + new\_sample\_value = alpha \* sample\_value + (1 - alpha) \* previous\_sample\_value
  + value | alpha
    - 1 | 1
    - 2 | 1\*2 + (1-1) \* 1 (previous sample value) = 2
    - 3 | 1\*3 + (1-1) \* 2 (previous sample value) = 3
  + Amplification value (beta)
    - It is the multiplicative factor used for a given sample
    - new\_sample\_value = beta \* sample\_value
      * 1 (new sample value) | 1 \* 1 = 1
      * 2 (new sample value) | 1 \* 2 = 2
      * 3 (new sample value) | 1 \* 3 = 3
* For input file 2:
  + 10\n -> 10 -> 10
  + 20\n -> \n2 -> 2
  + 30\n -> 0\n -> 0

-> 30 -> 30

-> \n

* + new\_sample\_value = alpha \* sample\_value + (1 - alpha) \* previous\_sample\_value
  + value | alpha
    - 10 | 10
    - 2 | 0.5 \* 2 + (1-0.5) \* 10 (previous sample value) = 1+5 =6
    - 0 | 0.5 \* 0 + (1-0.5) \* 6 (previous sample value) = 0+ 3 = 3
    - 30 |0.5 \* 30 + (1-0.5) \* 3 (previous sample value) = 15 + 1.5 = 16.5
  + Amplification value (beta)
    - It is the multiplicative factor used for a given sample
    - new\_sample\_value = beta \* sample\_value
      * 10 (new sample value) | 0.5 \* 10 = 5
      * 6 (new sample value) | 0.5 \*6 = 3
      * 3 (new sample value) | 0.5 \* 3 = 1.5
      * 16.5 (new sample value) | 0.5 \* 16.5 = 8.25
* The final output will be
  + 1 + 5 = 6
  + 2 + 3 = 5
  + 3 + 1.5 = 4.5 ~ 5
  + 0 + 8.25 = 9

**Example 2**

**Buffer size** **=** 3

**Input file 1**

23

4

10

**Input file 2**

130

1256

137

**Metadata file**

2

/inputs/files/file\_1.txt

0.5

1

/inputs/files/file\_2.txt

0.5

0.5

**Output**

88

68

48

51

**The above calculation is as follows**

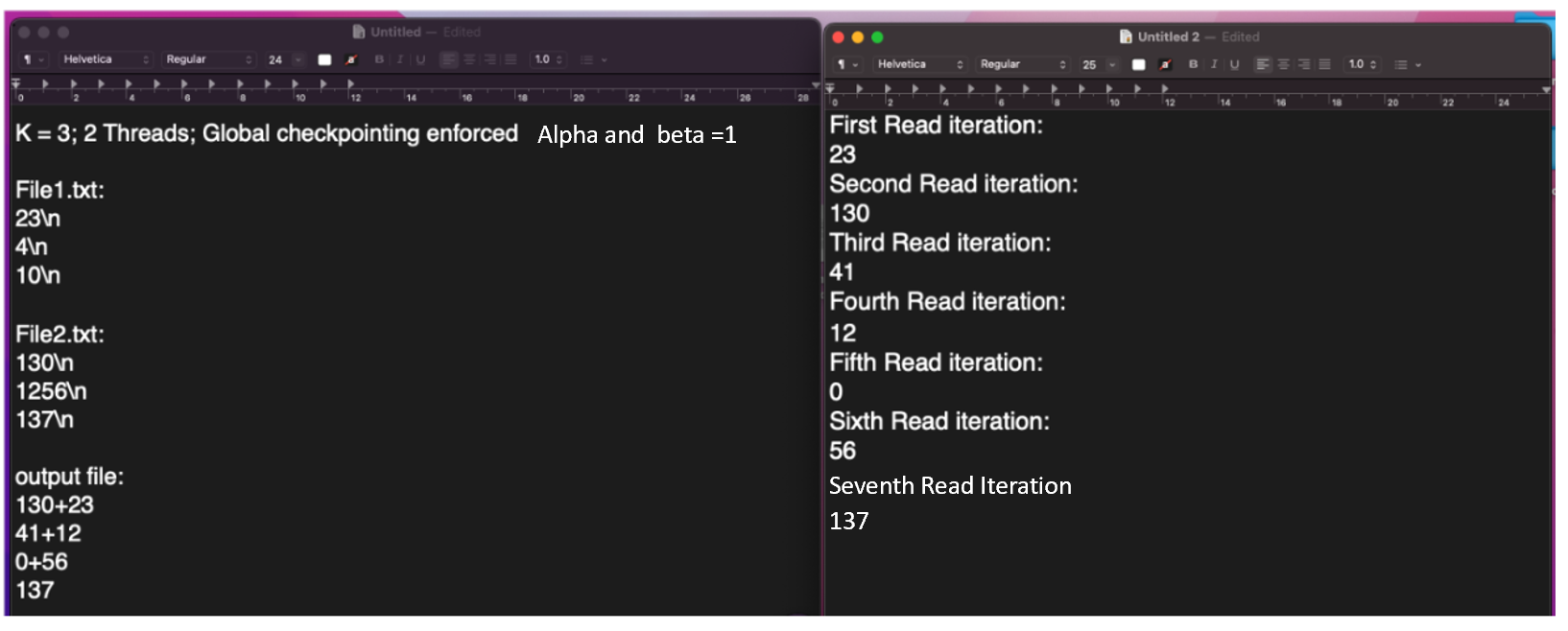
* For input file 1
  + 23\n -> 23\n -> 23
  + 4\n -> 4\n1 -> 41
  + 10\n -> 0\n -> 0
  + new\_sample\_value = alpha \* sample\_value + (1 - alpha) \* previous\_sample\_value
  + value | alpha
    - 23 | 23
    - 41 | 0.5\*41 + (1-0.5) \* 23 (previous sample value) = 20.5 + 11.5 = 32
    - 0 | 0.5\*0 + (1-0.5) \* 32 (previous sample value) = 16
  + Amplification value (beta)
  + It is the multiplicative factor used for a given sample
  + new\_sample\_value = beta \* sample\_value
    - 23 (new sample value) | 1 \* 23 = 23
    - 32 (new sample value) | 1 \* 32 = 32
    - 16 (new sample value) | 1 \* 16 = 16
* For input file 2
  + 130\n -> 130 -> 130
  + 1256\n -> \n12 -> 12
  + 137\n -> 56\n -> 56

-> 137 -> 137

-> \n

* + new\_sample\_value = alpha \* sample\_value + (1 - alpha) \* previous\_sample\_value
  + value | alpha
    - 130 | 130
    - 12 | 0.5 \* 12 + (1-0.5) \* 130 (previous sample value) = 6 + 65 = 71
    - 56 | 0.5 \* 56 + (1-0.5) \* 71 (previous sample value) = 28+ 35.5 = 63.5
    - 137 |0.5 \* 137 + (1-0.5) \* 63.5 (previous sample value) = 68.5 + 31.75 = 100.25
  + Amplification value (beta)
    - It is the multiplicative factor used for a given sample
    - new\_sample\_value = beta \* sample\_value
      * 130 (new sample value) | 0.5 \* 130 = 65
      * 71 (new sample value) | 0.5 \*71 = 35.5
      * 63.5 (new sample value) | 0.5 \* 63.25 = 31.625
      * 100.25 (new sample value) | 0.5 \* 100.25 = 50.125
* The final output will be
  + 23 + 65 = 88
  + 32 + 35.5 = 67.5 ~ 68
  + 16 + 31.625 = 47.625 ~ 48
  + 0 + 50.125 = 50.125 ~ 51

**One more example**



**Submissions**

For A3, the concrete required deliverables are:

* myChannels.c
* .h files (if any)
* Makefile
* metafile + input channel file (optional)

**FAQs (will be constantly updated)**

**Q1. Is the path to the metadata file and the channel files the relative or absolute path? Can it be either?**

A1. Absolute path is required

**Q2. Each thread can only read "k" bytes from a file at a time. Is this k value the parameter buffer\_size?**

A2. Yes

**Q3. Are we providing our channel files or will you just be using your own test channel files?**

A3. Save your channel files on /tmp/inputs and test accordingly. But you don't have to submit the channel files.

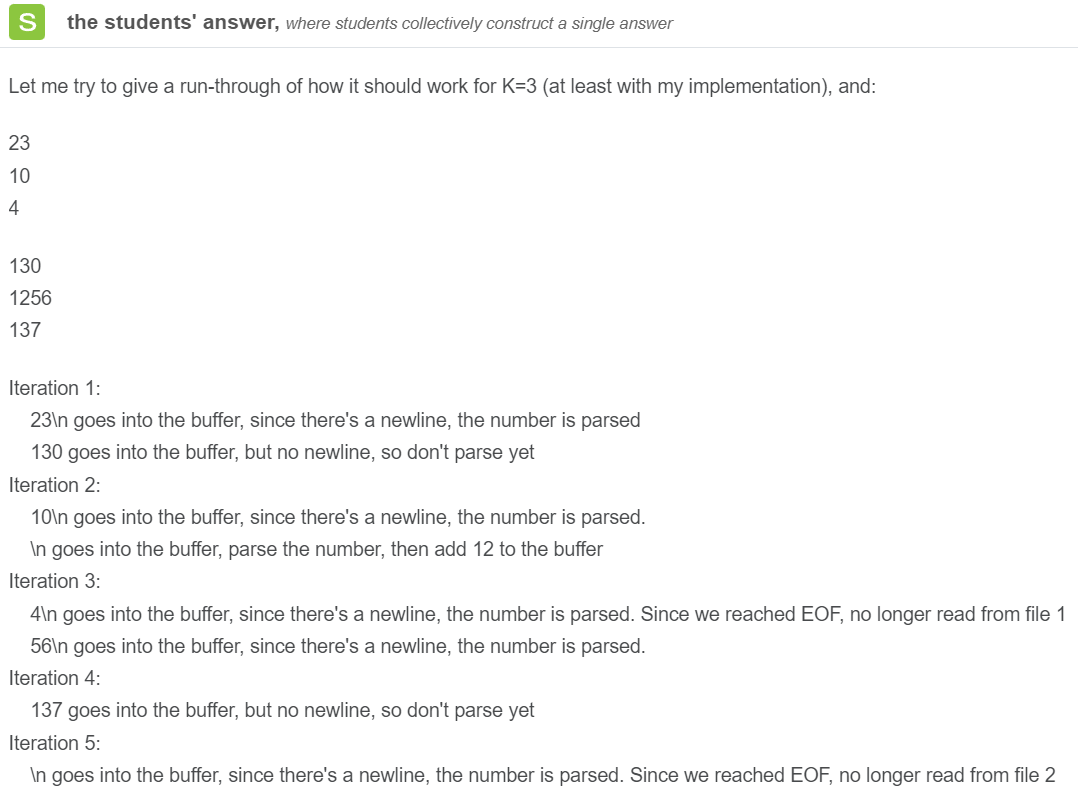
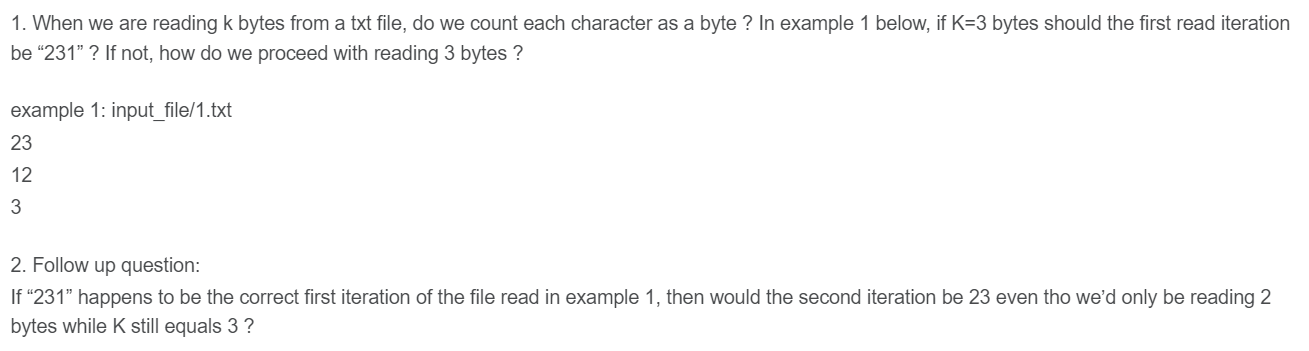
**Q4. Assume that the number of input files = p \* num\_threads (each thread will work with exactly p files, where p is some integer value) .Do we need to handle a case where the "p" value is a decimal? For example, 7 files with 2 threads means that each thread gets 3.5 files...? Is this a valid value, and how exactly do we proceed with this?**

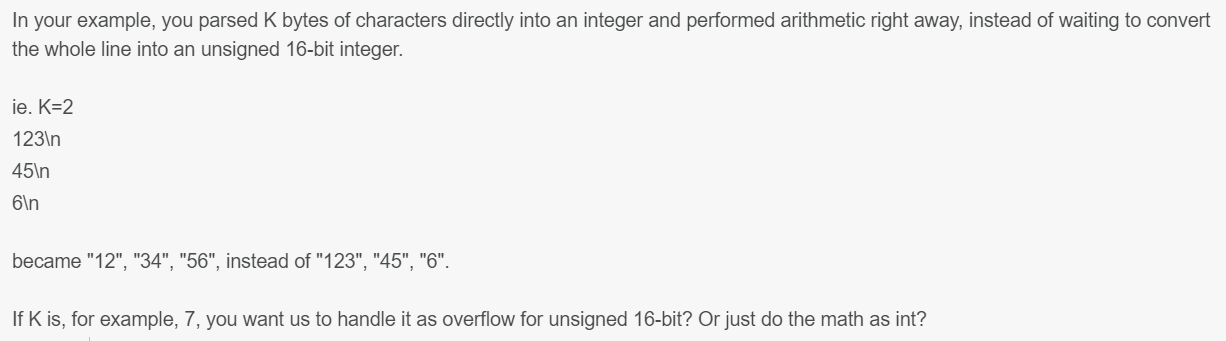
A4. p-value is an integer. The program should output the response "file is not in multiple of threads".

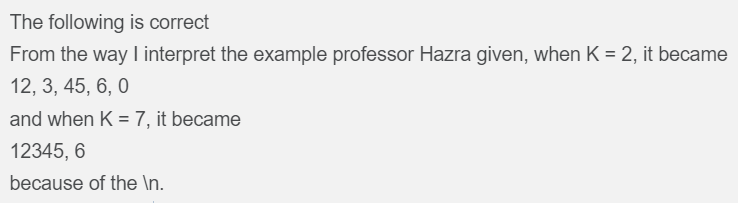
**Q5. Each thread can only read "k" bytes from a file at a time. The entire file should not be read/stored in memory. Can we read at most k bytes, or do we need to read exactly k bytes? For example, how can we deal with reading half of a number and also with line feeds if we read exactly "X" bytes?**

A5. Yes, it is k bytes + line feeds if any

Helpful conversation from previous Piazza







**Q6. Will the input file lines terminate with \r\n?**

A6. Yes.

Unix-based files typically use \n as the line ending, while Windows-based files use \r\n as the line ending convention. By specifying the use of \r\n as the line ending convention, we ensure compatibility with text files from different platforms and maintain consistency with the specified format. This approach allows for seamless handling of line endings and ensures proper interpretation of the file contents.

In computations, we are not considering \r as a byte. We are only considering \n as byte (check the given examples)