```
/*
   Kaveh Pezeshki and Christopher Ferrarin
   E155 Lab 6 Generic Webserver

   Hosts a webserver displaying webpages sent over UART, sending back redirect
    URLs to provide a measure of user interaction.

   In more detail:

   1) The webserver starts a 9600 baud serial connection over the hardware UART
    (for debug)
   2) The webserver starts a 9600 baud software serial connection over pins 14
    and 15 (RX, TX)
   3) The webserver connects to a given network, and prints status information
    over the debug UART. By default, this is CINE.
   4) The webserver sends an empty request '<>' to the microcontroller to
    obtain a copy of the webpage
   5) The webserver initializes an HTTP server and a hardware refresh timer
   6) The webserver waits for a request from the client, after which it
    transmits an updated webpage

   steps 4-6 are repeated while the program runs

   In parallel, a hardware timer triggers an interrupt every 10 seconds. This
    will cause a refresh of the webpage from the MCU by
   sending the most recent abbreviated URL to the MCU as "<" + <abbr. url> +
    ">", and waiting for an updated webpage to be returned
   over UART. This only occurs if the following conditions are met:

   1) The hardware timer interrupt has not been handled, occuring when
    refreshWebpage = true
   2) A client has requested data from the webpage after the last refresh of
    the webpage from the MCU. This occurs when webpageUpdated = false

   Abbreviated URL explanation:
   The webserver automatically parses a client request to simplify code on the
    MCU. If the server has example IP address '192.168.1.1', a client request
    may be the URL:
   'http://192.168.1.1/webpage'
   The server will return everything after the IP address and slash. For
    example:
   'http://192.168.1.1/webpage' => '<webpage/>'

   We expect these abbreviated URLs to be under 10 characters
*/

//Importing required libraries
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>

//defining start and end HTML tags
```

```
const String htmlStart = "<!DOCTYPE html><html>";
const String htmlEnd = "</html>";

//Defining network information
const char* networkName = "CINE"; //set this to the selected network SSID
const char* password    = NULL;   //set this to a non-null value if selected
 network requires authentication
String      ip;                   //stores the current IP. Set in the setup
 function

//Defining the web server and HTTP request variables
WiFiServer server(80);           //The server is accessible over port 80
String     request;              //Stores the client HTTP request
String     parsedRequest = "<>"; //Stores a simplified version of the HTTP
 request to transmit to the MCU
String     currentLine;          //Stores a semi-parsed version of the HTTP
 request
String     webpage = "WAITING FOR DATA";  //The current webpage, updated by the
 MCU

//Defining the softwareSerial interface
SoftwareSerial mcuSerial(14, 15);

extern "C" {
#include "user_interface.h"
}

//defining the webpage refresh timer
os_timer_t refreshTimer;
bool refreshWebpage = false;
bool webpageUpdated = true;

//Timer callback. This function will run when the timer reaches the set webpage
 refresh time
void timerCallback(void *pArg) {
  refreshWebpage = true;
}

//Setup code. Runs once on program execution before loop code
void setup() {
  //starting the debug and MCU serial connections
  Serial.begin(115200);
  mcuSerial.begin(9600);
  Serial.println("Set up serial connections");

  //connecting to WiFi network
  Serial.print("Connecting to network ");
  Serial.println(networkName);
  Serial.print("With password ");
  Serial.println(password);
  WiFi.begin(networkName, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Attempting connection...");
  }
  //connected to the network. Printing status information
  Serial.print("Connected to WiFi network with IP: ");
  Serial.println(WiFi.localIP());
  ip = ipToString(WiFi.localIP());
  Serial.print("AP IP: ");
  Serial.println(WiFi.softAPIP());

  //starting server
  Serial.println("Starting server");
  server.begin();

  //fetching a new webpage
  receiveWebPage("<>");

  //defining the webpage reload timer
  os_timer_setfn(&refreshTimer, timerCallback, NULL);
  os_timer_arm(&refreshTimer, 10000, true); //fetches a new webpage every 10
   seconds
}

//Main program. Runs repeatedly after setup code
void loop() {

  //we update the webpage every 9 seconds to prevent timeout errors
  if (refreshWebpage & !webpageUpdated) {
    refreshWebpage = false;
    webpageUpdated = true;
    webpage = receiveWebPage(parsedRequest);
  }

  //Wait for a new connection
  WiFiClient webClient = server.available();
  //If a client has connected, we wait for a request
  if (webClient) {
    webpageUpdated = false;
    currentLine = "";
    Serial.println("\nClient Connected");
    while (webClient.connected()) {
      //Reading available bytes from the client if available
      if (webClient.available()) {
        char byteIn = webClient.read();
        request += byteIn;

        //if the line is only a line feed, we have reached the end of the
         client request and will therefore send a response
        //This requires sending a request for a new webpage to the MCU
        if (byteIn == '\n') {
```

```
          if (currentLine.length() == 0) {
            //transmitting the response
            //transmitting HTTP header and content type
            Serial.println("Transmitting webpage");
            webClient.println("HTTP/1.1 200 OK");
            webClient.println("Content-type:text/html");
            webClient.println("Connection: close");
            webClient.println();
            //transmitting the full webpage
            webClient.println(webpage);
            //transmitting an extra newline to catch transmission termination
             errors
            //webClient.println();
            break; //disconnect from the client by breaking from while loop
          }
          else {
            currentLine = "";
          }
        }
        else if (byteIn != '\r') {
          currentLine += byteIn;
        }
      }
    }
    //ending the transaction
    if (parseRequest(request) != "<>") {
      parsedRequest = parseRequest(request);
    }
    request = "";
    webClient.stop();
    Serial.println("Client disconnected");
  }
}

//Converts an IP address to a String
String ipToString(IPAddress address)
{
  return String(address[0]) + "." +
         String(address[1]) + "." +
         String(address[2]) + "." +
         String(address[3]);
}

//Parses an input http request as specified in "Abbreviated URL Explanation"
String parseRequest(String request) {
  Serial.print("////START Received Request: ");
  Serial.println(request);
  Serial.println("////END Received Request: ");

  //favicon is a common icon formatting scheme that tends
  if (request.indexOf("favicon.ico") != -1) return "<>";
```

```
  int getLocation = request.indexOf("GET /");
  int httpLocation = request.indexOf(" HTTP");

  String parsedRequest = "<" + request.substring(getLocation + 5, httpLocation)
   + ">";

  Serial.print("Reduced URL: '");
  Serial.print(parsedRequest);
  Serial.println("'");
  return parsedRequest;
}

int substringInString(String haystack, String needle) {
  for (int i = 0; i < haystack.length()-needle.length(); i++) {
    bool foundsubString = true;
    for(int j = 0; j < needle.length(); j++) {
      if( haystack[i+j] != needle[j]) {
        foundsubString = false;
      }
    }
    if (foundsubString) {return true;}
  }
  return false;
}

//Sends parsedRequest over UART to the MCU and waits for a complete webpage to
 be returned
String receiveWebPage(String parsedRequestIn) {
  Serial.println("////START Received Web Page");
  webpage = ""; //clear webpage in preparation for new webpage to be
   transmitted
  bool webpageReceived = false;
  bool startReceived = false;
  bool endReceived = false;

  //transmitting the parsed request from the client
  Serial.print("Transmitting:");
  Serial.println(parsedRequestIn);
  mcuSerial.print(parsedRequestIn);

  //wait until the entire webpage has been received
  while (!webpageReceived) {
    ESP.wdtFeed(); //resetting watchdog timer
    //checking for new serial data, adding to website
    while (mcuSerial.available()) {
      char newData = mcuSerial.read();
      webpage.concat(newData);;
    startReceived = (webpage.indexOf(htmlStart) != -1);
    endReceived = (webpage.indexOf(htmlEnd) != -1);
    webpageReceived = startReceived && endReceived;
```

```
    }
  }
  Serial.println("Received webpage");
  Serial.println(webpage);
  Serial.println("////END Received Web Page");
  return webpage;
}
```