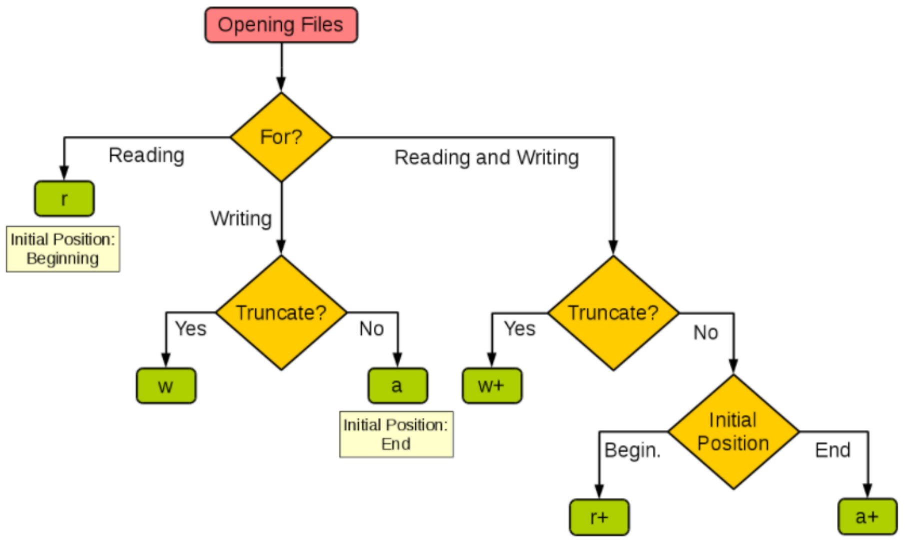
1. **For condition variables, are notify_one() and signal() the same thing? it says signal on the lecture slides but notify_one on cppreference website.**
   `notify_one` is used by cpp for signal (which unblocks one of the threads that was blocked by the condition variable on wait). `notify_all` is used for broadcast (which unblocks all threads that were blocked by the condition variable on wait). ***You must use notify_one in this PA.***

2. **Since there is only one histogram per patient, does that mean we will only use the ecg 1 or 2 values? Or will each histogram handle both values?**
   You should only be using ecg1 to update your histograms.

3. **Should I release the lock before I notify or notify before I release the lock?**
   Generally, it shouldn't matter which way you choose to go. But, for the purposes of minor optimizations - if you notify before you release the lock, it can lead to the notified thread being blocked again because the notifying thread still holds the lock. Therefore, unlocking before notifying is the preferred method to use.

4. **Are there any tips for where to find what the specific way of opening a file is that will allow multiple threads to write to it without corruption?**
   In the file request thread, you would allocate the file in memory (by opening in write mode then using a seek operation to touch all bytes that need to be allocated). The seek operation in the file request thread just tells the kernel how large the file will be.
   Then in the worker threads, when you process one of the file message requests from the request buffer, you would open the file in update mode, then use the offset to to set the seek location to the appropriate place to write the response. As long as the file message requests are well-formed and you reposition to the correct byte to start writing at, you shouldn't encounter a conflict.



Follow this diagram in order to help you determine the update mode that you will use. Note: Truncate means that you will empty the contents of the file and then write. We don't desire this.

5. **How would a worker thread know when it needs to stop reading from the bounded buffer?**
   You would push an equal amount of quit messages as there are worker threads to the request buffer after joining the data/file threads. Then when the worker thread reads a quit message, it would send it along its channel, then end its execution. You can now call join on the worker threads.

6. **How do I convert a vector<char> to a char* for popping from the BoundedBuffer? And similarly how do I convert char* to a vector<char> for pushing into the BoundedBuffer?**
   Use `vector::data()` to help you copy over the contents of vector<char> to char*. For converting char* to vector<char>, you can call the vector constructor in a certain way that would allow you to do this. The reference link here will be useful (you can look at the example code for hints).

7. **I see different ways to use the condition variable: the busy waiting in the while loop as well as the predicated version using lambda expressions. Which one should I use?**
   Either one works just fine, it is up to you to choose which style you want to implement. You can find more information here

8. **My program breaks when I call join(). What is the reason for this?**
   It's usually because your functions are stuck in an infinite loop and have not completed their functions because of not receiving the QUIT_MSG and terminating upon receiving it.

9. **I'm running into some problems with the filemsg transfer. Any tips on potential points I've overlooked?**
   Make sure you are calling `fclose(...)` in your file_thread_function after opening it and using seek to touch all bytes necessary. This allows the system to flush all data in the disk intended to be written to the file.