Chapter 1

Practical assignment 1

Note that all assignments consist of three sections: One that provides relevant background, another that provides the assignment itself and, finally, some remarks about assessment. Ensure that your submission addresses the issues discussed in the *assignment* part.

1.1 Background

When the web was invented, web pages were encoded as static HTML pages. In addition, web pages could be generated by CGI programs. As time progressed a number of mechanisms were developed to shift (some) execution from the web server to the browser. Examples include Java applets, JavaScript and a variety of other mechanisms to enable web servers to serve 'active' content. However, HTML remains the major notation used to encode content (and active content is typically incorporated into the HTML encoded pages). However, no browser-side active content should be used in any of the assignments of this module. Note that the use of CGI programs will be required in this assignment (and you are allowed to use CGI programs in any of the other assignments, when meaningful. (Most assignments will not be web-based, so CGI would only be an option in a few of the assignments.)

HTML has been revised several times and HTML 5 is the current standard. You are expected to use (correct) HTML 5 for all assignments where HTML is used.

Web pages may be served by general purpose servers, or servers designed for a specific application. Apache is (and has for a long time been) the most popular general purpose web server. Chapter B describes the manner in which a virtual computer could be instantiated for practical assignments in this module. After your virtual computer has been built it should be simple to install Apache on it (if not already present after initial installation of the selected Linux distribution). Any search engine will point one to further instructions — if required.

In order to use the HTTP server one typically simply has to copy the HTML files to the appropriate 'home' directory and the server will start serving those. To enable dynamic content the early web servers provided a mechanism that was (and still is) called CGI (*Common Gateway Interface*). A CGI program is a program that 'generates' output in the form of HTML; this HTML will then be sent to the browser, were it would be rendered.

A simple Hello World CGI program may look as follows (using some imaginary programming language):

```
print "<!DOCTYPE HTML>"
print "<HEAD>"
print "<TITLE>Example</TITLE>"
print "</HEAD>"
print "<BODY>"
print "<P>Hello world</P>"
print "</BODY>"
print </hHTML>"
end
```

If you run this program on a console, it will simply print out the marked up "Hello world" lines. However, when installed in the CGI directory of a web server, it will 'print' its output to the pipe that connects the web server to the browser, and the page will be rendered by the web browser. For it to work, the program should be executable code and the web server must be authorised to execute it. Since the CGI and 'home' directories are not always treated equally, you may need a file called `index.htm`, `index.html`, or similar suitable 'start' file that will be displayed when you open the home directory of the server. This (static) file may include an `<a href="...">` link that points to the CGI program. When one clicks on the link, the CGI program will be executed.

Note that a CGI program may also 'print' `<a href="..."` lines — just as if they were embedded in a static file.

Of course, writing such a program that simply produces static content is pretty useless; it is intended to be used in a context where what it outputs will change depending on current conditions. As an example, the CGI program may be connected to a database that operates as a back-end and that, for example, contains the types and numbers of items a merchant has in stock. Running this program will then not display static data, but the current stock levels of the merchant.

More generally, the CGI program may need some inputs so that one may, for example, query the stock level of some specific item. However, in this assignment we will not assume that one can provide the CGI program with input. Another simplifying assumption for this assignment is that the back-end will consist of a program (without any need for a database). Our intention is that you should demonstrate that

- You are able to install and use the web server;
- You can install one or more simple static pages in the appropriate server directory;
- You can install one or more executable CGI programs in the appropriate directory; and
- Get it all to work via a browser.

You may use any language to write your CGI programs. Run them from the console / command line to test them. As an example, if your CGI program is called `test`, run `./test` from the command line; you should see the output that should look like something encoded in HTML. Better still, execute `./test > test.htm` and the output will be redirected to `test.htm`; open `test.htm` in your favourite browser and the expected output should be rendered properly. Also consider validating `test.htm` using

`https://validator.w3.org/`

to be sure that your program generates valid HTTP. (It is a good idea to also validate any static HTML files you create for this — and other — assignments.)

While you may use any programming language to develop your CGI program, 'conventional' languages such as Java, C++, Pascal or COBOL are preferred. You do not need any network functionality in the programming language; you just need the ability to write text to the screen. For that reason, even a language such as COBOL would work. (COBOL is not recommended for subsequent assignments, though.)

Recall that the 'back-end' of your system will consist of a simple data file in an appropriate directory and with an appropriate initial value. Remember to create it once you have read the remainder of the assignment.

The assignment focusses on so-called server-side execution, so you are not allowed to execute any code in the browser — hence no JavaScript or other client-side software is allowed.

Package the files for this assignment (static HTML and executable CGI programs) in an archive that will install the files into the correct directory irrespective of the current directory from which the archive is unpacked. This requires you to use `tar`, `tgz` and/or `zip` to package the files using absolute paths.[1] The correct absolute paths may be changed in the Apache configuration file(s), but you are expected to use the default locations from the Linux distribution you use for this assignment. (If your CGI programs are compiled, let the archive unpack your source code to your home directory — or a suitable subdirectory in your home directory.)

Note that the intention is not to show your prowess to build sites. It is intended to show that you are able to achieve the goals set out above A tiny site will thus suffice.

1.2 Your assignment

As you are probably aware, a Fibonacci sequence is a sequence of numbers such that $f_n = f_{n-1} + f_{n-2}$ with $f_0 = 0$ and $f_1 = 1$.

For this assignment you are expected to write two CGI programs. (They will be so similar that it may be easier to write the one and then create the second from a copy.)

One of the programs will read three numbers from a text file. Those three values are initialised by you, the programmer, to be Fibonacci numbers. When the program runs, it reads the three numbers in the file, adds the second and the third number to produce a new number in the Fibonacci sequence, overwrites the numbers in the file with the second, third and new number and also displays these three new numbers. In essence, this program is a ‘next’ program in the sense that it displays the next set of three Fibonacci numbers. When it is executed the first time, it may display 0, 1, 1 (because $f_0 = 0$, $f_1 = 1$ and $f_2 = 1$). If one executes the program n times in sequence the values of f_{n-1}, f_n and f_{n-1} would be displayed. (Given this behaviour the file should obviously not be initialised to 0, 1, 1, because the first execution would then display 1, 1, 2, which is not what is expected.)

When we say that the program *displays* the three numbers, we mean that the program outputs text that forms valid HTML that would be rendered by a browser as a webpage with these three numbers. Additionally, the webpage should contain two hyperlinks: One would display the word ‘Next’ and be linked to this program that has been described thus far. When the program is installed in the correct directory and first invoked (possibly from a static initial webpage), it would cause 0, 1, 1 to be displayed in the browser window. Successively clicking ‘Next’ would (each time) cause the program to display the next Fibonacci triple.

[1]Many students have in the past struggled to figure out how to use absolute paths. Figure this out early, because it is really simple once you are aware of the concept!

There may be one tiny catch. If your browser caches the output from your CGI program it may think that it already knows the page that should be displayed, without executing the CGI program each time. Consider whether you need a metadata tag in your webpage (or some similar mechanism) that prevents such caching.

The other hyperlink on the webpage that your program generates should be called 'Previous' and be linked to the second CGI program you are expected to write.

This second CGI program is almost identical to the first program described above. However, when it reads (x, y, z) from the file, it overwrites the file with $(y - x, x, y)$ and displays this latter triple in the browser's window. It also displays the two hyperlinks that the first program displays.

There are interesting boundary cases. If the file contains $(0, 1, 1)$ and 'Previous is clicked, it may make sense to 'go to' $(---, 0, 1)$. Arguably, 'Previous' cannot be clicked again (and ideally should not even be listed as a clickable link). Does it make sense to consider some boundary case at the high end, where 'Next' does not make sense anymore?

To create a webpage that literally consists of three tiny numbers and two tiny hyperlinks would meet the requirements, but little effort is required to make it a bit more aesthetically pleasing. However, always remember that this module is about networking and aesthetics will not count much — if anything at all. However, a display that is really bland may or may not be commensurate with your self-image. That being said, spending more than about 10 minutes on aesthetics would be a waste.

A more meaningful question would be the following: Does one really need two programs when they are so similar? It would be easier to maintain one program that accepts a parameter. Given, say, a + moves it forward and a - moves it backwards.

One simple solution would be to write this program that is, say, called `fib`. Then create two scripts called `next` and `prev`.

`next` may look at follows:

```
!bash
fib +
```

The `prev` script looks almost identical, with just one character changed. Once these scripts are placed in the CGI directory, they should be made executable (with `chmod`) that that the web server is authorised to execute it.

It seems an even better solution would be to simply make the parameter part of the hyperlink linked to the `fib` program. This is indeed the case. However, the challenge is: how does one obtain the parameter from within the `fib` program? Yes, there are library functions that enable you to extract such parameters. In many scripting languages the ability to retrieve parameters is built in. However, **recall the prohibition of using any functions or built-in features to achieve network functionality** that applies to this module.

1.3 Assessment

This assignment — like most of those that will follow — will count out of 10. If you manage to execute your assignment perfectly, you will be awarded 10. (This will not be true for subsequent assignments). However, marks will be deducted for aspects that do not work correctly; examples include files that are installed in incorrect directories from your archive, a web server that does not serve the pages correctly, or output that is clearly wrong.

Warning

Using any functions or built-in features in this module to achieve network functionality will lead to a mark of 0 for the assignment. The only exception is using functions to open and close network sockets.

Warning

This assignment deals with server-side processing. **Using client-side computation (for example, using JavaScript) would lead to a mark of 0 for this assignment (and will be heavily penalised in any other assignment in this module).**