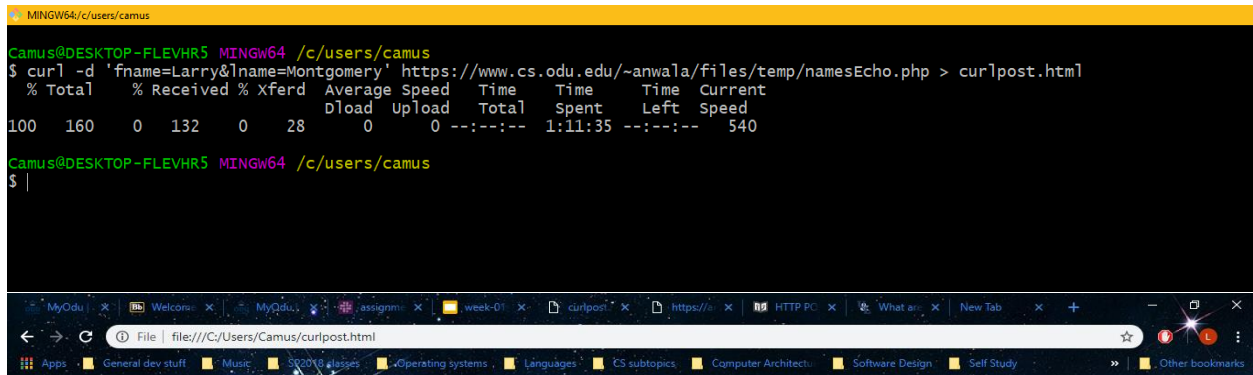


CS 432 Assignment #1

Larry Montgomery 01093132

Part 1: Post Command Curl

For this section I decided to use the suggested server provided by Professor Anwala. I used the names of the form fields the server would accept, those being "lname" and "fname" in conjunction with the curl "-d" option to POST my name to the server. The "-d" option in curl will send the name=value pairs to the server. Each of the name=value pairs must be separated by a "&". The response to POST option can be redirected to a file using the ">" symbol followed by the name of the file. The code I used to generate the response is in the screen shot below.



```
MINGW64/c/users/camus
Camus@DESKTOP-FLEVHR5 MINGW64 /c/users/camus
$ curl -d 'fname=Larry&lname=Montgomery' https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php > curlpost.html
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total   Spent    Left   Speed
100  160    0  132    0  28      0      0 --:--:--  1:11:35 --:--:--  540

Camus@DESKTOP-FLEVHR5 MINGW64 /c/users/camus
$ |
```

The screenshot shows a Windows terminal window with a yellow title bar. The user 'Camus' is at the prompt. They run a curl command to POST 'fname=Larry&lname=Montgomery' to a specific URL, redirecting the output to 'curlpost.html'. The terminal shows progress bars for data transfer. Below the terminal, a browser window is open, displaying the file path 'file:///C:/Users/Camus/curlpost.html'. The browser's address bar and tabs are visible at the top. The browser window shows the content of the file created by the curl command.

fname Posted: Larry
lname Posted: Montgomery

Part 2: Python Program

For the design of my python program I made use of two libraries outside of the default python packages Requests and BeautifulSoup4. I incorporated the Requests library to easily handle the GET and HEAD requests that were made. There is urllib but requests library provides a simple interface to switch between different HTTP operations and well as easily handle headers. The choice to use BeautifulSoup4 was to easily parse html code and extract the anchor tags and all references without having to write a html parser on my own. I made use of the system library to handle the command line arguments. I found a very simple process was to try and make a get request with the provided url from the command line argument. Take the response body and use the html parser from the BeautifulSoup4 to find all the anchor tags and extract the "href" field from them. This is where I will find the URL associated to the link. The picture bellow shows how I perform both.

```
# for each link in the object returned from find_all method  
# the link is checked to see if the link is absolute, site root relative,  
# or relative and adjusts the links before making the head request  
for link in soup_OBJ.find_all('a'):  
    curr_Link = link.get('href')
```

Once I had each link, I had to classify it as either absolute or relative to the page itself. Absolute links contain the entire path where the relative contain only the site relative path. Some relative paths start with a forward slash where some would not. The picture below shows the check for each type.

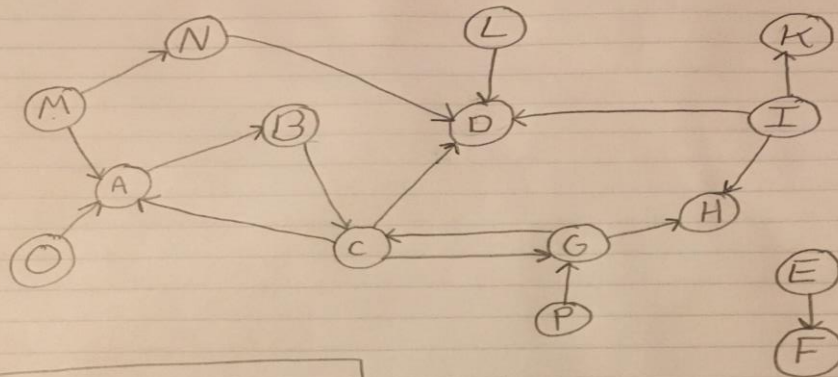
```
# if no links are found return  
if curr_Link == None:  
    break  
  
# skip all comments  
if curr_Link[0] == '#':  
    continue  
  
# check for site relative links  
elif curr_Link[0] == '/':  
    link = sys.argv[1] + curr_Link  
  
# check for relative links  
elif curr_Link[:4] != 'http':  
    link = sys.argv[1]+'/'+curr_Link  
  
# used for absolute links  
else:  
    link = curr_Link
```

Once the link classification is done, the link name is adjusted to the absolute path and a HEAD request is made, since I only care about the meta information for each of these links. The status code is checked to make sure it was a 200 meaning successful retrieval of the representation we requested and then the content-type is checked. Since the program only cares about PDF's I check to see if the content-type response header is of 'application/pdf', if so I print the link to stdout and the number of bytes of the representation. The number of bytes can be gained from the content-length response header. The picture below shows the checking the status and content-type.

```
# checks for conditions of successful transaction and the type of data  
# then outputs to stdout  
if res.status_code == 200 and res.headers['content-type'] == 'application/pdf':  
    print('link: ' + link)  
    print('content-length: ' + res.headers['content-length'] + ' bytes')
```

Part 3: Bow-Tie Graph

My process for determining the categories for each node in the graph below was to first start by finding the SCC nodes, since they have the hardest requirement of being reachable from every other SCC node as well as being able to reach all other SCC nodes. Nodes A, B, C, G were the only nodes I found that could satisfy the requirements. I then moved to classifying input nodes focusing on the simple case of a node having only output directed connections. I discovered O, P, I, M, L. Next, I moved to out nodes, again focusing on the simple case of a node having no output directed nodes. I found D, H, K. This left me with the nodes N, E, F. Since both nodes E and F are not connected to the rest of the nodes in the graph they were labeled as disconnected. Now leaving me with node N. I looked at both tubes and tendrils to see if this fit either case. At first I thought about having node N be a tendril but the rule stated that the node can only have in-links from a node in IN or out-links to a node in OUT but not both, since N has both, an in-link from node M and a out-link to node D, it fit the description of the Tubes best.



IN: O, P, I, M, L

SCC: A, B, C, G

OUT: H, K, D

Tendrils:

Tubes: N

Disconnected: E, F