Computer Science 50 Fall 2009 Scribe Notes

Week 8 Wednesday: October 28, 2009
Andrew Sellergren

Contents

1 Announcements (0:00–5:00)

- This is CS 50.
- 0 new handouts.
- Don't forget office hours! This time of year, as the course starts accelerating and the material gets more difficult. We'll be diverting more hours toward earlier in the week since you all seem to have learned the lesson that starting the problem sets earlier rather than later is best practice! We realize, also, that OHs may not be the most nurturing environment for those of you struggling with the larger concepts rather than single lines of code. Do feel free to e-mail any of the staff directly or heads@cs50.net if you'd like to set up a one-on-one appointment.
- Dinner with David (and faculty) next Wednesday at 6 PM. Also Lunch with David next Friday at 1:15 PM. RSVP here.

1.1 Baby Steps

- Even though the problem sets have gotten larger, you will still benefit greatly from approaching them in bitesize chunks. For that very reason, the problem set specifications are written with checkboxes in place so that you can check off one task at a time.
- Problem Set 6, for example, asks you to implement a dictionary. However, more specifically, it asks you to implement `load()`, `unload()`, `size()`, and `check()`. Of those four functions, you almost certainly need to start with `load()`. But don't tackle a trie or hash table implementation all at once. Start by opening the dictionary file and just counting the characters in it, perhaps. At least you will be making forward progress.
- Even CS ninjas like David[1] take baby steps like this. When he was implementing HarvardTweets, for example, he first took on the task of amassing the tweets in a database. Then he implemented the skeleton of a webpage with markers that said "content here." Slowly but surely he moved toward the final goal.

2 Dynamic Web Development with PHP and SQL (5:00–75:00)

2.1 A Command-Line Stock Lookup Application

- One major drawback to the web pages we implemented last time is that they are all static. Their content is not going to change based on who is visiting the site or when he is visiting. Ultimately, static sites aren't nearly as interesting as dynamic sites.

[1](scoff)

- If we visit Yahoo's stocks site and look up Google's stock price, it's very easy for us as humans to view the source and find that stock value within the HTML. However, for a machine to do the same thing using screen-scraping is actually much more difficult, though not impossible.

- Actually, instead of screen-scraping Yahoo's stocks page, we have a much better option: downloading the data as *comma-separated values* (CSV). As the name implies, this simply means that each of the fields is separated by a comma. Generally speaking, this is much easier to parse than an entire web page. The data for GOOG might look like the following:

  ```
  "GOOG", 541.166,"10/28/2009","1:03pm"
  ```

  Even without knowing exactly what the fields are, we can infer that the first is the stock symbol, the second is probably the trade price, and the third and fourth are the date and time of the lookup. Incidentally, CSV files can be opened with programs like Excel, which will automatically display them as spreadsheets.

- Let's start by fetching that URL (which returns the CSV file) using PHP:

  ```
  <?
  $url =
    "http://download.finance.yahoo.com/d/quotes.csv?s=GOOG&f=sl1d1t1c1ohgv&e=.csv";

    $fp = fopen($url, "r");

    $row = fgetcsv($fp);

    print_r($row);
  ?>
  ```

  The `<?` and `?>` are open[2] and close tags which signal that the lines within should be parsed by the PHP interpreter. As well, we need to save our file with a `.php` file extension so that the server knows to pass the file to the PHP interpreter before serving it up to the browser.

- The syntax for opening in PHP is nearly identical to that in C. One difference is that variables in PHP begin with `$`. Also, they do not have type declarations before them. This *loose data typing* can be alternately an advantage and a disadvantage of PHP.

- The function `fgetcsv()` actually parses the CSV file for us. It will return an array containing the first row of the CSV file. If we pass this value to `print_r()`, we can see this array printed out.

[2]Actually this is the short open tag. The conventional long version is `<?php`.

- But how do we run a PHP program? As we said earlier, PHP requires an interpreter. Unlike C, PHP is not a compiled language, so at runtime, it must be passed to another program which will parse it and cause its instructions to be executed. On a Linux computer, this program is `php`, so we can run `php quote.php` from our command line to see the following:

```
Array
(
    [0] => GOOG
    [1] => 541.90
    [2] => 10/28/2009
    [3] => 1:08pm
    [4] => -6.39
    [5] => 546.51
    [6] => 550.00
    [7] => 539.50
    [8] => 1558790
)
```

- So now we know that the stock price is contained in the first index of the `$row` array, we can access it using syntax similar to C:

```
<?
$url =
  "http://download.finance.yahoo.com/d/quotes.csv?s=GOOG&f=sl1d1t1c1ohgv&e=.csv";

  $fp = fopen($url, "r");

  $row = fgetcsv($fp);

  print($row[1]);
  print("\n");
?>
```

- Okay, this is somewhat more dynamic than what we built last time, but it's only dynamic because the stock price itself is changing. What if we want to look up a stock other than Google?

- Notice in the `$url` variable this part of the string: `s=GOOG`. Recall from last time when we re-implemented Google that the actual query was passed to Google's servers in the URL. Recall also that it was named `q`. Here, it seems, we have something similar. The stock symbol is being passed to Yahoo via the URL. We can also see that the list of parameters begins with a `?` and each parameter is separated by an `&`. For now, we don't really know what the rest of the parameters in the URL actually mean.

- Instead of hardcoding in the value GOOG, then, let's try making the symbol dynamic at runtime by inserting a variable which takes the value of a command-line argument:

  ```
  <?

  $symbol = $argv[1];

  $url =
    "http://download.finance.yahoo.com/d/quotes.csv?s=$symbol&f=sl1d1t1c1ohgv&e=.csv";

    $fp = fopen($url, "r");

    $row = fgetcsv($fp);

    fclose($fp);

    print($row[1]);
    print("\n");
  ?>
  ```

  This `$argv` certainly looks familiar. Turns out the array of command-line arguments is named the same in PHP as it was in C. One other thing to note is that because double quotes are "magic" in PHP, we don't have to mess around with concatenation. We can simply place `$symbol` between the quotes and its value will automatically be inserted at runtime. This is called *interpolation.*

- Now we can run from the command line `php quote.php <symbol>`, where `<symbol>` is a valid stock symbol. The current stock price will then be printed out to stdout, like so:

  ```
  cs50@cs50.net (~): php quote.php GOOG
  541.58
  cs50@cs50.net (~): php quote.php MSFT
  28.2915
  cs50@cs50.net (~): php quote.php YHOO
  16.30
  ```

2.2 A Frosh IMs Registration Application

- Recall from last time that David, as a sophomore, undertook the task of implementing an electronic registration module for frosh IMs in order to replace the annoyingly inefficient paper-based system. The language he was using at the time was Perl, but today we can mimic his application using PHP. Take a look at froshims1.php.

- Looks like we have a text field, a checkbox, a radio button, and a select menu. Amazingly, pretty much all the user-driven websites out there rely on these simple HTML forms in order to get input from the user. Take a look at the source code:

```
<?
    /***************************************************************
     * froshims1.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.
     * Submits to register1.php.
     ***************************************************************/
?>

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    <div align="center">
      <h1>Register for Frosh IMs</h1>
      <br /><br />
      <form action="register1.php" method="post">
        <table border="0" style="text-align: left;">
          <tr>
            <td>Name:</td>
            <td><input name="name" type="text" /></td>
          </tr>
          <tr>
            <td>Captain:</td>
            <td><input name="captain" type="checkbox" /></td>
          </tr>
          <tr>
            <td>Gender:</td>
            <td><input name="gender" type="radio" value="F" /> F
                    <input name="gender" type="radio" value="M" /> M
            </td>
          </tr>
          <tr>
```

```
                <td>Dorm:</td>
                <td>
                  <select name="dorm" size="1">
                    <option value=""></option>
                    <option value="Apley Court">Apley Court</option>
                    <option value="Canaday">Canaday</option>
                    <option value="Grays">Grays</option>
                    <option value="Greenough">Greenough</option>
                    <option value="Hollis">Hollis</option>
                    <option value="Holworthy">Holworthy</option>
                    <option value="Hurlbut">Hurlbut</option>
                    <option value="Lionel">Lionel</option>
                    <option value="Matthews">Matthews</option>
                    <option value="Mower">Mower</option>
                    <option value="Pennypacker">Pennypacker</option>
                    <option value="Stoughton">Stoughton</option>
                    <option value="Straus">Straus</option>
                    <option value="Thayer">Thayer</option>
                    <option value="Weld">Weld</option>
                    <option value="Wigglesworth">Wigglesworth</option>
                  </select>
                </td>
              </tr>
            </table>
            <br /><br />
            <input type="submit" value="Register!" />
          </form>
        </div>
      </body>
    </html>
```

Note that even though this is a PHP file, there's actually no dynamic content—no actual PHP code. We'll change that with later versions. We saw most of these tags last time, namely `h1`, `br`, `form`, `title`, `head`, `body`, and `html` itself. One new one is `div`, which simply designates a division.

- We saw last time the `action` attribute of `form`, which specifies where the user input is sent. We can also assign the `method` attribute which specifies *how* the data will be sent. For our purposes, `method` can either be `get` or `post`. If `get`, the data will be passed in the URL, as we have seen with Yahoo and Googlel. If `post`, the data will be passed in the headers of the HTTP request.

- Aesthetically, we're using a table to lay out our HTML form. We can reveal this table by adding a `border` attribute with the value 1. The tags `tr` and `td` designate rows and cells of tables, respectively.

- In the select menu, we assign 1 to the `size` attribute so that only one option gets shown at once. As well, we assign the empty string as the first option so that "nothing" is selected by default.
- It's worth noting that form elements can always have a `value` that's different from what they display. So we could've assigned a numeric value for each dorm, for example.
- Ultimately, the form ends with a submit button. If we click it, we get a dummy message saying that we've registered. The URL has changed to the value specified in the `action` attribute of the form, namely `register1.php`. Let's take a look at its source code:[3]

```
<?
    /***************************************************************
     * register1.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.
     * Redirects user to froshims1.php upon error.
     ***************************************************************/

    // validate submission
    if ($_POST["name"] == "" || $_POST["gender"] == "" || $_POST["dorm"] == "")
    {

        $location = "http://cloud.cs50.net/~cs50/lectures/";
        $location .= "8/src/froshims/froshims1.php";

        header("Location: $location");
        exit;
    }

?>

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Frosh IMs</title>
```

[3]Note that my code below differs slightly from the source code posted on the website. I had to break apart the URL so that it would be readable.

```
  </head>
  <body>
    You are registered!  (Well, not really.)
  </body>
</html>
```

For the first time, we've commingled PHP and XHTML. In the same vein, we could've printed out XHTML tags within our command-line stock lookup application so that the stock price would display properly in a browser.

- We are accessing the input passed from the user via the forms using a variable named `$_POST`. This is actually an associative array, a superglobal one, to be specific, which is pre-populated with the key-value pairs corresponding to the user's input. So, for example, because we assigned `gender` to the `name` attribute of the radio button field, we access the radio button input via `$_POST["gender"]`. In this file, we're doing a simple check: if any of the fields (except the `captain` field) is blank, then return the user back to the registration page. We redirect the user to a different URL using the `header()` function.

2.2.1 E-mailing Data

- Of course, we're still not actually registering the user. Let's skip ahead to `register3.php`:

```
<?
    /***********************************************************************
     * register3.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.
     * Reports registration via email.
     * Redirects user to froshims3.php upon error.
     **********************************************************************/

    // validate submission
    if ($_POST["name"] != "" && $_POST["gender"] != "" && $_POST["dorm"] != "")
    {

        $to = "malan@cs50.net";
        $subject = "Registration";
        $body = "This person just registered:\n\n" .
                $_POST["name"] . "\n" .
                $_POST["captain"] . "\n" .
                $_POST["gender"] . "\n" .
                $_POST["dorm"];
        $headers = "From: malan@cs50.net\r\n";
        mail($to, $subject, $body, $headers);
    }
    else
    {
        $location = "http://cloud.cs50.net/~cs50/lectures/";
        $location .= "8/src/froshims/froshims3.php";

        header("Location: $location");
        exit;
    }
?>

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    You are registered!  (Really.)
  </body>
</html>
```

`froshims3.php` is actually no different than `froshims1.php` except that it submits to `register3.php` instead of `register1.php`. We just changed the `action` attribute so that the numbering would match up.

- A reasonable upgrade from paper-based registration would be to send an e-mail to the proctor who was in charge of keeping track of registrants. Again, we're checking that none of the important fields are blank. If they aren't, then we assign a few variables, namely, the recipient (`$to`), the subject of the e-mail (`$subject`), the body of the e-mail (`$body`), and the headers of the e-mail (`$headers`). To assign the body of the e-mail, we make heavy use of the dot or concatenation operator, which attaches each of the quoted strings end-to-end along with newline characters. Finally, we send the e-mail by calling the `mail()` function.
- If any of the important registration fields are blank, we redirect the user. However, if the mail is successfully sent, then we continue on down to output the XHTML at the bottom, which spits out a message saying that registration has succeeded. If we actually click Submit and we check David's CS 50 mail account, we can see that a message has been sent!

- If a user makes a request for a PHP page, the web server not only has to find that PHP file, it also needs to pass it to the PHP interpreter before sending output back to the user. This is actually a simple matter of configuring the web server to do so with any files ending in `.php` (or custom file extensions). Because of this middle man interpreter, serving up PHP content is slower than serving up static HTML content.
- Question: what's to stop us from making up an e-mail address for the registration e-mail to be sent from? Nothing, in fact! It's very easy to spoof an e-mail address in this way. Not recommended, though, as you can be sent to the Ad Board for impersonating another student's address. David actually got in trouble once for chiming in on behalf of someone he disagreed with. He spoofed the guy's e-mail address but forgot to get rid of the automatically appended signature. Oops![4]
- One of the other problems with our implementation is that if the user provides invalid input, he receives no feedback, but only gets bounced back to the registration page. `register2.php`, however, provides some feedback:

```
<?
    /***********************************************************************
     * register2.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.
     * Informs user of any errors.
     **********************************************************************/
?>

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    <? if ($_POST["name"] == "" || $_POST["gender"] == "" ||
              $_POST["dorm"] == ""): ?>
      You must provide your name, gender, and dorm!
       Go <a href="froshims2.php">back</a>.
```

[4]Seriously, though, you will get caught.

```
    <? else: ?>
      You are registered!  (Well, not really.)
    <? endif ?>
  </body>
</html>
```

Here's a new syntactic trick. If we end the if statement with a colon, then whatever comes next in XHTML will be printed only if the statement is satisfied. Otherwise, the XHTML that comes after the else will be displayed. Finally, we end with an explicity `endif`.

- Another of the downsides of our implementation is that if a user inputs invalid data, then he has to click the back button, in which case all of the data he previously inputted has disappeared. We can fix this by implementing a registration form that submits to itself, as we do in `froshims4.php`:

```
<?
    /***********************************************************************
     * froshims4.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.
     * Submits to itself.
     **********************************************************************/

    // if form was actually submitted, check for error
    if ($_POST["action"])
    {
        if ($_POST["name"] == "" || $_POST["gender"] == ""
             || $_POST["dorm"] == "")
            $error = true;
    }
?>


<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Frosh IMs</title>
  </head>
```

```
<body>
  <div align="center">
    <h1>Register for Frosh IMs</h1>
    <? if ($error): ?>
      <div style="color: red;">You must fill out the form!</div>
    <? endif ?>
    <br /><br />
    <form action="froshims4.php" method="post">
      <table border="0" style="text-align: left;">
        <tr>
          <td>Name:</td>
          <td><input name="name" type="text" /></td>
        </tr>
        <tr>
          <td>Captain:</td>
          <td><input name="captain" type="checkbox" /></td>
        </tr>
        <tr>
          <td>Gender:</td>
          <td><input name="gender" type="radio" value="F" /> F
              <input name="gender" type="radio" value="M" /> M
          </td>
        </tr>
        <tr>
          <td>Dorm:</td>
          <td>
            <select name="dorm" size="1">
              <option value=""></option>
              <option value="Apley Court">Apley Court</option>
              <option value="Canaday">Canaday</option>
              <option value="Grays">Grays</option>
              <option value="Greenough">Greenough</option>
              <option value="Hollis">Hollis</option>
              <option value="Holworthy">Holworthy</option>
              <option value="Hurlbut">Hurlbut</option>
              <option value="Lionel">Lionel</option>
              <option value="Matthews">Matthews</option>
              <option value="Mower">Mower</option>
              <option value="Pennypacker">Pennypacker</option>
              <option value="Stoughton">Stoughton</option>
              <option value="Straus">Straus</option>
              <option value="Thayer">Thayer</option>
              <option value="Weld">Weld</option>
              <option value="Wigglesworth">Wigglesworth</option>
            </select>
          </td>
          </tr>
        </table>
        <br /><br />
        <input name="action" type="submit" value="Register!" />
      </form>
    </div>
  </body>
</html>
```

Now we have two cases: the user is visiting for the first time or the user has already submitted data and we need to tell them whether it's valid or not. To distinguish these two cases, we have an if condition at the top of the file which checks whether the `action` index of the `$_POST` superglobal is set or not. If it is, then the user must've gotten here by submitting the form already, so we go ahead and validate the data. If the data is invalid, we set a flag variable named `$error` to true. Even though this variable is declared within curly braces, its scope is global by default in PHP. Later, we check if `$error` is set, in which case we print out an error message.

2.2.2 Storing Data

- Although we've improved on the paper-based system, we'd like to go one step further and actually store the data in a more convenient format than e-mail, perhaps CSV. Even CSV isn't very convenient, however, and we might go one step further and implement an actual SQL database.
- The database engine we'll use in CS 50 is MySQL, mostly because it's free, well-documented, and widely used. However, there are many other options out there, including SQL Server and Oracle.
- Back in the day, you would've only been able to interact with a database using the command line. However, now there's a convenient GUI called phpMyAdmin which allows you to interact with your databases using a website.
- You can think of a SQL database as an Excel spreadsheet, each of whose worksheets represents a table in your database. A database is a little more stringent, however, in that when we create a new table, we have to specify how many fields it will have, the names of those fields, as well as the data types and lengths of those fields.
- Let's start by creating a table with four fields named name, captain, gender, and dorm. For data types, we'll specify `VARCHAR` for name, with a length of 100, `BOOL` for captain, `ENUM` for gender (writing `'M','F'` for Length/Values), and `VARCHAR` for dorm, with a length of 100. When we hit the Go button, the table will be created, but we'll also see the actual SQL statement which was executed. SQL, by the way, stands for *structured query language.*

- Once we've created a table, we can insert values manually using phpMyAdmin. The basic list of SQL statements which can be executed are as follows:
  - SELECT
  - INSERT
  - UPDATE
  - DELETE

  So it looks like if we want to interact with our database via PHP, we'll need to use the INSERT command, as the phpMyAdmin interface did for us, in order to add registrants. Let's do this in `register8.php`:

```
<?
    /***********************************************************************
     * register8.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.
     * Records registration in database.
     * Redirects user to froshims8.php upon error.
     **********************************************************************/

    // validate submission
    if ($_POST["name"] == "" || $_POST["gender"] == ""
        || $_POST["dorm"] == "")
    {
        $location = "http://cloud.cs50.net/~cs50/lectures/";
        $location .= "8/src/froshims/froshims8.php";

        header("Location: $location");
        exit;
    }

    // connect to database
    mysql_connect("localhost", "malan_lecture", "12345");
    mysql_select_db("malan_lecture");

    // scrub inputs
    $name = mysql_real_escape_string($_POST["name"]);
    if ($_POST["captain"])
        $captain = 1;
    else
        $captain = 0;
    $gender = mysql_real_escape_string($_POST["gender"]);
    $dorm = mysql_real_escape_string($_POST["dorm"]);

    // prepare query
    $sql = "INSERT INTO registrants (name, captain, gender, dorm)
            VALUES('$name', $captain, '$gender', '$dorm')";

    // execute query
    mysql_query($sql);

?>

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    You are registered!  (Really.)
  </body>
</html>
```

Using the functions `mysql_connect()` and `mysql_select_db()`, we connect to our database server and select our actual database. php.net is your friend, by the way. Just by Googling a function name, or even what you *think* might be a function name in PHP, you'll usually get the PHP manual as your first result. Possibly the best feature of the manual is the list of examples at the bottom.

- Before we insert data into our database directly from the `$_POST` superglobal, we need to sanitize it. Users can give us all sorts of invalid, ugly, and even malicious data that might disrupt our database. Just ask Bobby Tables' school. To sanitize or escape user input so that it's safe for MySQL, we pass it to the function `mysql_real_escape_string()`.

- Once we've sanitized the user's input, we construct the SQL statement in a variable called `$sql`. Within `$sql`, the actual data to be inserted is between parentheses and enclosed in single quotes (if it's a string).

- If we enter data in the registration form and click Submit, we can browse our database on phpMyAdmin and see that the data was actually inserted in our database. What if we want to fetch that data and display it in the browser? We can use the `SELECT` statement to do so, as in `registrants.php`:

```
<?
    // connect to database
    mysql_connect("localhost", "malan_lecture", "12345");
    mysql_select_db("malan_lecture");

    // prepare query
    $sql = "SELECT * FROM registrants";

    // execute query
    $result = mysql_query($sql);


?>

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    <?
        // iterate over results
        while ($row = mysql_fetch_assoc($result))
        {
            print($row["name"]);
            print("<br />");
        }
    ?>
  </body>
</html>
```

The while loop is iteratively grabbing one row at a time from our SQL query result until there are no rows left, in which case `mysql_fetch_assoc()` will return false.

- All of this syntax will become useful to you as you compete Problem Set 7, for which you will be given a database to store registrants for your stock-trading website. You'll have to implement your own front end for user registration as well as stock lookup and stock buying and selling. The Problem Set 6 Big Board will be replaced with another that ranks competitors by the worth of their (fake) stock portfolios!