

LAB 8

PHP ARRAYS AND SUPERGLOBALS

What You Will Learn

- How to create, traverse, access and insert into PHP arrays
- How to use non numeric keys as indexes in arrays
- How PHP makes HTTP variables easily accessible through superglobals
- How to write a script that responds to user choice.

Approximate Time

The exercises in this lab should take approximately 90 minutes to complete.

Fundamentals of Web Development, 2nd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
<http://www.funwebdev.com>

Date Last Revised: Feb 24, 2017

ARRAYS

PREPARING DIRECTORIES

- 1 If you haven't done so already, create a folder in your personal drive for all the labs for this book. Like Lab 7, this set of labs requires a functioning webserver to interpret the PHP code. If you do not have a webserver running go back to the last Lab exercises and get one set up.
- 2 From the main labs folder (either downloaded from the textbook's web site using the code provided with the textbook or in a common location provided by your instructor), copy the folder titled Lab08-Material to your course folder created in step one.

Arrays are important data structures in programming because they allow programmers to manage sets of objects as a single entity. In PHP, arrays take on additional importance in the form of super global arrays that hold data sent from the client to the server.

STARTING WITH PHP ARRAYS

EXERCISE 8.1 — USE PHP ARRAYS

- 1 Examine lab08-exercise01.php in a text editor and in your browser. Note that the page is going to contain a weather forecast.
- 2 In order to define weather for the next 7 days, we will use arrays to store the expected high, low and type of weather each day.

Although later you will learn about more efficient data structures to store related data, for now we will define three separate arrays: one for the date, one for the high temperatures, one for the low ones. Define these arrays inside of the `<?php ... ?>` tags as follows:

```
$days = array("Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
               "Saturday", "Sunday");
$highs = array(20,30,26,30,30,29,25);
$ lows = array(10,11,15,18,20,13,11);
```

- 3 Now we will define a function that takes 3 parameters: the day of the week, the high value, and the low value. Note that the markup that is being output in this function is the same as that within the container `<div>` below the header.

```
/*
  Outputs the relevant bootstrap markup to display the weather forecast
  for a single day
*/
function outputForecast($day,$high,$low) {
    echo '<div class="panel panel-default col-lg-3 col-md-3 col-sm-6">';
    echo '<div class="panel-heading">';
    echo '<h3 class="panel-title">' . $day . '</h3>';
```

```

        echo '</div>';
        echo '<div class="panel-body">';
        echo '<table class="table table-hover">';
        echo '<tr><td>High:</td><td>' . $high . '</td></tr>';
        echo '<tr><td>Low:</td><td>' . $low . '</td></tr>';
        echo '</table>';
        echo '</div>';
        echo '</div>';
    }

```

- 4 Recall that to access the values in an array you use the square bracket notation where the key is put inside the brackets. Replace the container markup with the following.

```

<div class="container theme-showcase" role="main">
<?php
    outputForecast($days[0], $highs[0], $lows[0]);
?>
</div>

```

- 5 Test in browser.
- 6 Now add in a loop so our page displays the forecast for the entire week.

```

<div class="container theme-showcase" role="main">
<?php
    for ($i=0; $i<count($days); $i++) {
        outputForecast($days[$i], $highs[$i], $lows[$i]);
    }
?>
</div>

```

Notice that this loop calls the new function for each element in the \$days array. The results should look similar to that shown in Figure 8.1

- 7 Array keys in PHP can be integer or strings. Using string keys can often make your array code more intuitive. Modify your arrays as follows.

```

$highs = array("Monday" => 20, "Tuesday" => 30, "Wednesday" => 26, "Thursday"
=> 30, "Friday" => 30, "Saturday" => 29, "Sunday" => 25);
$lows = array("Monday" => 10, "Tuesday" => 11, "Wednesday" => 15, "Thursday"
=> 18, "Friday" => 20, "Saturday" => 13, "Sunday" => 11);

```

- 8 Now modify your loops as follows and then test (the result in the browser should be the same).

```

<div class="container theme-showcase" role="main">
<?php
    /* version 2 */
    foreach ($highs as $key => $todayHigh) {
        outputForecast($key, $todayHigh, $lows[$key]);
    }
?>
</div>

```

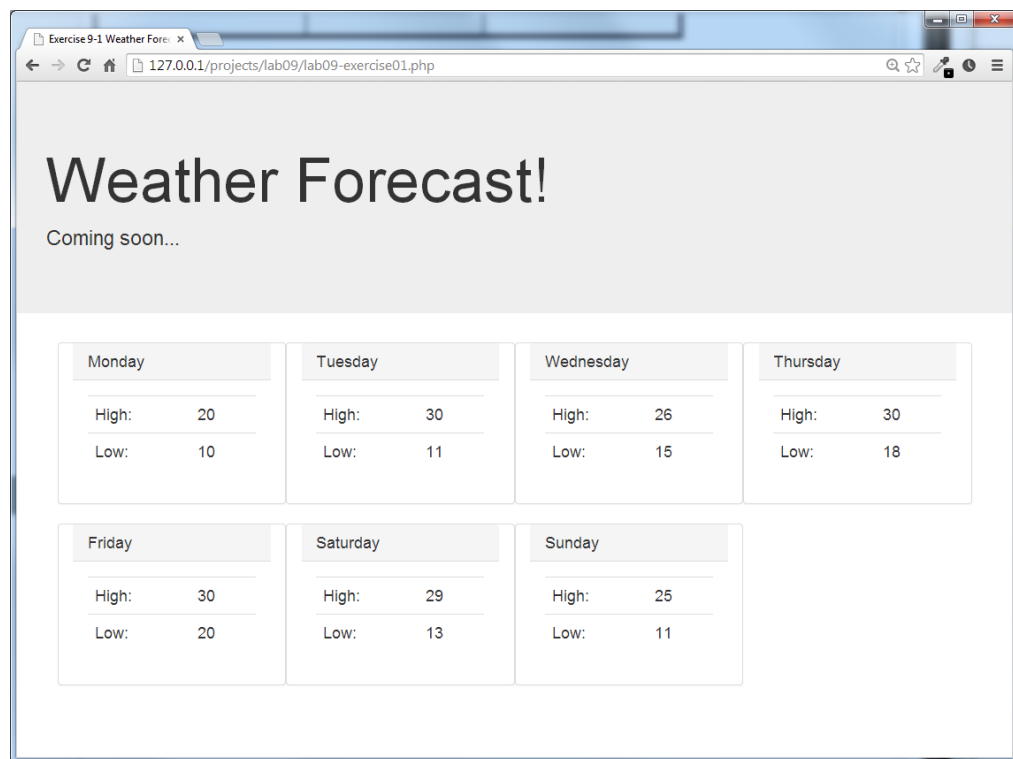


Figure 8.1 – Completed Exercise 1 completed.

EXERCISE 8.2 — ITERATING THROUGH A 2D ARRAY

- 1 To demonstrate the usage of the 2D arrays we will improve and build on Exercise 1 (or you can open [labo8-exercise02.php](#)).

Whereas we had two parallel arrays (one of highs and one of lows) before, we would be better served by a single array where each day of the week has an array of values rather than one. Replace your two weather arrays with the following:

```
$forecast = array("Monday" => array(20,10),
                  "Tuesday" => array(30,11),
                  "Wednesday" => array(26,15),
                  "Thursday" => array(30,18),
                  "Friday" => array(30,20),
                  "Saturday" => array(29,13),
                  "Sunday" => array(25,11));
```

- 2 Since the previous exercise made use of two arrays, we will have to make some changes to make it work with one. Modify your loop as follows.

```
<div class="container theme-showcase" role="main">
<?php
    foreach ($forecast as $key => $weather) {
        outputForecast($key, $weather[0], $weather[1]);
    }
?>
```

```
</div>
```

Notice that in comparison to step 8 in the previous exercise, our loop code now looks a bit cleaner.

- 3 To illustrate why the 2d array is more flexible than the parallel arrays we will now add a third element to our forecast: a string description.

Modify each element in the `$forecast` array declaration to take a third element.

```
$forecast = array("Monday" => array(20,10,"Cloudy"),
    "Tuesday" => array(30,11,"Sunny"),
    "Wednesday" => array(26,15,"Sunny"),
    "Thursday" => array(30,18,"Cloudy"),
    "Friday" => array(30,20,"Rain"),
    "Saturday" => array(29,13,"Rain"),
    "Sunday" => array(25,11,"Cloudy"));
```

- 4 Modify the `outputForecast()` function as follows.

```
function outputForecast($day,$high,$low, $description) {
    echo '<div class="panel panel-default col-lg-3 col-md-3 col-sm-6">';
    echo '<div class="panel-heading">';
    echo '<h3 class="panel-title">' . $day . '</h3>';
    echo '</div>';
    echo '<div class="panel-body">';
    echo '<table class="table table-hover">';
    echo '<tr><td>High:</td><td>' . $high . '</td></tr>';
    echo '<tr><td>Low:</td><td>' . $low . '</td></tr>';
    echo '</table>';
    echo '</div>';
    echo '<div class="panel-footer"> ' . $description . '</div>';
    echo '</div>';
}
```

- 5 Modify the output loop as follows.

```
foreach ($forecast as $key => $weather) {
    outputForecast($key, $weather[0], $weather[1], $weather[2]);
}
```

- 6 Test in browser. The result should look similar to that shown in Figure 8.2.

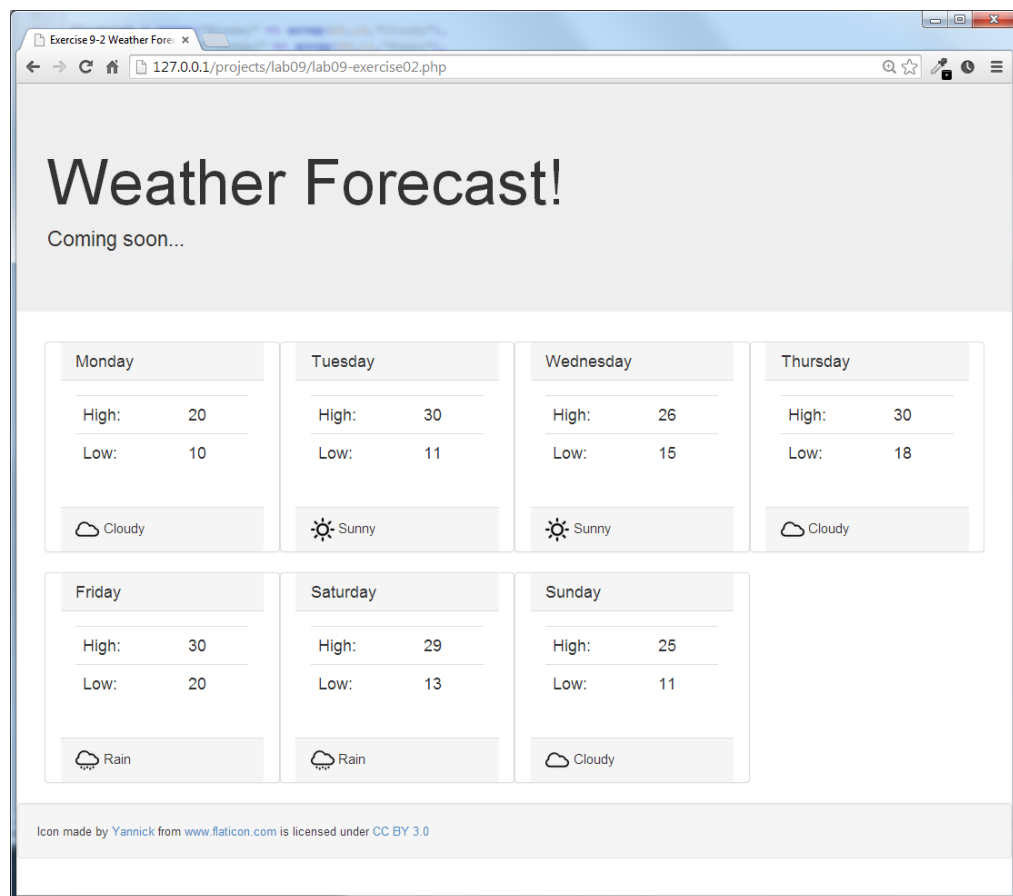


Figure 8.2 Output from the completed Exercise 2

EXERCISE 8.3 — ARRAY SORTING

- 1 Open `lab08-exercise03.php` in a text editor and your browser. Notice it prints some sports leaderboard in no particular order making use of the `print_r()` function.
- 2 The data is stored in an array where the keys are the names of the players and the values are their points this season. You can sort the array using the `sort()` function. Add the following lines:


```
echo "<pre>";
print_r($players);
sort($players);
print_r($players);
echo "</pre>";
```
- 3 Save and test the page. Since this is an associative array, this is not the result we probably wanted since the `sort()` function assigns new keys to the elements in the array. That is, it will remove any existing keys that may have been assigned, rather than

just reordering the keys.

- 3 We obviously need to keep the keys and values associated with each other. Let us try another sort function, namely the `asort()` function

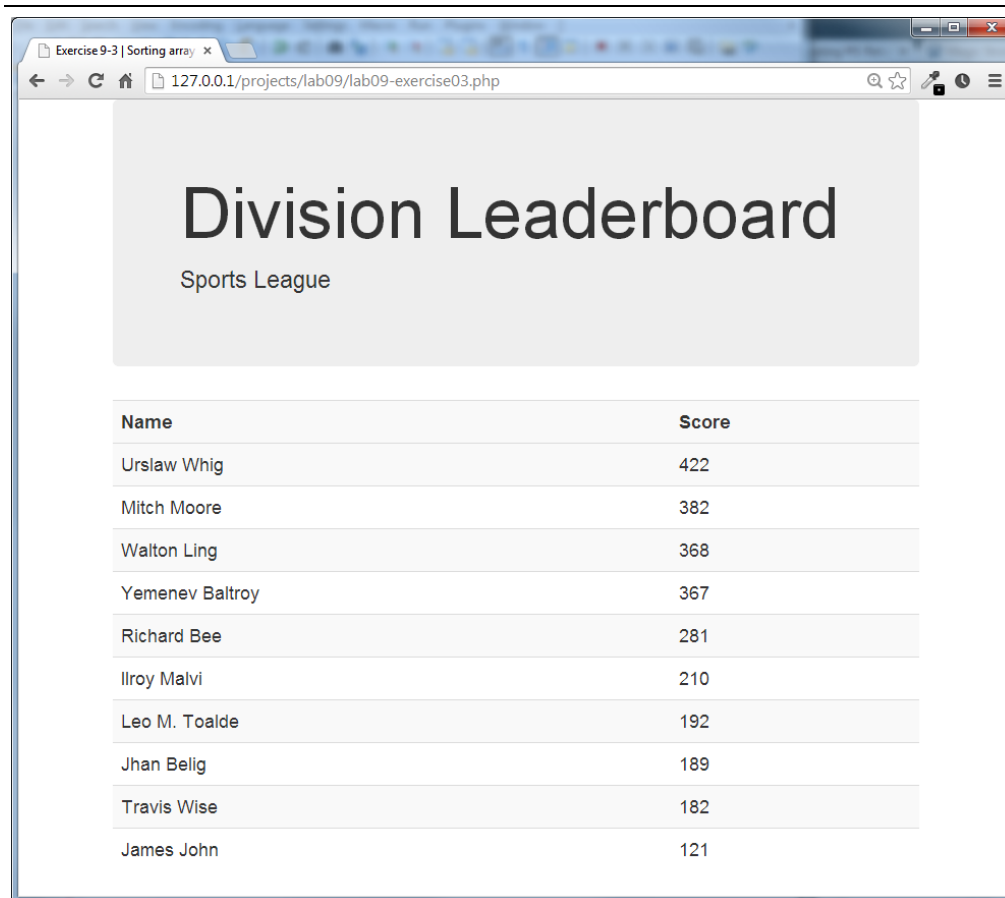
```
asort($players);  
print_r($players);
```

Save this and refresh the page. The array is now sorted from smallest to largest.

- 4 Let us now reverse the sorted array so that the largest score is on top. Add this line right after the sort:

```
asort($players);  
$players =array_reverse($players);
```

- 5 Finally replace the `print_r()` output of the array with a more nicely formatted table such as that shown in Figure 8.3.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1/projects/lab09/lab09-exercise03.php'. The page content includes a title 'Division Leaderboard' and a subtitle 'Sports League'. Below this is a table with two columns: 'Name' and 'Score'. The table contains 12 rows of player names and their corresponding scores, sorted in descending order of score.

Name	Score
Urslaw Whig	422
Mitch Moore	382
Walton Ling	368
Yemenev Baltroy	367
Richard Bee	281
Ilroy Malvi	210
Leo M. Toalde	192
Jhan Bellig	189
Travis Wise	182
James John	121

Figure 8.3 The sorted array from Exercise 3 formatted in a table for output.

\$_GET AND \$_POST

EXERCISE 8.4 — CHECKING FOR POST

- 1 Test lab08-exercise04.php in a browser and notice that it contains a simple one field form that posts the results back to itself.
- 2 The first thing to do is detect whether the page was reached through a **get** (a link followed) or whether the page was arrived by the form being **posted**.

Edit the displayPostStatus() function as follows and test.

```
function displayPostStatus() {
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        echo "Post detected";
    }
    else{
        echo "No Post Detected";
    }
}
```

Now when you post the form you see the post detected, and when you enter the URL and hit enter (GET) it does not see the POST.

- 3 Next, let us use the values being transmitted to the server.

Inside the code for when a POST was detected, we can print out the full contents of the \$_POST array to see what variables we have using print_r.

```
echo "<pre>";
print_r($_POST);
echo "</pre>";
```

You will see that the only field being posted (and can be seen in the HTML form as well) is alias. Comment these lines out for now and bring them back to help with debugging if needed.

We will now say "Hello user", where user is the value submitted through the input field alias. Modify our earlier if statement with the following and test.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    echo "Hello ". $_POST['alias'];
}
```

- 4 This works, and will say hello to whatever you type into the box. Now let's do a little checking to ensure it's not blank. Place the single echo name inside and modify our if statement to check for blank.

```
if (isset($_POST['alias']) && $_POST['alias']!="") {
    echo "Hello " . $_POST['alias'];
}
else{
    echo "Post detected, but no value";
}
```


- 5 Test in browser. Now the page says hello if you type a name and other detects the post but is able to handle the post accordingly if the name is blank.

Note: The page is vulnerable to attacks such as XSS (see Chapter 18). Before using these techniques in production, remember that we should never trust user input.

EXERCISE 8.5 — USING QUERY STRING VALUES

- 1 Open, examine, and test lab08-exercise05.php. It contains an array of data and a basic page with navigation.
- 2 Modify the code that outputs the menu by changing the URL of the links to use query strings. Whereas currently all the links are all to # change them to:

```
foreach ($content as $key => $elements) {
    echo "<li><a href='?page=$pageCount'>$key</a></li>";
    $pageCount++;
}
```

Now when you click on the links the URL should look like

lab08-exercise05.php?page=1

...

lab08-exercise05.php?page=4

- 3 In order to make the menu correctly highlight the current item we must add a CSS style to the for the current page. But how do we know which page we are on, when all the pages are being served by the same script?

Since the querystring in the URL contains page=XXX we can check the \$_GET superglobal array for the current page.

In Bootstrap we only need to add class="active" to the if the page is the current page. Using the condensed conditional notation we can change our one echo line in the loop to output the class attribute if the pageCount equals the value in the \$GET array:

```
foreach ($content as $key => $elements) {
    echo "<li ";
    echo ($pageCount==$_GET['page'] ? " class='active'" : "");
    echo "><a href='?page=$pageCount'>$key</a></li>";
    $pageCount++;
}
```

- 4 If you found the above example a little too concise, you can replace it with the following.

```
foreach ($content as $key => $elements) {
    echo "<li ";
    if ($pageCount==$_GET['page'])
        echo " class='active'";
    else
        echo "";
    echo "><a href='?page=$pageCount'>$key</a></li>";
    $pageCount++;
}
```

- 5 Now let us make use of the data array to output the title and content in the page.

In the head section of the page modify the <title> tag so that we access the \$_GET array, and use the page value as an index into our \$content array. To do this, add the following line.

```
$content = array("Home" => array("Home Page","This page is the home page"),
    "Page 2" => array("The second page", "second page content here"),
    "Page 3" => array("The third page", "third page content here"),
    "Page 4" => array("The fourth page", "fourth page content here"));

$justContent = array_values($content);

if (!isset($_GET["page"])){
    $_GET["page"] = 1;
}
```

The if statement will make sure that the first time you load the page \$_GET["page"] has the value of "Home"

- 6 Modify the <title> element as follows.

```
<title><?php echo $justContent[$_GET['page']-1][0]; ?></title>
```

- 7 Finally, in a similar way to the title, we will output the final field of content as content in the page. Your final page should look similar to that in Figure 8.4.



Figure 8.4 – A screen shot of one of the four pages, with the menu highlighted

\$_SERVER ARRAY

EXERCISE 8.6 — USING THE \$_SERVER SUPERGLOBAL

- 1 Examine lab08-exercise06.php and then test in a browser and text editor. In the

browser you will see that all the values in the `$_SERVER` array are being displayed. Some of these values are sent by the client (you at your computer) and other headers are responses from the server.

- 2 Remove the `print_r()` invocation, and replace it with two sections that highlight some interesting variables on both the client and server side.

```
echo "<h1>Server Side Values</h1>";
echo "SERVER_NAME: " . $_SERVER["SERVER_NAME"] . "<br/>";
echo "SERVER_SOFTWARE: " . $_SERVER["SERVER_SOFTWARE"] . "<br/>";
echo "SERVER_ADDR: " . $_SERVER["SERVER_ADDR"] . "<br/>";

echo "<h1>Headers from the client</h1>";
echo "REMOTE_ADDR: " . $_SERVER["REMOTE_ADDR"] . "<br/>";
echo "HTTP_USER_AGENT: " . $_SERVER["HTTP_USER_AGENT"] . "<br/>";
echo "HTTP_REFERER: " . $_SERVER["HTTP_REFERER"] . "<br/>";
```

- 3 Now let's use a built-in function to get more easy-to-use information out of the user-agent headers

```
$browser = get_browser($_SERVER['HTTP_USER_AGENT'], true);
print_r($browser);
```

Note: In order for `get_browser()` to work, your `php.ini` file must point to the `browscap` setting to the correct location of the `browscap.ini` file on your system. A current `browscap.ini` file can be downloaded from php.net.

\$_FILES ARRAY

EXERCISE 8.7 — PROCESSING FILE UPLOADS

- 1 Open `lab08-exercise07.php` in a browser. You will notice that it is a form for uploading files. You can try to upload a file now and the form will post, but for now nothing will happen with the file.
- 2 In order to properly process the uploaded file we must take the posted file and store it permanently on the server. We must also update the user interface so the user knows the upload was successful.

We will begin by determining whether to display the form, or instead to process the uploaded file and output a message by changing the PHP as follows:

```
// if the form was posted, process the upload
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    echo "File was sent to server <a href=''>Upload another file</a>";
}
else {
    echo getFileUploadForm();
}
```

- 3 Now we will actually copy the uploaded file to a location we control.

Create a folder in your working directory named **UPLOADS** and grant the apache user permissions to upload (if you are working locally using something like easyPHP or XAMP this will already be the case). Now add code to the if statement which moves the file:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $fileToMove = $_FILES['file1']['tmp_name'];
    $destination = "./UPLOADS/" . $_FILES["file1"]["name"];
    if (move_uploaded_file($fileToMove,$destination)) {
        echo "File was sent to server <a href=''>Upload another file</a>";
    }
    else {
        echo "there was a problem moving the file";
    }
}
```

Try uploading the file and confirm that the file was placed in the UPLOADS directory.

EXERCISE 8.8 — MANAGING UPLOADED FILES

- 1 The last exercise allowed us to upload a single file to an upload directory. This exercise builds on that one and adds multiple file uploads and improved file management features.
- 2 To make our code a little cleaner, create a function at the top of your page as follows.

```
/*
    Moves an uploaded file to our destination location
*/
function moveFile($fileToMove, $destination) {
    echo $destination . ' Uploaded successfully<br/>';
    move_uploaded_file($fileToMove, "UPLOADS/" . $destination)
        or die("error");
}
```

- 3 Now, you will enable multiple file uploads. In the HTML field for the file input you will need to add the multiple attribute to the upload form as follows (that is, you will need to make the change to the markup generated by the getFileUploadForm() function).

```
<input type="file" name="file1" id="file1" multiple/>
```

Now if you refresh the page and click browse you will be able to ctrl or shift click for pick multiple files.

Another change is that the name of the attribute should include square brackets so that it can be processed as an array in PHP. To do so, make the following change.

```
<input type="file" name="file1[]" id="file1" multiple/>
```

- 4 Currently when you try to upload two files only one will be uploaded successfully. To handle multiple files you must change your upload processing code to handle arrays of values (i.e., multiple uploaded files).

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
```

```

// check if user uploaded multiple files
if (is_array($_FILES["file1"]["error"])) {
    $count=0;
    // loop through each uploaded file
    foreach ($_FILES["file1"]["error"] as $error) {
        if ($error == UPLOAD_ERR_OK) {
            $clientName = $_FILES["file1"]["name"][$count];
            $serverName = $_FILES["file1"]["tmp_name"][$count];
            moveFile($serverName, $clientName);
            $count++;
        }
    }
}
else {
    // user only uploaded a single file
    if ($_FILES["file1"]["error"] == UPLOAD_ERR_OK) {
        $clientName = $_FILES["file1"]["name"];
        $serverName = $_FILES["file1"]["tmp_name"];
        moveFile($serverName, $clientName);
    }
}
}
else {
    echo getFileUploadForm();
}

```

- 5 Test in browser and verify that the upload and move worked as expected.
- 6 Finally, to illustrate some of the validation we can do on the server side, add a restriction to only allow the upload of jpg and png files. Define some valid extensions and MIME types:

```

function moveFile($fileToMove, $destination) {
    $validExt = array("jpg", "png");
    $validMime = array("image/jpeg", "image/png");
}

```

- 7 Now in your moveFile() function, you will need to add code to check the extension and mime type against our list of valid types. You will also have to pass in the file type of the file being uploaded.

```

function moveFile($fileToMove, $destination, $fileType) {
    $validExt = array("jpg", "png");
    $validMime = array("image/jpeg", "image/png");

    // make an array of two elements, first=filename before extension,
    // and the second=extension
    $components = explode(".", $destination);
    // retrieve just the end component (i.e., the extension)
    $extension = end($components);
    // check to see if file type is a valid one
    if (in_array($fileType, $validMime) && in_array($extension, $validExt)) {
        echo $destination . 'Uploaded successfully<br/>';
        move_uploaded_file($fileToMove, "UPLOADS/" . $destination) or
        die("error");
    }
    else
        echo 'Invalid file type=' . $fileType . ' Extension=' . $extension .
        '<br/>';
}

```

```
}
```

- 8 Change the invocation of the function by passing the file type.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // check if user uploaded multiple files
    if (is_array($_FILES["file1"]["error"])) {
        $count=0;
        // loop through each uploaded file
        foreach ($_FILES["file1"]["error"] as $error) {
            if ($error == UPLOAD_ERR_OK) {
                $clientName = $_FILES["file1"]["name"][$count];
                $serverName = $_FILES["file1"]["tmp_name"][$count];
                $fileType = $_FILES["file1"]["type"][$count];
                moveFile($serverName, $clientName, $fileType);
                $count++;
            }
        }
    }
    else {
        // user only uploaded a single file
        if ($_FILES["file1"]["error"] == UPLOAD_ERR_OK) {
            $clientName = $_FILES["file1"]["name"];
            $serverName = $_FILES["file1"]["tmp_name"];
            $fileType = $_FILES["file1"]["type"];
            moveFile($serverName, $clientName, $fileType);
        }
    }
}
```

- 9 Test in browser. Try uploading both valid (JPG or PNG) and non-valid files.

READING/WRITING FILES

EXERCISE 8.9 — PHP FILE ACCESS

- 1 Finally, we will demonstrate how easily you can read and write files in PHP. Examine the text file [datafile.txt](#) in an editor.

- 2 Open lab08-exercise09.php in your favorite editor.

- 3 This file reads in data from a file and echoes it out using an echo statement.

Remove the echo statement and write the following code which will convert the string from the file into an array. We are using the `explode()` function, which takes advantage of the fact that this file is newline delimited. We can output the array using `print_r()`.

```
$daysOfWeek = explode("\n",$fileContents);
```

- 4 We can now loop through the elements of the array, and each element in the array corresponds to one line of the file! Note that each line of the file is a bunch of numbers separated by commas. The values are:

Day, month, year, high temperature, low temperature.

We can therefore iterate through our array, and then split out each string into another array as follows:

```
foreach($daysOfWeek as $day) {
    $elements = explode(",",$day);
    echo $elements[1]; // example
    //...
```

- 5 Using the `outputForecast()` function from EXERCISE 8.2, convert the data in the file into output that resembles the 14 day forecast as shown in Figure 8.5.

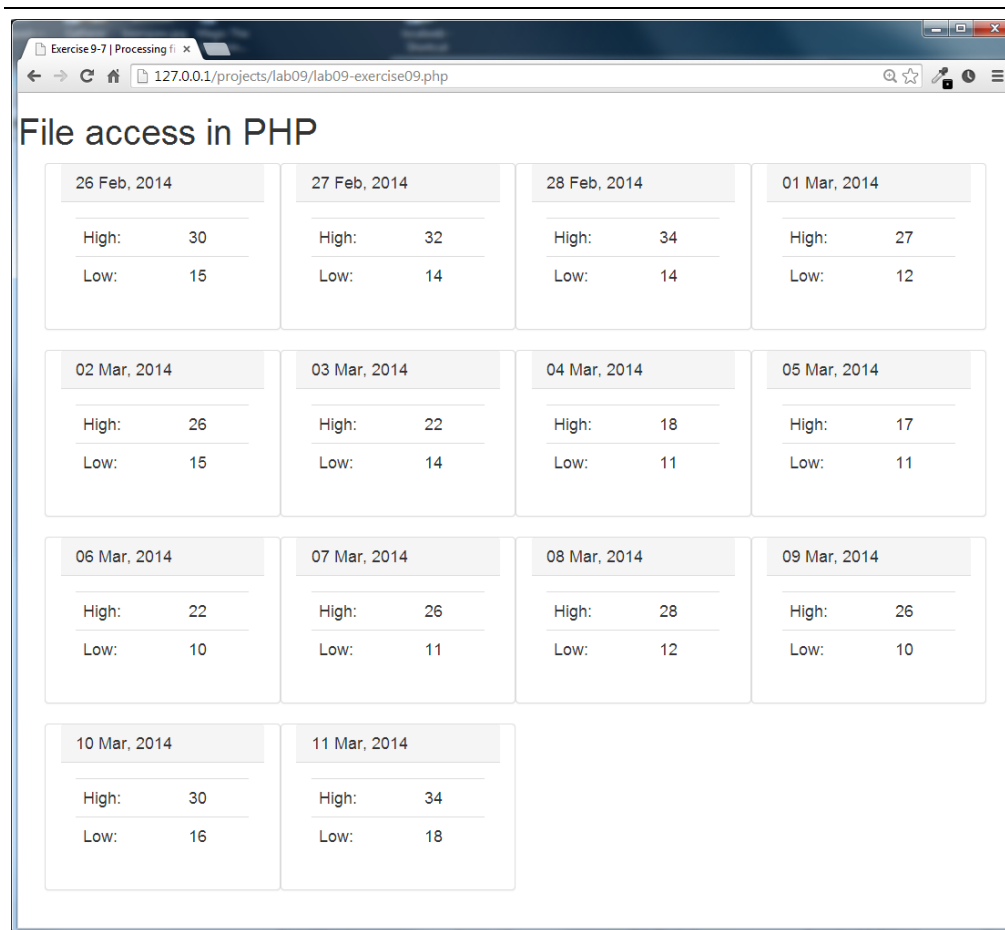


Figure 8.5 Illustrating the output of the completed EXERCISE 12