**Introduction**

In this PA, you will write a HTTP client that will download a webpage given its URL (i.e., the link to the page). Just like the server in your PA6, an HTTP server supports TCP connections. However, the difference is that HTTP servers use the standard HTTP protocol which you must implement from your client program to be able to communicate with it. As you can imagine, there are similar programs out there who perform similar functions. The most notable example is your browser that can connect to a remote web server, download the content and view it in a nicely formatted manner inside the browser. Another example is the linux command wget, which downloads a given page; but instead of showing it, wget actually just the page as a file in the current directory. We are going to emulate the functionality of wget in this PA. The work done by the browser for viewing the page content requires more work, and therefore it is beyond the scope of this PA.

**HTTP Protocol Basics**

Have a read on the basics of this protocol [here](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview). Similar to the protocol we used in PA4/5/6, each request/response pair is isolated from the other pairs, which underscores the ***stateless*** property of HTTP. However, through HTTP cookies, you can still establish a ***session*** which lets you run a lengthy conversation with a web server.

The workflow in a HTTP request/response is the following:

1. The client always initiates a conversation with the server by first making a TCP connection, and then by sending a properly formatted HTTP request to the server. The server never initiates a conversation. Look [here](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#HTTP_flow) to find more about how to construct such a request. Note that given a URL, you have to parse it into two parts: *host* and *request*, which you will then use to formulate the proper HTTP request and send it over. You can also specify certain client properties and preferences (e.g., whether compression allowed, supported language(s), page encoding support).
2. The server sends a response back to the client. The response contains two parts: a *header* and an optional *body,* separated by an empty line. The header is basically the metadata for the response body and encoded in the form of key-value pairs. The header also informs you of the *status code*, which tells you what happened to your request. [Here](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html) is a list of comprehensive status codes that you can expect. Status code 200 means that the request is successful and the body contains the HTML page that you requested for. However, other status codes mean other things. For example, if the requested page no longer exists, you will get status code 404. If the page is moved to a known location, the server may give back a 302 status code and a new location where the page moved to, which is usually put in the *Location* field inside the header.

Upon receiving the response (header and body), first you should check the header. If the response code is 200, you only need to save the body as a file. If the response code is 302, you should be able to follow the “Location” field and issue another request to get the moved page. If the status code is 40x, you should report that too and not make any file, because the body must be empty.

**Your Task**

Write a C++ client program where you:

1. Take in a URL from the user as a command line argument

2. Make a TCP connection with the server that hosts the URL (e.g., microsoft.com)

3. Issue a properly formatted HTTP GET request to the server. Find out how to do that from this

source https://www.tutorialspoint.com/http/http\_requests.htm

4. Receive the response from the above request, discard the HTTP header and then, save the rest (i.e., HTML content of the page) into disk after examining the response code.

5. To verify that you downloaded the file correctly by downloading the same page using the linux command wget and save that using another name. Now compare the 2 files using the diff command in linux. There should not be any difference

6. Test your program to download at least 20 URL in this [link](https://docs.google.com/document/d/1YrPHi5n2KHPkya6Ii0pkNqHwVRWF79DXPxewEVVWKH4/edit?usp=sharing) and show that your program works correctly for all of them.

A few other points:

1. You must test your program with URLs containing both host name and request portions. For

instance, the URL www.microsoft.com only has the host name portion – the request portion is an

empty string. However, http://faculty.cse.tamu.edu/tanzir/fa19/313/ has both non-empty host

(i.e., faculty.cse.tamu.edu) and a non-empty request (i.e., /tanzir/fa19/313).

2. You must handle dynamic web pages with URLs containing “?” symbols.

3. Your program must be able to work with binary web-pages and show the content correctly. This will not be a problem if you do not use std::string to hold the response.

4. Your program must be able to download large web pages that do not fully download in a single response. In such cases, the server returns multiple HTTP fragments and you must put them all in a single output file.

Submit your tested program in ecampus with a demo video. No report necessary