My approach to project #2

Project #2 builds off the work you did in project #1. The following is how I approached the assignment. As with Project #1 you are under no obligation to follow my suggestions but if you are having trouble getting started, I hope this document will help.

You can start with the code you created in project 1 and add functions that will read the request, check to be sure the request is valid (it should be a GET request for a filename of the correct format). After parsing the request, you send back the requested file or the proper error code.

A few common problems to watch out for:

- We are not dealing with null-terminated strings. You cannot use the c-string or std::string library on data you read from or send to the network. Treat everything as raw data. If you want to use the string library to help with parsing, you must convert the data first.
- Because everything is a raw data don't try to use arrays. Use malloc() or new[] and work directly with pointers.
- The transport layer is not line-oriented. Each time you call read you will get some number of bytes (assuming there is data to be read). However what you read may not be a full line, or it may be several lines. The "lines" in HTTP are terminated with <CR><LF>, which look to the network like any other data.
    - When I'm coding, I make all my buffers really small (10 bytes) so I'm forced to remember that I have to reassemble things. It's less efficient (you wouldn't want to do that in the real world) but it causes me to be sure I'm reassembling things correctly.
- Double and triple check the size of the file and the value you put in the content-length parameter.

```
// ********************************************************************************
// * Read the header. Return the appropriate status code and filename
// ********************************************************************************
int readRequest(sockeFD, std::string *filename)
   1. Set the default return code to 400
   2. Read everything up to and including the end of the header.
   3. Look at the first line of the header to see if it contains a valid GET
          a. If there is a valid GET, find the filename.
          b. If there is a filename, make sure it is a valid filename.
                i. If the filename is valid set the return code to 200.
               ii. If the filename is invalid set the return code to 404.
```

```
// ********************************************************************************
// * Send one line (including the line terminator <LF><CR>)
// ********************************************************************************
sendLine(socketFD, stringToSend)
   1. Convert the std::string to an array that is 2 bytes longer than the string.
   2. Replace the last two bytes of the array with the <CR> and <LF>
   3. Use write to send that array.
```

```
// ********************************************************************************
// * Send a 404
// ********************************************************************************
```
send404(socketFD)

1. Using the sendLine() function you wrote, send the following:
    - Send a properly formatted HTTP response with the error code 404
    - Send the string, "content-type: text/html" to indicate we are sending a message
    - Send a blank line to terminate the header
    - Send a friendly message that indicates what the problem is (file not found or something like that)
    - Send a blank line to indicate the end of the message body.

```
// ********************************************************************************
// * Send a 400
// ********************************************************************************
```
send404(socketFD)

1. Using the sendLine() function you wrote, send the following:
    - Send a properly formatted HTTP response with the error code 400
    - Send a blank line

```
// ********************************************************************************
// * Send a file
// ********************************************************************************
```
sendFile(socketFD, filename)

1. Use the stat() function call to find the size of the file.
    - If stat fails you don't have read permission or the file does not exist. Send a 404
2. Using the sendLine() function you wrote send the header:
    - Send a properly formatted HTTP response with the error code 200
    - Send the content type depending on the type of file (text/html or image/jpeg)
    - Send the content-length
        - Note - if the content length and/or file type are not sent correctly, your browser will not display the file correctly.
3. Send the file itself.
    - Open the file.
    - Allocate 10 bytes of memory with malloc() or new[]
    - In a loop:
        - Clear out the memory with bzero or something similar.
        - read() up to 10 bytes from the file into your memory buffer
        - write() the number of bytes you read
    - when you are done you can just return. Since you set the content-length you don't send the line terminator at the end of the file.

```
// ************************************************************************
// * processConnection.
// * - this is the same processConnection function from project 1
// *   (with different guts inside)
// ************************************************************************
int processConnection(int sockfd) {
   1. returnCode = readRequest(sockfd, &filename);
   2. if returnCode is 400
         a. send400(sockfd);
   3. if returnCode == 404
            send404(sockfd);
   4. if returnCode is 200
         a. sendFile(sockfd, filename);
   5. Exit the function and return 0, so the main loop will close the connection and
      wait for the next request.
```