CST8234 – C Language and Network Programming

Assignment #3 – A Text-Based Web Source Browser

DUE: Sunday, December 7th at 11:59 p.m. (i.e. 1 minute to midnight)

Purpose

The purpose of this assignment is to:

- Explore C's string manipulation functions;
- Use network connection code in C to download HTML source files;
- Gain experience in parsing strings; this knowledge will assist you in CST8152, the Compilers course.

Description of the Problem to be Solved

In this short assignment, you'll use socket interface code to download HTML/JavaScript tags from the internet. The code is prepackage in the function `getHTML()`, so for the first part of this project you only need to send an URL as an argument to `getHTML()`, and store the HTML source code returned in a character array. Your program will then read through the HTML source code as one very long string and display each URL in the order found, with a number alongside each link. The user selects the number associated with each hypertext link in the list, the web page associated with that link is then accessed using `getHTML()`, and the process repeats until the user decides to quit. So, on each cycle through the code, the user is presented with the links contained in the previously-selected web page.

Terminology

Each URL consists of many pieces; in this program, we'll be operating on two of them, which for simplicity we'll call the *domain* and the *fragment*. In an URL like "www.example.com/myExample/foo.html", the *domain* is everything before the '/', i.e. "www.example.com", and the *fragment* is the '/' and everything after, i.e. "/myExample/foo.html". The fragment identifies a webpage, although the domain by itself can also identify a web page, usually the home page for the domain. In general, if the word *web page* is used, it means "the page identified by the fragment."

Program Methodology

1. The library `getHTMLPage.c` contains a single function called `getHTML()`, which has its `extern` declaration in `getHTMLPage.h`. This function takes three arguments, each as pointers to a `char`: the domain name of the web page (e.g. “www.algonquincollege.com”) , the web page within the host (e.g. “/sat/” for the School of Advanced Technology web page) , and the address of the buffer that receives the source code after the server responds with the HTML source code. `getHTML()` returns a ‘1’ if problems occur, 0 otherwise, and your code will need to check for this error condition frequently .

   Note: (1) the first argument of `getHTML()` must be *just* the domain name, and must not include anything after the first single ‘/’. This is required because the `getHTML()` gets the IP address of the web site from DNS, and will not work if you request anything other than a proper domain name; and (2) if you only enter the domain name in the first argument of `getHTML()` with no other webpage listed (for example, if you wish to access the home page only), the 2nd argument of the function should be `NULL`.

2. The files `getHTMLPage.c` and `getHTMLPage.h` are available in CANVAS, along with a file called `Assignment3_test.c` containing some sample code that demonstrates the use of `getHTML()`. Compile this code using

   ```
   gcc Assignment3_test.c getHTMLPage.c –o Assign3
   ```

   and check the output (which gets dumped, in a substantial mess, into your terminal). Note: (1) many modern web pages are tens of kilobytes of source code in size, and may take several seconds to download even on a fast computer; simply because there’s no graphical output (or any output at all for that matter), doesn’t mean you can expect an instantaneous response; and (2) the default text buffer in Ubuntu is probably not large enough to hold all the source text from most sites, so what you see when you run `Assign3` will be just the latter portion of the file. The entire file should be stored in the buffer (called `HTMLSource[]` in the test file) prior to output—this is what your program will be working with. The size of this buffer is set to 100 kB, and while this should be large enough to hold most web pages, it will not hold the largest. Where the first portion of a large file is truncated during download, don't worry about the missing bits; just use the portion of the file you receive and work with that.

3. The `strstr()` function in `<string.h>` allows you to search for one string inside another. The function prototype is:

   ```
   char *strstr(const char *searchMe, const char *subString);
   ```

   where `searchMe` is the string to be searched—essentially the entire download buffer—and `subString` is the term you will be searching for. The value returned is the address of the first character of the match; `NULL` is returned if no match is found. (When an address is returned, this address plus one becomes the new address of `searchMe` on the next iteration through the loop). Initially, your code should search for the `<base …>` tag; if found, then the base becomes the prefix for every subsequent anchor tag found.

4. Your code should include the following three functions, which will be used in the rest of the program and called from `main()`. They should be collected in a library called `ParseURL.c` and accessed via the appropriate .h file. Each function may use any function found in the `<string.h>` library, but the bulk of your code will probably make extensive use of `strstr()`, `strlen()`, `strcat()`, and `strcpy()`.

   a. `char *findURL (char *)` takes as an argument the address of the string buffer that contains a `<a …>` or `<base …>` tag and returns everything after `http://`, up to the occurrence of the first space afterwards. Remember, URLs do not include spaces, so when you find the address of the first space *after* `http://`, everything in between the two can be taken as part of a domain name. Remember that if a `<base >` tag is found, every anchor tag that appears afterwards is relative to that base. To find the base tag initially, you'll want to use

      ```
      char *addr = strstr(source, "<base ")
      ```

first to locate the address of the first (and presumably, only) occurrence of the `base` tag in the HTML source code. The value of `addr` returned becomes the starting place to look for an URL using the `findURL()` function just described.

b. `void URLSplit(char *completeURL, char* domain, char * fragment)` takes a complete URL, such as "http://en.wikipedia.org/wiki/Ada_Lovelace" and splits it into the domain ("en.wikipedia.org"—which is returned to the `domain` argument of the function), and a remaining fragment ("/wiki/Ada_Lovelace") which is passed to `fragment`. As a rule, the first single "/" after "http://" will mark the start of the fragment; everything else before can be taken as the domain.

c. `char *URLCat(char *base, char *fragment)` performs the opposite operation from `URLSplit()`, appending the base to the fragment to form a complete URL. For this operation you'll want to use `strcat()`, which has the function prototype:

```
  char *strcat(char *dest, const char *src);
```

where the string pointed to by `src` is appended on to the end of the string point to by `dest`. Recall from the course notes that `dest` must be large enough to hold the extra characters stored in `src`. Why do we need this function? Because nothing says the base has to be a well-formed domain name. For example, if the base tag is

```
   <base href="http://www.adobe.com/support/">
```

then we can't treat `base` as if it's a proper domain name. The easiest way to deal with this problem is to concatenate the base with its anchor tags/ fragments, and then split them properly into domain and fragment using `URLSplit()`.

These functions utilize very general rules for parsing an HTML file, and while you may find it necessary to make minor modifications, they will serve you well. The purpose here is not to make a text-based HTML 'browser' that works 100% of the time, but merely to make one that works reliably most of the time.

5. The following pseudocode may be of assistance in helping you think about this program. This is provided only as a guide; your code may differ.

```
// URLS is an array that holds domain & fragment/webpage as
// arrays of chars.  Alternately, use two separate arrays to
// hold pointers to the each set of strings, one array for the
// base/domain strings, the other for the fragment/webpage
// strings
DECLARE URLS[300][2] /to hold up to 300 URLS incl. base and frag

//initialize URLS[][1] to the initial URL entered
GET domainName      // Prompt user for domain
URLS[0][0] <- domainName
GET fragment/web page    // Prompt user for web page/fragment
URLS[0][1] <- fragment
DECLARE Pick = 1;   // initial page will be loaded into URLS[0]

// Enter main loop
WHILE(Pick)  //true initially; continues until user chooses 0

   // Find links in page selected
   // NOTE: URL number offset by 1, since 0 = quit in menu below
   source <- getHTML(URLS[Pick-1][0], URLS[Pick-1][1])
   LCASE(source);        // convert source to lower case
   base <- {0}, frag <- {0}; // initialize using memset()

   // Check to see if base tag exists
   IF ((addr <- strstr(source, "<base "))!=NULL)
      base <-findURL(addr);
   END IF

   CTR = 0;
   WHILE ((addr <- strstr(source, "<a "))!=NULL)
      frag <- findURL(addr); //but could be complete URL as well

      // If base exists, it may not be a true domain name; we
      // need to combine it with the fragment first, then break
      // it apart using URLSplit into its proper constituents

      IF (base) // if base, then combine with frags to form URL
         url <- URLCat(base, frag);
      ELSE
         url <- frag;  // fragment is in fact the complete URL
      END IF
      URLSplit(url, domain, frag); //break it into domain & frag
      URLS[CTR][0] = domain;  // should hold the domain
      URLS[CTR][1] = frag;  // should hold the fragment/web page
      CTR = CTR + 1;
      source = addr + 1;  // inc. address for next anchor search
   END WHILE
   PUT ("Please make a selection:")
   PUT ("0. Quit Program")
   FOR (I=0; I < CTR; I++)
      PUT I, base, frag     // "1. www.adobe.com/support/" etc.
   GET Pick  //Note: Pick = 1 stored in array location 0
END WHILE
```

Marking

1. Code executes successfully.	**/1**
2. Followed all the instructions in this document (along with late additions or changes as posted) related to the functionality of the code, where such information was specified; did not attempt to subvert any of the general instructions nor the spirit of the assignment. This includes, e.g. using .h files where requested or implied to do so.	**/5**
3. Documentation. The code is clearly labelled according to the documentation standard provided.	**/3**
4. Clarity of code. Terse is good, but not to the point of incomprehensibility. Clearly, bloated, unnecessary code is to be avoided at all costs. If your code looks ugly, then it probably is ugly; it needs to be rewritten.	**/4**
5. The makefile functions as required and clearly specifies a rule for each dependency within your code, but does not imply unneeded dependencies, e.g. .h files which are already included inside .c files	**/2**
Total:	**15**

Submission Requirements

- All submitted code should be documented according to the 'CST8234 Documentation Standard' (available in Canvas)
- Your must provide a makefile that, when run, creates properly compiled and executable output. Your code should include a `Main.c` file that is responsible for calling the other functions, and which includes both the `getHTMLPage.h` file and your own `ParseURL.h` file containing the two functions. You should of course also include both the `getHTMLPage.c` file and the `ParseURL.c` file with your submission as well.

Zip and ship your complete code to me using the link in CANVAS no later than December 7th, at midnight. Your file name should have the following format:

`CST8234_Assignment3_`*`YourLastName`*`_`*`YourFirstName`*`.zip`

Notes

1. Nothing in this program requires that you use `malloc()`; everything can be held in predeclared character arrays, like `char buf[30]`.
2. To reinitialize an array to 0, you may find it useful to use the `memset()` function, which is part of the `string.h` library. To use `memset()` to reinitialize an array declared as `char buf[30]`, use
```
memset(buf, 0, sizeof(buf));
```
3. `getHTML()` prepends an information string on to the front of the source code returned, but it will not affect your program's ability to find URLs.
4. There are several web sites that list the contents of the `string.h` library. One such site is: http://www.tutorialspoint.com/c_standard_library/string_h.htm
5. You are NOT required to supply pseudocode for this assignment
6. The code contained `in getHTML()` is largely taken from the web site http://coding.debuntu.org/c-linux-socket-programming-tcp-simple-http-client, with modifications suggested in the book Internetworking with TCP/IP Linux/POSIX Sockets Version, Vol 3 by Comer and Stevens. While the code is fairly robust, it isn't 'bomb-proof', and considerably more code would be required to make it so. Consider yourself suitably warned.
7. You may assume that the user initially enters the starting domain and fragment strings correctly; no need to check for the authenticity of the input.