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

This assignment is about making a simple multi-page "online dating" site that processes HTML forms with PHP. **Online dating** has become mainstream with popular sites such as eHarmony, match.com, OkCupid, Chemistry, 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**. Turn in the following **two files**:

- **signup.php**, a page with a form that the user can use to sign up for a new account
- **matches.php**, a page with a form for existing users to log in and check their dating matches

There are some **provided files** on the web site. The first is a complete version of the site's front page, **index.php**. This front page simply links to your two pages. The second complete provided file is **common.php**, which contains common header/footer HTML code that you should include in your two pages. We also provide a complete CSS file **nerdluv.css** with all of the page styles. You should link to this CSS file from all of your pages and use its styles in your code. You should be able to fully style all pages using the styles in **nerdluv.css** only.

Index Page (index.php) and Overall Site Navigation:
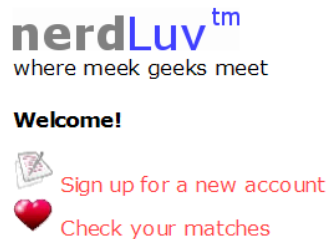
The provided **index.php** page has a header logo, links to **signup.php** and **matches.php**, and footer notes/images. You do not need to modify this file, but you should put it in the same folder with your other files and upload it to Webster with your files.

The "Sign Up" link leads to **signup.php** (left below), and "Check matches" to **matches.php** (right below):

When submitted, the View Matches form looks like this:

When submitted, the Signup page looks like this:

The details about each page's contents and behavior are described on the following pages. Screenshots in this document are from Windows XP in Firefox 3.5, which may differ from your system.

Sign-Up Page (signup.php):

The **signup.php** page has a header logo, a **form** to create a new account, and footer notes/images. You must write the HTML code for the form. The form contains the following labeled fields:

- **Name:** A 16-character box for the user to type a name.

- **Gender:** Radio buttons for the user to select a gender of Male or Female. When the user clicks the text next to a radio button, that button should become checked. Initially Female is checked.

- **Age:** A 5-letter-wide text input box for the user to type his/her age in years. The box should allow typing up to 2 characters.

- **Personality type:** A 5-character-wide text box allowing the user to type a Keirsey personality type, such as ISTJ or ENFP. The box should let the user type up to 4 characters. The label has a link to http://www.humanmetrics.com/cgi-win/JTypes2.asp .

- **Favorite OS:** A drop-down select box allowing the user to select a favorite operating system. The choices are Windows, Mac OS X, and Linux. Initially "Windows" is selected.

- **Seeking age:** 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 in each box. Initially both are empty.

- **Sign Up:** When pressed, submits the form for processing as described below.

Submitting the Sign-Up Form:

When the user presses "Sign Up," the form should **submit** its data as a POST. (The exact names/values of the query parameter(s) are up to you.) The form should submit the data back to **signup.php** itself. Your **signup.php** code should therefore have two modes:

- If the browser is performing a normal **GET** request, your page should display the form as described above.

- If the user is submitting a **POST** of data, your PHP code should read the data from the query parameters and store it as described below. The resulting page has no form; instead, it has text thanking the user, with the usual header and footer. The text "log in to see your matches!" should link to **matches.php**.

Check for a GET or POST request in PHP with the variable `$_SERVER["REQUEST_METHOD"]` as described in Ch. 6.

Form Data:

Your site's user data is stored in a file **singles.txt**, placed in the same folder as your PHP files. We will provide you an initial version of this file. The file contains data records as lines in *exactly* the following format, with the user's name, gender (M or F), age, personality type, operating system, and min/max seeking age, separated by commas:

```
Angry Video Game Nerd,M,29,ISTJ,Mac OS X,1,99
Lara Croft,F,23,ENTP,Linux,18,30
Seven of Nine,F,40,ISTJ,Windows,12,50
```

Your **signup.php** page's POST response should create a line representing the new user's information and add it to the end of the file. See the PHP `file_put_contents` function in book Chapter 5 or the lecture slides.

In all pages, **assume valid data** for the file's contents and form submissions. For example, no fields will be left blank or contain illegal characters (such as a comma). No user will resubmit data for a name already in the system.

View Matches Page (matches.php):

nerdLuv ^{tm}

where meek geeks meet

Returning User:

Name:

View My Matches

This page is for single nerds to meet and date each other!
Type in your personal information and wait for the nerdly luv
to begin! Thank you for using our site.

Results and page (C) Copyright 2010 NerdLuv Inc.

Back to front page

W3C XHTML 1.1   W3C CSS

The **matches.php** page has a header logo, a **form** to log in and view the user's matches, and footer notes/images. You must write the HTML code for the form. The form has just one field:

- **Name:** A label and 16-letter box for the user to type a name. Initially empty. Submit to the server as a query parameter `name`.

When the user presses "View My Matches," the form **submits** its data as a GET request back to **matches.php** itself. The exact name of the query parameter sent should be `name`, and its exact value should be the encoded text typed by the user. For example, when the user views matches for Rosie O Donnell, the URL should be:

- **matches.php?name=Rosie+O+Donnell**

As with **signup.php**, this page operates in one of two "modes":

- If the browser is performing a GET request *without* a `name` query parameter, display the form described above.

- If the browser is performing a GET request *with* a `name` query parameter, do not show the form. Instead, your PHP code should search for users that match the user with the given name, as described below.

Viewing Matches:

nerdLuv ^{tm}

where meek geeks meet

Matches for Lara Croft

Anakin Skywalker

gender:	M
age:	27
type:	INTJ
OS:	Linux

Marty Stepp

gender:	M
age:	30
type:	ISTJ
OS:	Linux

This page is for single nerds to meet and date each other!
Type in your personal information and wait for the nerdly luv
to begin! Thank you for using our site.

Results and page (C) Copyright 2010 NerdLuv Inc.

Back to front page

W3C XHTML 1.1   W3C CSS

When viewing matches for a given user, your page should show a central area displaying each match. Write PHP code that reads the name from the page's query parameter and figures out which other singles match the given user. The existing singles to match against are records found in the file **singles.txt** as described previously. You may assume that the user name submitted is found in the input file.

Below the banner slogan should be a heading saying "Matches for (name)". Below this is a list of singles that match the given user. A "match" is a person in the data set with the following qualities:

- The **opposite gender** of the given user;

- Of **compatible ages**; that is, each person is between the other's minimum and maximum ages, inclusive;

- Has the **same favorite operating system** as this user;

- Shares **at least one personality type letter in common**. For example, types ISTP and ESFP have two in common (S and P).

As you find each match, you must emit the HTML to display the matches, in the order they were originally found in the file. Each match shows the image **user.jpg**, the person's name, and an unordered list with their gender, age, personality type, and OS.

http://www.cs.washington.edu/education/courses/cse190m/10su/homework/4/user.jpg

Styling:

The styles you need are already given to you in **nerdluv.css**, but you still need to use proper tags and `class` attributes to make sure they are applied. Be mindful of the styles on forms and form controls. On the course web site are several screenshots of the various pages. Make sure that your form has the same width, colors, fonts, borders, etc. as in these examples. If you choose the right tags to represent your form, it should match. Make sure that form fields line up in **columns** by using a `strong` tag or `column` class so that each text label floats to the left and is 11em wide.

In **matches.php**, the matches are displayed in a `div` with `class` of `match`. First is an image of the match, shown with a width of 150px. The paragraph that shows the person's name to the right of the image has a light blue background color. The section with the match's gender, age, etc. must be represented as an unordered list (`ul`).

Uploading and Testing:

Upload all files to Webster to test them; include **index.php** and **common.php** even though you won't modify them. You may need to change the **permissions** on **singles.txt** so that PHP can write changes to it. In SSH Tectia, right-click **singles.txt** in the right pane and choose Properties. Enable Group Write by checking the box shown below.
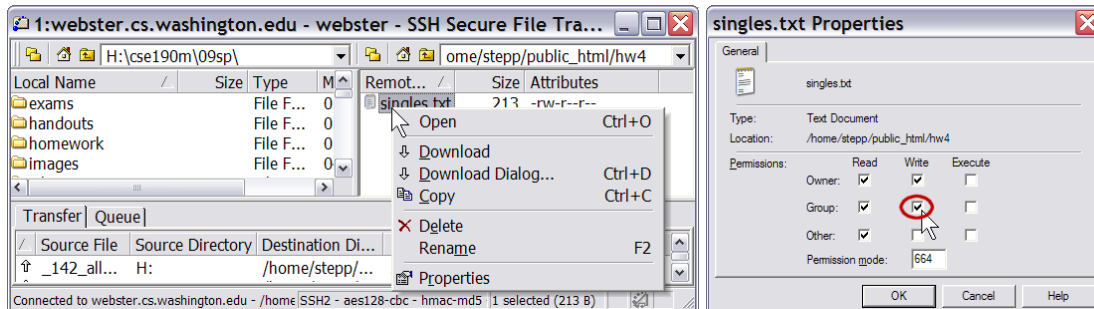
Suggested Development Strategy and Hints:
- Based on **index.php**, write **matches.php**, so that your site works properly for existing users.
  - Write an **initial version** that outputs *every* person, even ones who aren't compatible "matches." This way you can debug your file I/O, styles, etc. Then add checks like gender, age, and OS. Focus on the PHP code and behavior first, as opposed to style details (CSS is not worth many points on HW4).
- Write **signup.php**. If you finished the match page you'll understand forms, making the signup page easier. This page is tough; there are more form parameters to manage, and you must write to the file.

Use **debug print and print_r statements** to track down bugs. For example, you can `print_r($_REQUEST);` to see the query parameters submitted. Use **Firebug** and also **View Source** to find HTML output problems.

Form controls must have `name` attributes. Sometimes you must also add a `value` to affect how data is submitted. You can test a form by setting its `action` attribute to Webster's `params.php`, which prints debug information.

Implementation and Grading:

Your HTML output for all pages must pass the W3C **XHTML validator**. (Not the PHP source code itself, but the HTML output it generates.) Do not use HTML tables. Since we are using HTML **forms**, choose proper form controls and set their attributes accordingly. Properly choose between GET and POST requests for sending data.

Your PHP code should not cause errors or warnings. Minimize use of the `global` keyword, use indentation/spacing, and avoid lines over 100 characters. Use material from the first four weeks of class and the first six book chapters.

A major grading focus is **redundancy**. Use **functions, parameters/return, included files/code**, loops, variables, etc. to avoid redundancy. Some sections are shared redundantly between your PHP pages. These are in the provided file `common.php`. Include this file in both of your pages using the `include` function and make use of its contents..

For full credit, reduce the amount of large chunks of PHP code in the middle of HTML code. Replace such chunks with **functions** declared at the top or bottom of your file. You will also lose points if you use PHP `print` or `echo` statements. Insert dynamic content into the page using PHP **expression blocks**, `<?= ... ?>`, as taught in class.

Another grading focus is PHP **commenting**. We expect more comments here, similar to CSE 14x. Put a descriptive comment header at the top of each file, **each function**, and each section of PHP code.

Format your HTML and PHP code similarly to the examples from class. Properly use whitespace and indentation. Do not place more than one block element on a line or begin a block element past the 100th character.

Our solution uses this many lines: **signup.php** 80 (56 "substantive"); **matches.php** 80 (61 "substantive").

Please do not place a solution to this assignment online on a publicly accessible web site. Part of your grade will come from successfully uploading your files to the **Webster** server in the directory:

- **https://webster.cs.washington.edu/*your_uwnetid*/hw4/**