HW 5: Munching Text

You will create program to solve a variant of the producer-consumer problem, using multiple threads and only a limited amount of storage space. Your source code file should be called `muncher.c`.

This program will take 2 command-line arguments. They are (in order):

1. **The input file**, which is just a text file.
2. **The buffer size**, which is the size of an array of strings.

The program will start by allocating an array of strings of the given size (which will be called the *buffer*), and opening the input file. It will then start 4 separate threads, which are as follows:

1. **The reader**, which loops through the input file. Each iteration, it will read a single line of text from the file. It will then allocate space for it in position `(i % bufferSize)` in the string buffer, copying it over. Importantly, the space allocated for the string in the buffer must have enough space for up to 20 additional characters, which will be added to it later.
2. **The measurer**, which measures the length of each string that is in the buffer. It will append a space and the length (inside parentheses) to the end of the string.
3. **The numberer**, which prepends the line number of each string in the buffer (followed by a colon and a space).
4. **The printer**, which prints each modified string out, in order. It then frees the memory in the buffer that was used for the allocated string (allowing the reader to use that space to read the next string).

If any thread is unable to act, it should not enter a spinlock. Rather, it should remove itself from execution until such time as it is able to go.

Thus, if the input file `badpoem.txt` contained the following poem:

```
Roses are red,
Violets are blue.
Sugar is sweet,
And so are you.
```

then the program would work like this:

```
$ muncher badpoem.txt 2
1: Roses are red, (14)
2: Violets are blue. (17)
3: Sugar is sweet, (15)
4: And so are you. (15)
```

It would do this using a buffer of only size 2, so each string pointer in the buffer would be used twice.

Note that if you type the following:

```
$ muncher badpoem.txt 2 > out.txt
```

that it will direct the `stdout` stream to the file `out.txt`, rather than printing it to the terminal. You won't see any output on the screen at all! This might be a good way to check your code on large files.

Of course, the challenge is in making sure that the strings all work together nicely. The program cannot freeze, nor can any of the four threads do its operation on a single string more than once. No thread can operate on a string before the previous thread has done its work on that string. You will need to use mutexes, conditions, and/or atomic variables to accomplish this aim.

Also, be aware that you will probably be dealing with race conditions in your code! You will need to run it repeatedly (at least 10-20 times!) before you can be confident that it works!