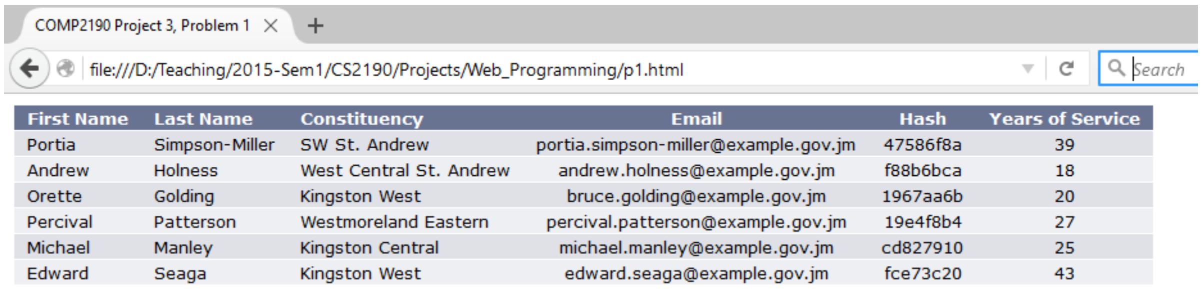
Web Programming

Assigned: Nov. 18, 2015
Due: Nov. 27, 2015 @2355

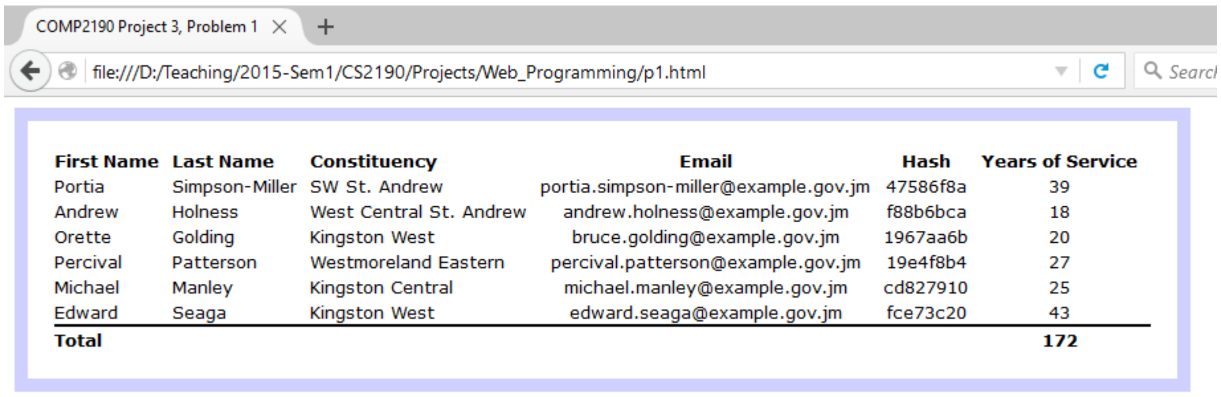
Problem 1 (10 points)

Create a single HTML document that presents two different appearances, determined by the document's CSS stylesheet. Your HTML file should be called p1.html and the two stylesheets should be called p1a.css and p1b.css. If the HTML file links to p1a.css then it should appear like this ("Version A"), assuming you are running Firefox 42.0 on a Windows 10 machine:


COMP2190 Project 3, Problem 1
file:///D:/Teaching/2015-Sem1/CS2190/Projects/Web_Programming/p1.html
Search

First Name	Last Name	Constituency	Email	Hash	Years of Service
Portia	Simpson-Miller	SW St. Andrew	portia.simpson-miller@example.gov.jm	47586f8a	39
Andrew	Holness	West Central St. Andrew	andrew.holness@example.gov.jm	f88b6bca	18
Orette	Golding	Kingston West	bruce.golding@example.gov.jm	1967aa6b	20
Percival	Patterson	Westmoreland Eastern	percival.patterson@example.gov.jm	19e4f8b4	27
Michael	Manley	Kingston Central	michael.manley@example.gov.jm	cd827910	25
Edward	Seaga	Kingston West	edward.seaga@example.gov.jm	fce73c20	43

If the HTML file links to p1b.css then it should appear like this in Firefox 42.0 on Windows 10 ("Version B"):


COMP2190 Project 3, Problem 1
file:///D:/Teaching/2015-Sem1/CS2190/Projects/Web_Programming/p1.html
Search

First Name	Last Name	Constituency	Email	Hash	Years of Service
Portia	Simpson-Miller	SW St. Andrew	portia.simpson-miller@example.gov.jm	47586f8a	39
Andrew	Holness	West Central St. Andrew	andrew.holness@example.gov.jm	f88b6bca	18
Orette	Golding	Kingston West	bruce.golding@example.gov.jm	1967aa6b	20
Percival	Patterson	Westmoreland Eastern	percival.patterson@example.gov.jm	19e4f8b4	27
Michael	Manley	Kingston Central	michael.manley@example.gov.jm	cd827910	25
Edward	Seaga	Kingston West	edward.seaga@example.gov.jm	fce73c20	43
Total					**172**

Here are some additional details and requirements for this problem:

- The content should be described in a single HTML file, using a <table> element to display the main table.
- There may not be any formatting information in the HTML file other than class and id attributes.
- The appearance must be generated entirely with CSS style information; you may not use images or Javascript for this problem.

- The only change that should be required to switch from Version A to Version B is to change the <link> element in the header to refer to a different CSS file.
- Your CSS files must appear in a directory called styles.
- Try to duplicate the appearance in the images above exactly ("pixel perfect"). For example:
    - Some of the columns should be centered whereas others are left-justified.
    - The "Total" line appears only in Version B (hint: you may find the display attribute useful in producing this effect).
    - The title in the browser title bar should read "CS 2190 Project 3, Problem 1".
    - Both versions use the Tahoma font in a 13-pixel size.
    - When the mouse moves over an email address, the text color should change to #cc6600.
    - The background color for the header row in Version A is #687290.
    - The background colors for the body rows in Version A are #eeeff4 and #dfe1e8.
    - The white lines between rows in Version A are 1 pixel wide.
    - The color for the frame around Version B is #d0d0ff.
    - The frame in Version B is 10 pixels wide; there are 20 pixels of empty space on either side of the frame.
    - The horizontal rule above the "Total" line in Version B is 2 pixels wide.
    - Match the paddings and spacings as closely as possible.
- Your HTML file must be a valid XHTML document that passes validation at http://validator.w3.org, and your CSS files must pass validation at http://jigsaw.w3.org/css-validator/
- Note: the border and margin styles are not supported for <tr> elements; <td> elements support border but not margin.

Problem 2 (15 points)

Write the XHTML code to create a form with the following capabilities:

- Text widgets to collect the first name, last name, constituency, email, years of service, password, and password confirmation.
- You should also include a hidden field with the value “`6eb6ac241942dc7226aeb`”
- When the Submit button the form is clicked, a JavaScript validation function must be run. This function should ensure that
    1. None of the fields listed above is empty.
    2. The email address contains at least one letter to the left of the ‘@’ symbol, at least one character immediately after the ‘@’ symbol, then a period, and then at least one character after the period. The following regular expression may prove useful as you write this function: `/\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i`.
    3. Years of service is a number between 0 and 50.
    4. The values of Password and Password confirmation match.
- The title in the browser title bar should read “MP creation form”
- The form should use the Tahoma font in a 13-pixel size.

- If an error is detected in a field during validation, that field should be colored in red.
- Your HTML file must be a valid XHTML document that passes validation at http://validator.w3.org, and your CSS files must pass validation at http://jigsaw.w3.org/css-validator/

Problem 3 (35 points)

Write a PHP script that collects the data from the form described above and store all the fields, except for the password field in the database. Later we will discuss how your script should handle the password field. Your code should verify that the data is valid before storing it. Valid data means that the email address is valid as discussed in Problem 2, the first name, last name, and constituency fields are not empty and only contain alpha characters—i.e., [A-Za-z] —spaces, and hyphens. Finally, the password and password confirmation fields must match even though we will not store them in the database.

In general it is not secure to store passwords directly in a database. Someone who is able to read the database, e.g., a rogue system administrator can easily retrieve all of the passwords for all users. A better approach is to apply a message digest function, e.g., MD5, to each password, and store only the message digest in the database. MD5 takes a string such as a password as input and produces a 32-character string of hex digits (called a message digest) as output. As we have already seen, the output provides a unique 'fingerprint' for the input string (there is no known way to produce two different strings with the same digest); second, given a message digest, there is no known way to produce a string that will generate that digest. You will take the password string submitted by each user, invoke PHP's MD5 function to compute the digest and store only the digest to the database. Once this is done, you can discard the password. With this approach you can make sure that a user enters the correct password when logging in, but if someone reads the digests from the database they cannot use that information to log in.

The approach of the previous paragraph has one remaining flaw. Suppose an attacker gets a copy of the database containing the digests. Since the MD5 function is public, they can employ a fast dictionary attack to guess common passwords. To do this, the attacker takes each word from a dictionary and computes its digest using SHA-1. Then the attacker checks each digest in the database against the digests in the dictionary (this can be done very quickly by putting all of the dictionary digests in a hash table). If any user has chosen a simple dictionary word as their password, the attacker can guess it quickly.

In order to make dictionary attacks more difficult, one must use password salting. When a user sets his/her password, compute a random number and concatenate it with the password before computing the MD5 digest (the `mt_rand()` function will generate a random number). The random number is called a *salt*. Store both the salt and the digest in the database. To check a password during login, retrieve the salt from the database, concatenate it to the password typed by the user, and compute the digest of this string for comparison with the digest in the database. With this approach, an attacker who has gained access to the login database cannot use the simple dictionary attack described above; the digest of a dictionary word would need to include the salt for a particular account, which means that the attacker would need to recompute all of the dictionary digests for every distinct account in the database. This makes dictionary attacks more expensive.

After each new and valid MP is submitted, print the contents of the database to a table that is formatted as in Problem 1-a. The database for this problem should be named MPMgmtDB. Use the database username “comp2190SA” with the password “2015Sem1”. The MPMgmtDB should contain a table called Representatives. The fields of the table are as follows:

Field	**Data type**
first_name	String
last_name	String
Constituency	String
Email	String
Yrs_service	Integer
Password_Digest	String
Salt	String
ID	Integer, autonumber

Problem 4 (Optional: 10 extra-credit points)

Create a login form for your application. Your form should have a text field for the email address, a password entry field, a hidden field, and a submit button. In the PHP to generate this login form, set the value of the hidden field to the MD5 hash of the session ID. To log the user in use the salt as described above.

What to turn in

1. You will hand in all of your XHTML, CSS, Javascript, and PHP files. All of your CSS files must appear in a directory called ‘styles’, your Javascript files in a directory called ‘scripts’.
2. A separate (typed) document of a page or so, describing the overall program design for Problem 3, a verbal description of “how it works,” and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
3. A separate description of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.

Grading

- XHTML and CSS files for Problem 1.
  - Page displays correctly as specified in assignment sheet – 7 points
  - Pass validation – 3 points
- XHTML, CSS and Javascript files for Problem 2.
  - Form displays correctly as specified in assignment sheet – 5 points

    - Validation function works as specified on assignment sheet – 7 points
    - Passes validation – 3 points
- Program listing for Problem 3
    - Works correctly as specified in assignment sheet – 20 points
    - Contains relevant in-line documentation – 3 points
    - Elegant code, i.e., proper decomposition of functionality – 2 points
- Design document
    - Description – 5 points
    - Thoroughness of test cases – 5 points

Acknowledgements

The text on salting and hashing is based on some text developed by John Ousterhout for his CS142 course.