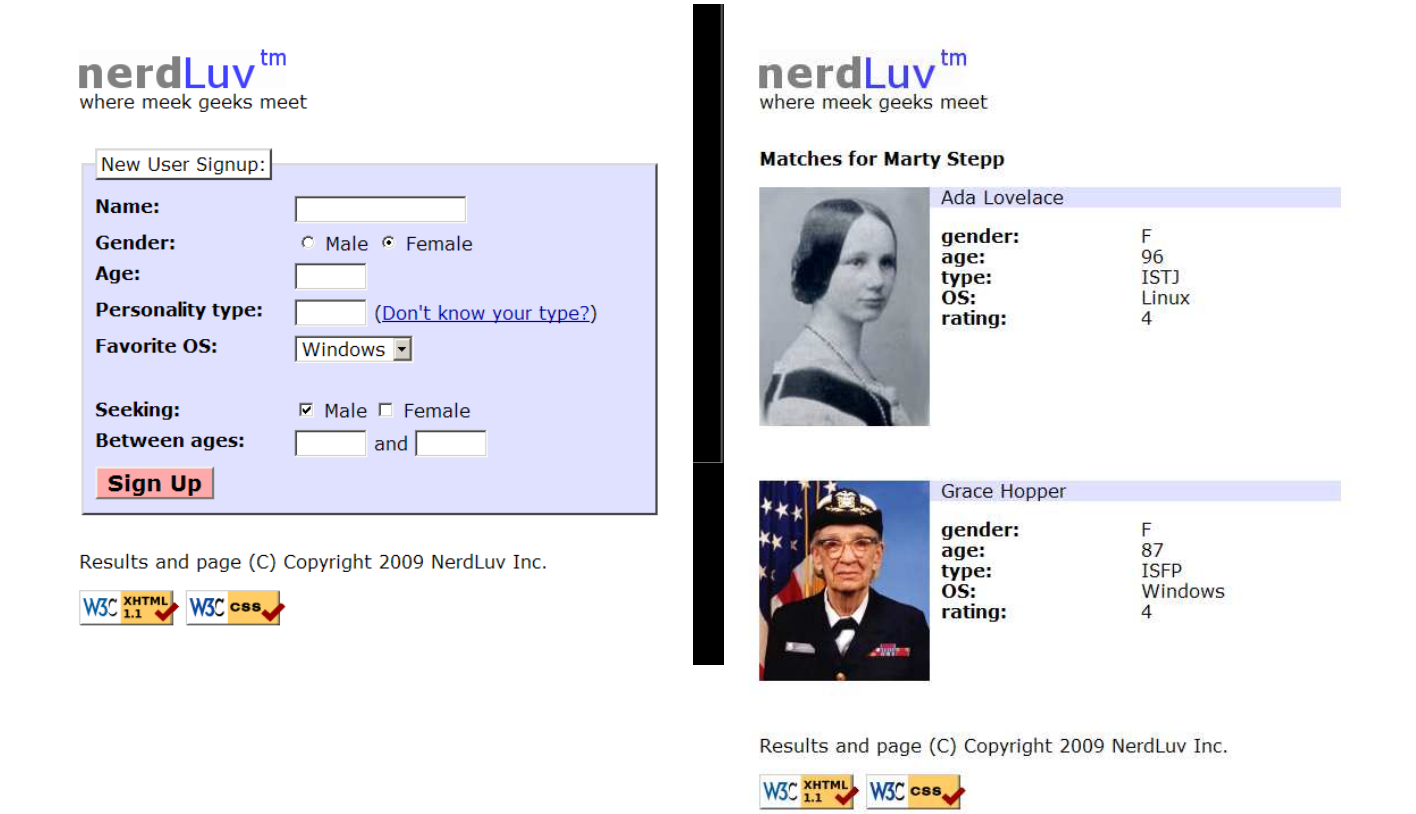
University of Washington, CSE 190 M, Spring 2009
Homework Assignment 4: NerdLuv

due Wednesday, April 29, 2009, 11:30pm electronically

This assignment is about processing HTML forms with PHP. You will write the following two pages:



Online dating has become mainstream with popular sites such as eHarmony, match.com, OkCupid, and Plenty of Fish. Your task for this assignment is to write HTML and PHP code for a fictional online dating site for desperate single geeks, called NerdLuv.com. The site will consist of two pages: a front page called `index.php` (skeleton provided to you) that contains a form for a user to submit information, and a result page called `results.php` (that you will write entirely) that displays single people who match with the applicant.

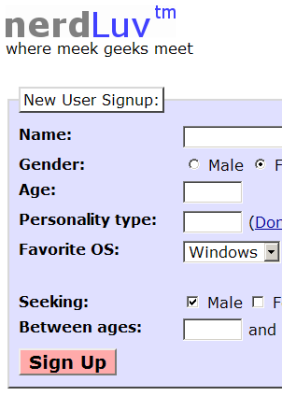
Turn in the following files:

- `index.php`, the front signup page *(skeleton provided)*
- `results.php`, the results page *(no starter code provided; write this yourself)*
- `nerdluv.css`, the CSS style sheet for both pages *(skeleton provided)*
- `common.php`, a file containing any code you want to share between pages *(optional)*

Part of your grade comes from uploading your files to Webster at the following URL. Please do not place a solution to this assignment online on a publicly accessible (un-passworded) web site.

`https://webster.cs.washington.edu/`**`your_uw_netid`**`/hw4/`

Signup Page (`index.php`):



The `index.php` page begins with the NerdLuv logo image, `nerdluv.png`. The text "where meek geeks meet" appears under the image. Below this text is a "New User Signup" form, which allows a user to join the dating site.

The overall set of form fields is 32em wide, 1em from the left edge of the overall page, and has a background color of `#E0E0FF` and a 2px-thick gray border in the "outset" style. Each field set's label has a white background, and a 2px-thick gray border in the "outset" style, with 0.2em of padding around the label.

The "New User Signup" form contains the following elements. Each element has a text label to its left that occupies 11em in width. The elements line up into columns.

- **Name:** A label and 16-letter box for the user to type a name. Initially empty.
- **Gender:** Radio buttons allowing the user to select a gender of Male or Female. When the user clicks the text, its radio button should become checked. Initially Female is checked.
- **Age:** A label and a 5-letter-wide text input box allowing the user type his/her age in years. The box should allow the user to type up to 2 characters. Initially empty.
- **Personality type:** A 5-character-wide text box allowing the user to type his/her Keirsey personality type, such as ISTJ or ENFP. The box should allow the user to type up to 4 characters. The label also contains a link to http://www.humanmetrics.com/cgi-win/JTypes2.asp . Initially empty.
- **Favorite OS:** A drop-down selection box allowing the user to select his/her computer's operating system. The choices are Windows, Mac OS X, Linux, and other. Initially "Windows" is selected.
- **Seeking:** Two checkboxes allowing the user to select the genders he/she is interested in meeting. It is legal for an applicant to check either or both boxes. Initially only Male is checked. When the user clicks the text next to a checkbox, the nearby checkbox should check or uncheck.
- **Between ages:** Two 5-character-wide text boxes for the user to specify the range of acceptable ages of partners. The box should allow the user to type up to 2 characters. Initially both are empty.
- **Sign Up:** When pressed, submits the form to `results.php` for processing. Has a background color of `#FFAAAA`, and its text is bold and 120% as large as normal text on the page.

Results Page (`results.php`):

The `results.php` receives input from `index.php` when the New User Signup" button is clicked. It expects query parameters representing the data from `index.php`, such as the new applicant's name, gender, and so on. (The exact names/values of those parameters are up to you.) Your results page should save the applicant's data to a file (described later) and display all users that match the applicant's preferences.

Unless you are completing the server-side validation extra feature described later, you may assume that the form data submitted is valid. For example, assume that no fields are left blank or contain illegal characters. You may also assume that no user will resubmit data for a name already stored in the data file.

The page begins with the same `nerdluv.png` banner image and slogan as `index.php`. 2em below the banner slogan should be a heading saying "Matches for *User Name*". Below this is a list of people that match the new user. A "match" is a person in the data set with two qualities when compared with the new user. The first is a **compatible gender and "seeking" value**. For example, if the new user is a male seeking females, list only females seeking males (and females seeking both men and women). The second required quality is a "strength of match" rating of at least **3 points**. Points are earned as follows:

- +1 point if the users are of compatible ages; that is, if the user's age is greater than or equal to the applicant's minimum age and less than or equal to the applicant's maximum age, and the applicant's age is between the user's min/max ages inclusive. Note: The implication of this is that **the min/max ages**

are not hard limits. Being within them increases an applicant's match score, but an applicant outside the age range who otherwise has a high enough score may still be shown.
- +2 points for having the same favorite operating system. For example, if the single prefers Mac OS X and the applicant also prefers Mac OS X, that single's "strength of match" gets +2 points.
- +1 point for having a dimension of personality type that matches the same dimension for the applicant. For example, the types ISTP and ESFP score +2 points because the S and P match.

Each matching single should be displayed in a section 32em wide, 1em from the left edge of the page. First is an image of the match, shown with a width of 150px. To the right of the image should be the person's name on a line with a background color of `#E0E0FF`. Below this is text about the match, including gender, age, personality type, OS, and their "strength of match" rating. There is 3em of space between matches.

Both pages have a title of `NerdLuv` and a "favorites icon" of `heart.gif`. At the bottom of both pages is the standard pair of W3C validator buttons. Screenshots in this document are from Windows XP in Firefox 3, which may differ from your system.

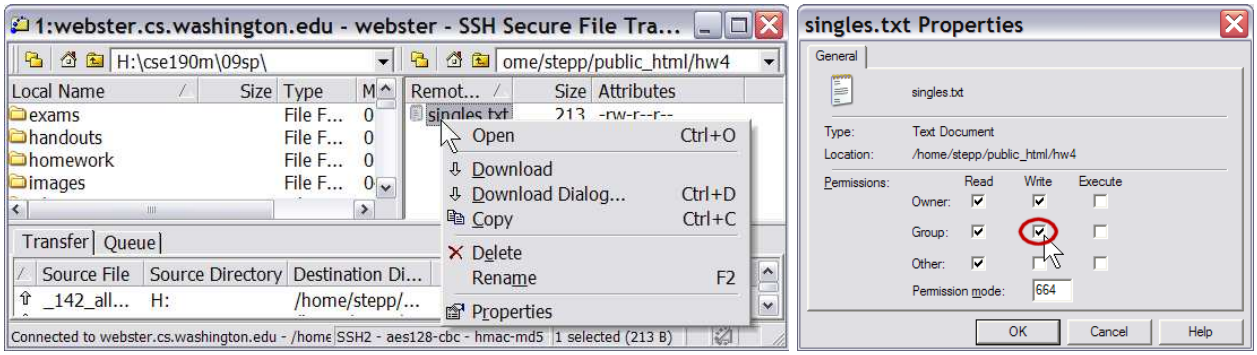
Data File (`singles.txt`):

Your `results.php` page reads its data about singles from a file named `singles.txt`, which is placed in the same folder as the script itself. This file contains data records in exactly the following format, where each line contains the user's name, gender (M or F), age, Keirsey type, favorite operating system, seeking gender(s) (M, F, or MF), and min/max seeking age, separated by commas. For example:

```
Ada Lovelace,F,96,ISTJ,Linux,M,59,99
Donald Knuth,M,70,INTJ,other,MF,12,17
Marty Stepp,M,29,ISTJ,Windows,F,18,39
```

Each time the script is run, it should read this file's contents to find the singles that match the applicant based on the above criteria. When a new user signs up, the user's line of information is added to the end of the file. You may want to look at the PHP `file_put_contents` function in the book or Lecture 8 slides.

When you initially upload your files to Webster, you may need to change the file permissions on `singles.txt` so that PHP is able to write changes to this file. In your SSH Tectia window, right-click `singles.txt` in the right-side pane and choose **Properties**. In the Properties window, enable Group Write permission by checking the box shown in the screenshot below.



The image for each single is stored in a file in the `images` subfolder relative to your PHP files. The image for a single is a file with the `.jpg` extension whose name is equal to the person's name in lowercase with all spaces converted into underscores. For example, the image for Bill Gates is stored in `bill_gates.jpg`, and the image for Rosie O Donnell is stored in `rosie_o_donnell.jpg`.

A default set of user data and image files will be provided on the course web site. Some users may not have images; in particular, any user who submits new data into the app will not have an image (unless you choose to add that as an extra feature). To deal with such cases, your `results.php` code should test whether each user's image file exists, and if it doesn't, you should instead display the image `default_user.jpg`.

Extra Features:

In addition to the previous requirements, you must also complete **at least one of the following additional features**. If you want to complete more than one, that is fine, but only one is required.

1. **server-side form validation**: Add PHP code in `results.php` that tests all query parameters received for validity. Use regular expressions as appropriate to do this. Specifically:
   - The name must not be blank.
   - The age submitted must be an integer and must be between 0 and 99 inclusive.
   - The gender submitted must be male or female. (No other values such as `"robot"` are allowed.)
   - The Keirsey personality type must be a 4-letter string whose letters come from the Keirsey personality dimensions: I or E for the 1st letter, N or S 2nd, F or T 3rd, and J or P 4th.
   - The favorite operating system must come from the choices provided.
   - The "seeking" gender submitted must be male, female, or both. (It is not valid to leave both blank or to submit other values.)
   - The seeking minimum and maximum ages submitted must be integers, must be between 0 and 99 inclusive, and the minimum seeking age must be less than or equal to the maximum.

   If some of the form data is invalid, instead of showing matches, the response page should contain an error message displaying what was wrong. For example, if neither "seeking" box is checked, suitable messages would be, "Error: Invalid submission data." or, "Error: You forgot to submit a value for seeking." No data should be saved if the submission is invalid, and no matches should be shown.

2. **ability to upload photos**: Modify the `index.php` page so that it allows the applicant to submit a photo. To handle uploading of photos, add a file input box to the `index.php` page that allows the user to browse for a file to upload as a photo. (A screenshot is provided on the course web site.) When the `results.php` script receives a POST request, it will save this photo in the `images` subfolder with the others using the file naming system described previously (the user's name with spaces replaced by underscores, such as `dick_cheney.jpg`) and show it when subsequent users search for matches. You may assume that every user submits a valid JPEG file.
3. **ability for returning users to view their matches**: Right now you see your matches when you sign up for NerdLuv, but there's no way to come back to NerdLuv and view your latest matches without signing up again. Add a second form to the top of the page for Returning Users, where all the user needs to type in is their name. When this form is submitted, it will show all matches for the user without writing any new data to the `singles.txt` data file.

   Note that in this case the user is just getting existing information from the server, not submitting any new data to be saved to the server. You should take this into consideration when choosing what type of request method to use on each form. If the user submits a name that is not found in the file, you should display an error message and not display any matches.

4. **Person class (OOP):** We will soon talk about how to write object-oriented PHP code with classes. This assignment's code can be written in a cleaner way if you make a class to represent each person in the data set. As an optional extra feature, create a file `Person.php` containing a Person class, and use it throughout your other file(s) as appropriate. Each object should contain all relevant data about a given person, as well as any methods closely related to that data. For example, you could put the logic for validating data or for computing strength of match ratings into this class.

Near the top of your HTML file, put a comment saying which extra feature(s) you have completed. If you implement more than one, also comment which one you want us to grade (the others will be ignored for grading). Regardless of how many additions you implement, the main program behavior and appearance should still work as specified. If you have a different idea for a creative addition to this program, please ask us and we may approve it for credit as a substitution for one of the above.

Hints and Suggestions:

We suggest the following development strategy for this assignment:

1. Modify `index.php` code to use forms. Make it submit to a testing page such as our params.php.
2. Write a basic `results.php` that doesn't actually save the new user; instead, just read the input file and always display all people regardless of whether they match well with the new user.
3. Modify `results.php` to save the new user to the file and to show only people that match the new user.
4. Add extra features.

Remember that controls must have `name` attributes. Sometimes you must also add a `value` attribute to a control to affect how it is submitted. Test your form by setting its `action` attribute to our `params.php`.

When working on the results page, if things don't look the way you expect, try *View Source* on the page. We also suggest that you use `print` statements to display which aspects of the user lead to "strength of match" points. Also consider using `print_r` function to print the contents of arrays for debugging, such as `$_REQUEST`.

Submission and Grading:

Submit your assignment online from the course web site. For reference, our `results.php` solution is roughly 130 lines long, though you do not need to match this exactly.

Your PHP code should generate no error or warning messages when run using reasonable sets of parameters. Decompose your PHP code into functions as appropriate, minimize global variables, utilize parameters/returns properly, correctly use indentation/spacing, and avoid long PHP lines over 100 characters. Use material that discussed during the first four weeks of class or the first six chapters of the textbook.

One grading focus in this assignment is avoiding redundancy. Use loops, functions, variables, `if/else` factoring, etc. to ensure that your code is not redundant. In addition, you may find that sections of text or code are shared between your PHP pages. If so, you should put any such text/code into the file `common.php` and include the contents of this file in both of your pages using the `include` function.

Another grading focus is commenting. Put a descriptive header at the top of your code, and a well-written comment on each major section of code and function explaining in detail what it is doing.

For full credit, you should reduce the amount of large complex chunks of PHP code injected into the middle of HTML code. When possible, replace such chunks with functions that are called in the HTML and declared at the top or bottom of your file. Also minimize the use of `print` and `echo` statements. As much as possible you should insert dynamic content into the page using PHP expression blocks as taught in class.

All of your HTML code and PHP output should use valid XHTML 1.1 as taught in class. For full credit, your pages must successfully pass the W3C XHTML validator. (Not the PHP source code itself, but the HTML output it generates, must validate.) Do not use HTML tables on this assignment.

Since we are using forms on this assignment, you should properly choose between GET and POST requests for submitting data. You should also choose proper controls and set their attributes accordingly.

Properly use whitespace and indent your XHTML and PHP code following examples shown in class. Do not place more than one block element on the same line or begin any block element past the 100th character on a line. The source code output by your PHP pages will be graded by the same criteria as normal HTML code, meaning that it should be valid XHTML, should be indented, and so on.

CSS is not a major grading focus of this assignment, but your CSS code should pass the W3C validator. Also do not express stylistic information in HTML, such as inline styles or presentational HTML tags.

Please do not place a solution to this assignment online on a publicly accessible web site.