Version: 2.0
Due: March 14

Problem Set 3

You are to develop a simple web server in C. The web server will bind to the loopback interface. The web server must be started with a single argument that is the port number on which the server will run. The server should handle http requests such as (assuming the server is running on port 8000):

http://127.0.0.1:8000/index.html
http://127.0.0.1:8000/cars/ford.html

The server will store html static content in a local directory called **static**. Accordingly, the above URLs will be resolved to the following files on the server:

URL	file location
http://127.0.0.1:8000/index.html	static/index.html
http://127.0.0.1:8000/cars/ford.html	static/cars/ford.html

The server should implement the following logic:

- **Connection Setup**: Create a socket to wait for incoming connections. Make sure the socket is bound to the loopback interface (INADDR_LOOPBACK). You should configure the socket to have only one pending connection. Upon receiving a connection, you should accept it and process the incoming http request as discussed in the next step.
- **HTTP Request:** Your server should only handle HTTP GET requests. A complete description of how an http request is formatted may be found in RFC 2616. For the purposes of this assignment it is safe to assume the following restricted format:

GET SP Request-URI **SP HTTP/1.1 CRLF**
(HTTP-Key**:** HTTP-Value **CRLF)***
CRLF

Notation: Text in bold blue will appear as it is in the header packet. **SP** stands for space and **CRLF** for '\r\n'. Request-URI is the request URI. The HTTP-Key:HTTP-Value are header fields that may appear. The ()* indicates that there could be zero or more header fields. Your server should read these fields and make sure the header is formatted correct but it can safely ignore them (i.e., you do not need to take any actions on them).

The server will make sure that the incoming request is consistent with the above format. It is safe to assume that the Request-URI will not exceed 255 bytes. This functionality should be

implemented as part of the following method:

```
// Input:
// socket - the client socket from where data will be read
// uri - a pointer to where the uri will be saved
// Output:
// returns 0 if there was no error and -1 if an error has occured
int process_http_header(int socket, char *uri)
```

- **Responding to an HTTP request:** There are three possible outcomes when handling an HTTP request:
    a. Bad Request: the HTTP request was invalid (i.e., it does not copy with the above format). In this case your server should respond with a Bad Request reply.
    b. Not Found: if the HTTP request is valid, then your program the prcess_http_header should store the Request-URI into the memory location pointed by uri variable. The Request-URI must be translated into a file location as indicated in the URL => file mapping table. After performing this translation, the server must check if the request file exists. If it does not, the server should respond with a Not Found error.
    c. No errors: in the case when the file exists, the server shall read the contents of the file and generate an HTTP reply.
- **Generating an HTTP reply:** In the case of finding the appropriate file to server on the web server, you need to send the content. However, you need to generate an appropriate HTTP header. In our case the minimal http header is:

```
HTTP/1.1 SP 200 SP OK CRLF
Content-Type: text/html CRLF
CRLF
```

After sending the header, you can send the contents of the file.

As usual, you have to submit your homework through ICON.
The following resources may be useful in solving the homework:

- HTTP RFC -- https://tools.ietf.org/html/rfc2616