**\*\*\*\*High-level approach\*\*\*\***

The goal of this project is to implement a web crawler that gathers data from a fake social networking website. We have created sockets for establishing client-server communication. We followed object oriented approach to integrate required functionalities. All properties of WebCrawler have been declared as variables (e.g. Frontier = [], listOfVisitedLinks = [], host = "cs5700sp15.ccs.neu.edu", port = 80, httpStatusCode = 0, csrToken ='', sessionID = ''), and behavior of WebCrawler (make GET/POST requests, send/receive messages, handle HTTP STATUS and perform calculations) have been defined as methods.

User runs this project proving credentials of his fakebook account (username and password). Parameterized constructor has been used to initialize values of variables like hostname, portnumber etc. The project is made responsive to two mandatory parameters that are provided as command line arguments while running project from the terminal i.e. username and password. We have implemented TRY and CATCH for handling exceptions.

Core logic of this project implements "DEPTH-FIRST SEARCH"; all links, which are yet to be crawled are saved in a FRONTIER, and links are obtained from this frontier one by one based on FIFO (FIRST IN FIRST OUT) until all five keys are discovered. As soon as a link is dequeued from the frontier it is marked as 'visited' (i.e. inserted into a list containing already crawled links) such that when new links are discovered while downloading a page, it is enqueued in the frontier if and only if it does not exist in FRONTIER and ALREADYVISITED list.

1. Constructor calls a method for making a GET request to server for downloading the root page of the website.

2. In response to the GET request made in Step 1, it parses downloaded page, handles http status message, and then in turn inserts received link into a frontier.

3. As soon as http://cs5700sp15.ccs.neu.edu/accounts/login/?next=/fakebook/ link is crawled it makes a POST request with username and password provided by the user.

4. Once the webcrawler program has successfully logged in, it makes GET requests on urls (picks from frontier on FIFO basis) to download pages. Once an URL is crawled it's removed from the frontier, and added to another list marking it as a visited/crawled link.

4. While downloading pages it handles 301,302, 404, 500, and 200 HTTP Status codes as mentioned in the problem set.

5. While crawling the website when following link is discovered in any page

<h2 class='secret\_flag' style="color:red">FLAG: 64-characters-of-random-alphanumerics</h2>, this program prints the key on screen.

6. After successfully discovering all five keys, this program gets termminated.

**\*\*\*\*Challenges faced\*\*\*\***

1. While creating headers for HTTP GET and POST request we used developer tools to include mandatory headers.

2. the most of the efforts were consumed while making HTTP POST header along with the message body. We not unable to log in to the website programmatically. The problem we were facing was related to the "CSRF Verification". We resolved this error checking the correctness of our HTTP header going through following steps:

a. Telnet [server name] 80

b. Pasted our HTTP header along with message containing username, password and csrfmiddlewaretoken fields.

Once we could send the response using telnet, we included that header in our program.

3. We faced problem while handling status codes, we resolved it by adding new method containing logic for this such that it was called on every message we were receiving from the server.

4. Filtering out secret flags, we accomplished it using htmlparser library.

**\*\*\*\*Overview of how we tested your code\*\*\*\***

1. We performed unit testing without running project from the terminal, we hard coded credentials and tested the functionality.

2. We started testing from the terminal by passing authorized username and password as command line arguments.

3. We provided wrong username and correct password to test if our program terminates with an error message.

$ ./webcrawler wrong\_user\_name correct\_password

**Expected Behavior**: terminates the program and prints an error message.

**Output** : "Error occured!! Please provide correct parameters and try again!!"

4.We provided correct username and wrong password to test if our program terminates with an error message.

$ ./webcrawler correct\_user\_name wrong\_password

**Expected Behavior**: terminates the program and prints an error message.

**Output** : "Error occured!! Please provide correct parameters and try again!!"

5.We provided wrong username and wrong password to test if our program terminates with an error message.

$ ./webcrawler wrong\_user\_name wrong\_password

**Expected Behavior** : terminates the program and prints an error message.

**Output** : "Error occured!! Please provide correct parameters and try again!!"

6. We provided correct username and correct password to test if the program terminates after collecting the secret flags

$ ./webcrawler correct\_name correct\_password

**Expected Behavior**: terminate the program after printing the five secret flags successfully

**Output** : terminates the program after printing the five secret flags successfully