**Phase 2 Project Requirements**

The second phase of the project focuses on the ability to obtain information about files and webpages found on the web. This phase should be an independent standalone project. The functionality described here was presented in class and emailed to you when we discussed “source code” for accessing/reading webpages and documents on the web. For this phase ONLY, You can modify the code I provided in class as a basis for this project, but you will need to comment before each method used the source of your code (for example. the Flanagan code in class). For ALL other phases, you must write your OWN code and cannot use code from another source. (I will also allow you to modify the hash code found on the links I provided in a different email, but you MUST state the correct source in the comment section of that code.) Bottom line, assume that you CANNOT use any code besides which you wrote and if you have questions as to whether you can use a specific code for your project, then ask me.

This phase must work from the command line only. For this phase, your program will accept two file names from the command line by using the flags –i and -o. There will also be a -d flag which will contain a pathway to a directory that the user wants the system to store any of the obtained or outputted files. When submitting, submit just the .java code(s), any associated documents describing how to deploy and use your program and an example of input and output that demonstrates your program (detailed at end of this document.)

Now, for the details:

For this phase, your input file entered with the flag –i will be a list of URLs that You will provide. The lines of input in the file should be of webpages that end in a mix of .htm (or .html), .jpg (or .jpeg), .txt, .pdf and .docx file endings. Use the search engines to find documents with the required file endings.

The -d flag will be followed by a string in quotes provided by the user as to the path to the directory where the files referred to by the URLs and the info/output your system provide in the output file based on the -o flag.

For each URL you are processing, obtain URLInfo (as we discussed in class; if find other info that you can obtain, please do tell me about them). Output the name of the URL and all of its info about the URL to the output file (whose name will be the -o parameter on the command line you pass to java; again. –i provides the name of the input file, which contains a list of all of the URLs tested). Included in the processing (output), is determining the number of bytes in the file pointed to by the URL. See <https://www.mkyong.com/java/how-to-get-file-size-in-java/> for code samples on how to accomplish this.

The remaining processing will depend on the file ending that the URL points to (i.e. that you will be reading from). If the URL is a .html or .htm or .txt file, then read all the lines of the file and save it to a separate file on your system with the Same Name as the URL file name (i.e. last portion of actual URL). Then, output the number of lines read in and the name of the file to the output file (again this is referred to by the -o parameter on the command line). If the URL is an image file, .jpeg or .jpg or .gif, then save the image file to your computer with the Same Name as the URL file name. Then, output the name of the file to the output file. If the URL is .pdf or .docx then save the file to your computer with the Same Name as the file name found in the URL and output the name of the file to the output file. In the case of .pdf and/or .docx you will copy byte by byte instead of line by line because parts of .pdf files are stored in binary encodings and not ascii. See <https://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html> for a template of Java code that will help accomplish this. You will need to integrate this kind of code with the code discussed in class.

Your input file will be a listing of URLs each ending in a different ending from the previous paragraph: minimally,

one url ending in .htm or .html

one url ending in .txt

one url ending in .jpeg, .jpg or .gif

//First Note: some sites will not return these files due to copyright

//Second Note: Some .gif file have sophisticated encoded animations that may not be compatible //with the Java methods. If this is the case for your .gif file, then you will need to obtain such a .gif //file byte by byte using bytestreams. These are explained above.

one url ending in .pdf

one url ending in .docx

As emailed previously, the submission of this phase must include ALL of the following in one email from your Qmail to Class Projects email:

0) Subject:CS370 Phase 2 Submission <Due-Date-Of-Phase2>

1) The .java files needed for the project submission. These need to be PROPERLY commented as we discussed in class.

2) Final CLOC report on all of the submitted .Java files pasted to the BODY of the Email.

3) Readme.txt which explains any installation and compilation info from command line.

4) User Manual which is simply a word doc explaining how to use your code and if you have GUIs, provide snapshots of the GUIs and underneath explaining how the user can use it.

5) A sample input data file listing all of the URLs.

6) Output file

7) Any outputted logfiles (if your project produces them.)

A common question I receive is how do you find files with a specific filetype on the internet. There is an answer to this but at your level of the major should be able to figure that out on your own. If you cannot, you can use the following links:

<https://www.scipopt.org/academic.txt>

<https://www.ptgui.com/ptgui_eula.txt>

<https://www.pcre.org/changelog.txt>

<https://upload.wikimedia.org/wikipedia/commons/b/b6/Gilbert_Stuart_Williamstown_Portrait_of_George_Washington.jpg>

<https://upload.wikimedia.org/wikipedia/en/6/68/Walters_Gilbert_Stuart_George_Washington.jpg>

<https://upload.wikimedia.org/wikivoyage/en/1/1a/George_Washington.JPG>

<https://dmvnv.com/locat.htm>

<https://www.containerstore.com/welcome.htm>

<https://www.linkwitzlab.com/rooms.htm>