

# A Gentler Introduction to *PhP* and Its Application in Database Programming

Zhizhang Shen \*

Dept. of Computer Science and Technology  
Plymouth State University

## Abstract

This is Part I of the lab notes prepared for the students of *CS3600 Introduction to the Database Systems* for Fall 2009, with many examples taken from [1].

We discuss some of the basic components of *PhP5* as a programming language, and then turn to its usage in database programming, via its relationship with the *MySQL* DBMS and the *HTML* language.

## Contents

<b>1</b>	<b>An introduction</b>	<b>2</b>
<b>2</b>	<b>A simple example</b>	<b>3</b>
2.1	How to run a <i>PhP</i> program? . . . . .	6
<b>3</b>	<b><i>PhP</i> basics</b>	<b>7</b>
3.1	Everybody starts here . . . . .	7
3.2	The usual arithmetic operations . . . . .	7
3.3	The conditional structure . . . . .	8
3.4	The loop structure . . . . .	10
3.5	Associative arrays . . . . .	14
3.6	Strings . . . . .	21
3.7	Functions . . . . .	27
3.8	The scope of variables . . . . .	28
3.9	Recursion . . . . .	31

---

\* *Address correspondence to* Dr. Zhizhang Shen, Dept. of Computer Science and Technology, Plymouth State University, Plymouth, NH 03264, USA. *E-mail address:* [zshen@plymouth.edu](mailto:zshen@plymouth.edu).

<b>4</b>	<b>Data passing in <i>PhP</i></b>	<b>32</b>
4.1	The GET method . . . . .	32
4.2	The Post method . . . . .	35
4.3	An example . . . . .	36
4.4	Another example: a quiz . . . . .	39
4.5	More data structures . . . . .	42
<b>5</b>	<b>Work together with <i>MySQL</i></b>	<b>43</b>
5.1	A basic procedure . . . . .	43
5.2	An example: a sign-up service . . . . .	43
5.3	Find out the best and the worst . . . . .	46
5.4	Another example: on countries and cities . . . . .	50
5.5	Work with multiple tables . . . . .	54
5.6	A comprehensive example: rate your boss . . . . .	57
5.7	More data types . . . . .	62
<b>6</b>	<b>Debugging</b>	<b>64</b>
<b>7</b>	<b>Appendix: An introduction to HTML</b>	<b>65</b>
7.1	The basics . . . . .	65
7.2	Add in a picture . . . . .	66
7.3	Make it look better . . . . .	67
7.4	The list structures . . . . .	68
7.5	Add in a table . . . . .	68
7.6	Hyperlinks . . . . .	69
7.7	Sky is the limit . . . . .	70

# 1 An introduction

*MySQL*, as a (partial) implementation of the SQL specification, is a very good language for defining the structure of the database, and *generating ad hoc queries*. We will learn something about how to define and populate database tables, as well as how to write SQL queries, using *MySQL* in Part (II) of the lab notes, *A Gentler Introduction to MySQL Database Programming*.

However, to build meaningful database applications, particularly, transaction processing applications, the power of a full-fledged high-level programming language, such as Java, C++, or *PhP*, is needed. Moreover, in today's WEB age, lots of database programming are done over the Internet, using WEB as a visual media.

In this part of the lab, we discuss *PhP*, a programming language, and its relationship with *MySQL*, a "free" DBMS, and their connection with *HTML*. The three products are often used together because of the following reasons:

1. There are tons of DBMS, but only a few “good” ones. With its syntax quite similar to Oracle, *MySQL* has been getting better and better, serving all of our practical purposes. Moreover, Oracle costs about \$15,000, while *MySQL* costs between 0 to \$250 depending on what you use it for.
2. Considering the WEB age, it is natural to use WEB as the user interface, which can be written with *HTML*, or *XML*.
3. *PhP*, a language quite similar to Java, plays the part of a go-between, i.e., sends the queries from users to the *MySQL* database, and receives result back from it; together with the control structures that are needed for serious programming.

To summarize, we use *HTML* [6] to write up a WEB friendly interface, embedded with database programming code written in *PhP* [4], which sends queries to, and receives the result from, (a) *MySQL* [3] based database(s).

For more details of database programming for such an environment, see [1], referred to as the *Bible* henceforth; or other stuff such as [5].

**Note:**

- We would assume a familiarity with *HTML*, although we include with this lab notes a brief introduction to *HTML* in an Appendix. To further augment your understanding of *HTML*, you can simply do a google, and find tons of resources, including the one noted at the end of the aforementioned appendix.
- We will also assume that both *MySQL* and *PhP* have been installed in the machine, just like in our case. If you want to know how to install them in your own machine, please consult the *Bible*.
- *PhP* added database programming follows the client/server model in the sense that once a `.php` file with combination of `php` and *HTML* code is entered in a work station, e.g., in Memorial 312, it will be sent to the database server, turing in this case, to be executed. The result is then expressed in pure *HTML* code, and then sent back to your machine to be displayed.

## 2 A simple example

We start by showing a simple database programming example which combines *HTML* presentation, and a *PhP* script.

Assume that we have collected some information about students’ grade and would like to find out who has got the highest and lowest GPA. Below shows an *HTML* document, `sendSpecificQuery.html`, that provides a user interface.

```
<Html>
<!--This is to send a query, which will be picked up and processed by
```

```

    another script.php  -->
<Head>
  <Style Type="text/css">
    <!--
      Body, P, TD {color: black; font-family: verdana; font-size: 10 pt}
      H1 {color: black; font-family: arial; font-size:12 pt}
    -->
  </Style>
</Head>

<!--A table with only one row, consisting of two cells, the first being the -->
<!--left edge, 1/6; and the other contains the form, 5/6 -->
<Table Border=0 cellPadding=10 Width=100%>
  <!--Now define the row-->
  <Tr>
    <!--The following cell shows the left cushion edge-->
    <Td BgColor="FOF8FF" Align=Center VAlign=top Width=17%> </Td>

    <!--The following gives the right entry form part, completely white-->
    <Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>

    <H1>This html script shows how to generate results for simple queries.</H1>
    <p>The following button finds the student(s) with the highest GPA.</p>

<p>The associated SQL code is the following:

  <em> Select S.Name, S.Id From Student S Where S.GPA >= (Select Max(S1.GPA)
    From Student S1)</em></p>
  <!--The following says that this section is going to be handled -->
  <!-- by a separate php script.-->

  <Form Method="post" Action="highestGPA.php">
    <BR><BR>
    <!--The submit button, with the name being $_POST['submit'], -->
    <!--and the value shown being 'Highest GPA'-->
    <Input Type="submit" Name="submit" Value="Highest GPA">
  </Form>

<hr><hr>
<p>The following button finds the student(s) with the lowest GPA.</p>

```

<p>The associated SQL code is the following:

```
<em> Select S.Name, S.Id From Student S Where S.GPA <= (Select Min(S1.GPA)
      From Student S1</em></p>
```

```
<!--The following says that this section is going to be handled by another -->
<!--php script.-->
```

```
<Form Method="post" Action="lowestGPA.php">
  <BR><BR>
  <!--The submit button, with the name being $_POST['submit'], -->
  <!--and the value shown being 'Lowest GPA'-->
  <Input Type="submit" Name="submit" Value="Lowest GPA">
</Form>
```

```
</Td>
<!--end of the row definition>
```

```
</Tr>
</Table>
```

```
</Body>
</Html>
```

Now we show the *PhP* script, `highestGPA.php`, to find out the student(s) with the highest GPA. The other one is almost the same. This script also includes a function that is used to display the content of a table.

```
<?php
  //This segment gets in a function used to print out the content of
  //a single table
  include("displayQueryResult.inc");
?>
```

```
<Html>
  <Head>
    <Title>Student(s) with highest GPA</Title>
  </Head>
```

```
<Body>
  <!--The following setting ensures the interface consistency -->
  <Table Border=0 cellPadding=10 Width=100%>
    <!--Now define the row-->
    <Tr>
      <!--The following cell gives the left cushion edge-->
```

```

<Td BgColor="F0F8FF" Align=Center VAlign=top Width=17%> </Td>

<!--The following gives the right entry form part, completely white-->
<Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>
  <!--Now the content of this cell-->
  <?php
    //This is the query to get the highest GPA student(s)
    $query_string="Select S.Name, S.Id, S.GPA From Student S
                  Where S.GPA >= (Select Max(S1.GPA)
                  From Student S1)";

    print("The following displays students with the highest GPA.<BR><BR>");

    //Call the predefined function to print out the cellar table, together
    //with column titles and an appropriate border
    display_db_query($query_string, $global_dbh, TRUE, "Border=2");
    mysql_close($global_dbh);
  ?>
</Td>
<!--end of this cell, thus the whole row-->
</Tr>
</Table>
</Body>
</Html>

```

Notice that the file `displayQueryResult.inc` provides the definition of the `display_db_query` function and can be found at the end of §5 in the *MySQL* lab notes.

## 2.1 How to run a *PhP* program?

To run this program in a *php enabled* server such as `turing`, assume that you have set up the related database, and assume your user name is `j_doe`, you just place the triplet in a folder, e.g., `PhPFiles`, under the `Home` folder of `turing`, and then *Open* the following in IE: `http://turing.plymouth.edu/~j_doe/PhPFiles/sendSpecificQuery.html`.

On the other hand, if you have set up everything in your computer, there should be a folder `htdocs` underneath, e.g., `Apache`, where you should keep all the `.php` files. When running such a file, type in the following in the address line of IE: `http://localhost/sendSpecificQuery.html`.

**Labwork 2.1:** Read through the codes and the comments, figure out what each and every unit does as much as you could. To start with, notice the two different syntax we used to make comments within the respective environments of *HTML* and *PhP*.

## 3 *PhP* basics

We first learn something about the *PhP* language itself, without touching anything dealing with databases.

### 3.1 Everybody starts here

Let's have a look at a simple *PhP* program, which we call `World.php`.

```
<HTML>
<HEAD>
  <TITLE>My first php file</TITLE>
</HEAD>

<BODY>
  <?php
    print("Hello from the PhP world!<BR><BR>\n");
    phpinfo();
  ?>
</BODY>
</HTML>
```

**Question:** What does it do?

**Answer:** You figure it out.

**Question:** How to run this, and any, *PhP* program?

**Answer:** `http://turing.plymouth.edu/~j_doe/PhPFiles/World.php`.

**Labwork 3.1:** Make a directory in `turing`, under `Home`, type and test out the above code, then write something of your own, also test it out. More specifically, you have to complete the following in this lab:

- Comment out, but not delete, the line `phpinfo()`;
- The revised *PhP* script should print out your name, as well as the following line  
`My first PhP script now works.`
- Send me the URL of your script via an email.

### 3.2 The usual arithmetic operations

Let's learn something about how to deal with this sorts of stuff in *PhP* with an example. Below is `testNumber.php`.

```

<HTML>
  <HEAD>
    <TITLE> Test out the Case Sensitiveness </TITLE>
  </HEAD>

  <BODY>
    <p>This segment is to see how php evaluates an arithmetic expression,
      its concept of precedence.
    </p>
    <BR>
    <BR>

    <?php
      $result1=2*2+3*3-10;
      $result2=2*(2+3*3)-10;
      print("2*2+3*3-10=$result1, while");
      print(" 2*(2+3*3)-10=$result2<BR><BR>" );
    ?>

    <p>We will also see how to declare a constant.</p>

    <?php
      //We now define a constant
      define(MY_ANSWER, 42);
      print("The value of a constant MY_ANSWER is <BR>");
      print(MY_ANSWER);
    ?>
  </BODY>
</HTML>

```

Note that ' ' will print stuff out *as is*, but " " will do some evaluation.

**Labwork 3.2:** Test out the above code, write something of your own, test it out; then send me the URL of this script.

### 3.3 The conditional structure

Below does something conditional.

```

<HTML>
  <HEAD>
    <TITLE>If--then-else </TITLE>
  </HEAD>

```

```

<BODY>
  <p>PHP does the conditional structure and has
    its comment mechanism.</p>
  <BR>
<?php
  //The following shows how the if structure
  # We now test how the single line comment
  # works, and
  /* multiple line comment work. */
  if (3==2+1){
    print("<BR><BR>Good - I haven't lost<BR>");
    print("my mind yet. <BR>");
  }
?>
</BODY>
</HTML>

```

Below shows `testBranch.php`, another program involved with the conditional structure.

```

<HTML>
<HEAD>
  <TITLE>Test out logical operators</TITLE>
</HEAD>

<BODY>
  <?php
    $first=2; $second=2.3;

    if($first-$second!=0){
      if($first>$second)
        $difference=$first-$second;
      else $difference=$second-$first;
      print("The difference is $difference<BR>.");
    }
    else print("There is no difference");
  ?>
</BODY>
</HTML>

```

### Labwork 3.3:

1. Test out all the codes you have seen so far.

2. Write a program that prints out the maximum out of three numbers stored in three variables.
3. Think out some application using the conditional structure, write out a program and run it.
4. Send me the URL of all the scripts.

### 3.4 The loop structure

Let's begin with an easy one, `testWhile.php`.

```
<HTML>
<HEAD>
  <TITLE>Test out the while structure</TITLE>
</HEAD>

<BODY>
  <?php
    //Does this look similar?
    $count=1;

    while($count<=10){
      print("count is $count<BR>");
      $count=$count+1;
    }
  ?>
</BODY>
</HTML>
```

Here is another example, `testWhile2.php`, which is to look for the root of a given value.

```
<HTML><HEAD><TITLE>
  Test the for an unbounded while loop</TITLE>
</HEAD><BODY>
  <p>Estimating a root value.</p>
  <H3>Approximating a square root</H3>
  <?php
    //The given value
    $target = 81;
    //Initial guess
    $guess = 1.0;
    //How good it should be
    $precision = 0.0000001;
```

```

//initial estimation
$guess_squared = $guess * $guess;
//an abs function should be handy
while(($guess_squared-$target>$precision)
      or ($guess_squared-$target< -$precision)){
    print("Current guess: $guess is the square
          root of $target<BR>");

    //another guess with a certain formula
    $guess = ($guess + ($target / $guess))/2;
    $guess_squared = $guess * $guess;
}
print("$guess squared = $guess_squared<BR>");
?>
</BODY></HTML>

```

We also can use the *for* loop if we know for sure how many times the loop body is going to run. Below is the `testFor.php`, which sends out a table of stuff.

```

<HTML>
  <HEAD>
    <TITLE>Test the for loop</TITLE>
  </HEAD>

  <BODY>

  <p>Below is the code for constructing a division table. </p>

  <?php
    $start_num = 1;
    $end_num = 10;
  ?>

  <H2>A division table</H2>
  <TABLE border=1 bordercolor=#0099FF>
  <?php

    //<TR> starts the first row
    print("<TR>");

    //The first empty data for that row
    print("<TH> </TH>");

```

```

//The header item for the first row, 1, 2, ..., 10.
//Notice that the <TH> tag is the same as <TD>, except that they indicate that
//it is for the header, thus in bold face.
for ($count_1 = $start_num;
    $count_1 <= $end_num;
    $count_1++)
    print("<TH>$count_1</TH>");

//end of the first row
print("</TR>");

//Now the content of the table, a two level loop
for ($count_1 = $start_num;
    $count_1 <= $end_num;
    $count_1++)
{
    //Start another row, with a specific value ranging from 1 to 10, placed
    //in the first column of this row.
    print("<TR><TH>$count_1</TH>");

    //start yet another column
    for ($count_2 = $start_num;
        $count_2 <= $end_num;
        $count_2++)
    {
        $result = $count_1 / $count_2;

        //Now the content for that cell, notice .3f means that exactly three
        //digits after the decimal point.
        printf("<TD>%.3f</TD>",
            $result); // see Chapter 108
    }
    //end of this row, and going back to the beginning of the second level loop
    print("</TR>\n");
}
?>
</TABLE>
</BODY>
</HTML>

```

If every thing goes right, you should see something like the following as the output of the above script.

Below is the code for constructing a division table.

A division table

	1	2	3	4	5	6	7	8	9	10
1	1.000	0.500	0.333	0.250	0.200	0.167	0.143	0.125	0.111	0.100
2	2.000	1.000	0.667	0.500	0.400	0.333	0.286	0.250	0.222	0.200
3	3.000	1.500	1.000	0.750	0.600	0.500	0.429	0.375	0.333	0.300
4	4.000	2.000	1.333	1.000	0.800	0.667	0.571	0.500	0.444	0.400
5	5.000	2.500	1.667	1.250	1.000	0.833	0.714	0.625	0.556	0.500
6	6.000	3.000	2.000	1.500	1.200	1.000	0.857	0.750	0.667	0.600
7	7.000	3.500	2.333	1.750	1.400	1.167	1.000	0.875	0.778	0.700
8	8.000	4.000	2.667	2.000	1.600	1.333	1.143	1.000	0.889	0.800
9	9.000	4.500	3.000	2.250	1.800	1.500	1.286	1.125	1.000	0.900
10	10.000	5.000	3.333	2.500	2.000	1.667	1.429	1.250	1.111	1.000

We often want to get out a loop when something happens. Just like in Java, we can use `break` to break out a loop. Below is an example, `testBreak.php`, which prints out all the prime numbers less than 500.

```
<HTML>
  <HEAD>
    <TITLE>Test the break statement</TITLE>
  </HEAD>
  <BODY>
    <p>Below shows primes less than 500. </p>
<?php

//end of it
$limit=500;

//the first candidate
$to_test=2;
while(TRUE){
  //the very first number that might divide $to_test, thus making
  // the latter a non-primer
  $testdiv=2;
  if($to_test>$limit)
    break;

  //We still have to test $to_test
  while(TRUE){

    //if the following test is true, there is no way $testdiv
```

```

//divides $to_test thus $to_test is a prime
if($testdiv>sqrt($to_test)){
    print "$to_test ";
    break;
}
//test if $to_test is divisible by $testdiv
if($to_test % $testdiv==0)
    break;
    $testdiv++;
}
$to_test++;
}

?>
</BODY>
</HTML>

```

### Labwork 3.4:

1. Test out all the above codes as contained in this subsection.
2. Write a script that prints out the squares of 1 through 100.
3. Write a script that generates the infamous multiplication table for  $a \times b$ , where  $1 \leq a \leq 9, 1 \leq b \leq 9$ .

Your print-out should also take care of the communitive law, i.e.,  $a \times b = b \times a$ . Hence, the table should hold a blank for any pair  $(a, b)$  such that  $b > a$ . More specifically, the first two lines of this table should look like the following:

	1	2	3	4	5	6	7	8	9
1	1								
2	2	4							

4. Send in the URLs for the last two scripts.

## 3.5 Associative arrays

*PhP* also allows the array type, which is used a lot. Although quite similar to that in Java, *PhP* array is a bit nicer in the sense that it also supports *associative array*, i.e., its index can be a string, such as `$_POST['Contrib']`. This is particularly useful in database programming since a relational database table is implemented as an array, and it is natural to get to a row using its key value which is often a string.

Let's go through some of the basic operations related to the associative array type with the following `testArrayBasics.php`.

```

<HTML>
  <HEAD>
    <TITLE>This is to test array creation in PHP</TITLE>
  </HEAD>

  <BODY>
    <p> The easiest way is simply to assign a value to an array element. </p>
    <?php
      $my_array[1]="The first thing in my array that I just made.";
      echo $my_array[1];
    ?>

    <p> You can also create one explicitly by using the array() construct. </p>
    <?php
      $fruit_basket=array('apple', 'orange', 'banana', 'pear');
      echo $fruit_basket[1];
    ?>

    <p> We can also set the indices for the arrays created using the array()
      construction. </p>
    <?php
      $another_basket=array('red'=>'apple', 'orange'=>'orange',
                          'yellow'=>'banana', 'green'=>'pear');
      echo $another_basket['green'];
    ?>

    <p> Another way is to use a function that returns an array to create one. </p>
    <?php
      $my_another_array=range(1, 5);
      echo $my_another_array[3];
    ?>

    <p> Array puts things together, while list takes things apart.</p>
    <?php
      list($red_one, $yellow_one)=$fruit_basket;
      echo $red_one."<BR>". $yellow_one;
    ?>

    <p> We can also have the multi-d array concept.</p>

    <?php
      $cornucopia=array ('fruit'=>
                        array('red'=>'apple', 'orange'=>'orange',
                              'yellow'=>'banana', 'green'=>'pear'),
                        'flower'=>

```

```

        array('red'=>'rose', 'yellow'=>'sunflower',
              'purple'=>'Iris'));
echo $cornucopia['flower']['purple'];
echo "<BR>";

$kind_wanted='fruit';
$color_wanted='yellow';

//for the {} in the following expression, refer to pp. 138
//interpolation with curly braces.
echo "The $color_wanted $kind_wanted is {$cornucopia[$kind_wanted][$color_wanted]}";
echo "<BR>There are". count($cornucopia). ' elements in the array $cornucopia';
echo "<BR>But, there are ". count($cornucopia['fruit']). ' fruits'."<BR>";
?>

<p> It is easy to delete an element from an array by using the unset function.</p>
<?php
    echo $fruit_basket[1]. '<BR>';
    unset($fruit_basket[1]);
    if(IsSet($fruit_basket[1]))
        echo $fruit_basket[1];
    else echo 'Not found';
?>

</BODY>
</HTML>

```

We will also check out the `testArrayIteration.php` script to see how to make use of such structures as `foreach` to deal with all the elements in an array in a uniform way.

This is quite similar to what we did in Java, demonstrating a close relationship between a loop structure and the array data structure.

```

<HTML>
  <HEAD>
    <TITLE>This is to test Iteration with array</TITLE>
  </HEAD>

  <BODY>
    <p> Have an example as follows. </p>
    <?php
      $major_city_info=array();
      $major_city_info[0]='Caracas';
      $major_city_info['Caracas']='Venezuela';
      $major_city_info[1]='Paris';
    </?php>
  </BODY>
</HTML>

```

```

$major_city_info['Paris']='France';
$major_city_info[2]='Tokyo';
$major_city_info['Tokyo']='Japan';

//Now define a function, which we will see in detail
//in a later unit of this notes.
function city_by_number($number_index, $city_array){
    if(IsSet($city_array[$number_index])){
        $the_city=$city_array[$number_index];
        $the_country = $city_array[$the_city];
        print("$the_city is in $the_country<BR>");
    }
}

city_by_number(0, $major_city_info);
city_by_number(1, $major_city_info);
city_by_number(2, $major_city_info);
?>

```

<p> The interesting foreach loop. </p>

```

<?php
function print_all_foreach1($city_array){
    foreach($city_array as $name_value)
        print("$name_value<BR>");
}

function print_all_foreach2($city_array){
    foreach($city_array as $key_value=>$name_value)
        print("$key_value". "=>". "$name_value<BR>");
}

print_all_foreach1($major_city_info);
print "<BR>";
print_all_foreach2($major_city_info);
?>

```

<p> The current and next functions. Notice the classic way to go through a list. </p>

```

<?php
function print_all_next($city_array){
    //Process the first one.
    $current_item=current($city_array);

```

```

    if($current_item)
        print("$current_item<BR>");
    else print("There is nothing to print");

    //for the rest
    while ($current_item=next($city_array))
        print("$current_item<BR>");
}

```

```

print_all_next(&$major_city_info);
print "<BR>";
print_all_next(&$major_city_info);

```

?>

<p> The reset function. </p>

<?php

```

//Define another function
function print_all_reset($city_array){
    //Process the first one.
    $current_item=reset($city_array);
    if($current_item)
        print("$current_item<BR>");
    else print("There is nothing to print");

    //for the rest
    while ($current_item=next($city_array))
        print("$current_item<BR>");
}

```

```

//This is where the just defined function
//is used.
print_all_reset($major_city_info);
print "<BR>";
print_all_reset($major_city_info);

```

?>

<p> Reverse order with end and prev functions. </p>

<?php

```

//Yet one more function...
function print_all_backwards($city_array){
    //Process the first one.
    $current_item=end($city_array);
    if($current_item)
        print("$current_item<BR>");
    else print("There is nothing to print");

    //for the rest
    while ($current_item=prev($city_array))
        print("$current_item<BR>");
}

//... and where it is used.
print_all_backwards($major_city_info);
print "<BR>";
print_all_backwards($major_city_info);

```

?>

<p> Get out the keys and values with the key function. </p>

<?php

```

//Now you are not surprised with more functions.
function print_keys_values($city_array){
    //Process the first one.
    reset($city_array);
    $current_value=current($city_array);
    //Thus, the key function returns the key of the current
    //item in an array.
    $current_key=key($city_array);
    if($current_value)
        print("Key: $current_key; Value: $current_value<BR>");
    else print("There is nothing to print");

    //for the rest
    while ($current_value=next($city_array)){
        $current_key=key($city_array);
        print("Key: $current_key; Value: $current_value<BR>");
    }
}

```

```

    print_keys_values($major_city_info);
    print "<BR>";
    print_keys_values($major_city_info);
?>

```

<p> Get out the keys and values with the each function. </p>

```

<?php
    function print_keys_values_each($city_array){
        //Process the first one.
        reset($city_array);

        //for the whole list, notice that each sends back the current,
        //thus saving the code for the first one.
        while ($current_value=each($city_array)){
            $current_key=key($city_array);
            print("Key: $current_key; Value: $current_value<BR>");
        }
    }

    print_keys_values_each($major_city_info);
    print "<BR>";
    print_keys_values_each($major_city_info);
?>

```

<p> Walk through an array with the array\_walk function, which lets you apply any function to a named array. </p>

```

<?php
    function print_value_length($array_value, $array_key_ignored){
        $the_length=strlen($array_value);
        print("The length of $array_value is $the_length<BR>");
    }

    array_walk($major_city_info, 'print_value_length');
?>
</BODY>
</HTML>

```

### Labwork 3.5:

1. Play with the above scripts to understand all the features as mentioned within.
2. Come up with your own script that works with the nutrition of various fruits, such as apple, banana, pear, pineapple and watermelon. Technically, construct an associate

array with the names of the fruits as the indices, and for each of them, do some google to find out four nutrients, such as Vitamin *A*, Vitamin *B*<sub>12</sub>, Calcium and Iron. Then print out various facts, similar to what we did in the script `testArrayBasics.php`.

## 3.6 Strings

We do lots of strings in our work and life. It also plays an important part in database programming. Technically, a string is any array of characters.

There are several string based operations, such as the concatenation; and we will now go through with some of these operations with the `testStringOperation.php` script.

```
<HTML>
<head>
<Title>Test String Operations</Title>
</head>

<body>
<p>Test the concatenation operator.</p>

<?php
    $my_two_cents="I want to give you a piece of my mind ";
    $third_cent=" And another thing";
    print($my_two_cents . "... " . $third_cent);
?>

<p>An alternative of .= which adds things to the right</p>

<?php
    $my_two_cents="I want to give you a piece of my mind ";
    $third_cent=" And another thing";
    $my_two_cents.= $third_cent;
    print($my_two_cents);
?>

<p>Now, we see how to construct large chunk of words using the <<< operator,
    called heredoc. The last word EOT must start with the first column </p>
<?php
$my_string_var= <<<EOT
Everything in this rather unnecessarily wordy ramble of prose will be
incorporated into the string that we are building up inevitably, inexorably,
character by character, line by line, until we reach that blessed final line
which is this one.
EOT;
```

```
echo $my_string_var;
?>
```

<p>We can use this heredoc feature to construct simple forms.</p>

```
<?php
    $firstname='Zhizhang';
    echo <<<ENDOFFORM
    <Form Method=Post Action="{ $ENV['PHP_SELF'] }">
        <Input Type=text Name=FirstName Value=$firstname>
        <Input Type=SUBMIT Name=SUBMIT Value=SUBMIT>
    </Form>
ENDOFFORM;
?>
</body>
</HTML>
```

There are also several string based functions, and we will now talk about it a bit with the `testStringFunctions.php` script.

```
<HTML>
    <head>
        <Title>Test String Functions</Title>
    </head>

    <body>
    <p>Test the inspection operator. Notice the automatic conversion of the
        result of strlen back to a string.</p>
    <?php
        $short_string="This string has 29 characters";
        echo("\ "$short_string"\ "<BR>");
        print("It does have " . strlen($short_string) . " characters");
    ?>
```

<p>Test the `strpos` function, which returns the position of a substring in a target string. Notice that location starts with 0.</p>

```
<?php
    $twister = "Peter Piper picked a peck of pickled peppers";
    echo("\ "$twister"\ "<BR>");
    print("location of 'p' is " . strpos($twister, 'p') . '<BR>');
    print("location of 'p' in the reversed side is " .
        strrpos($twister, 'p') . '<BR>');
```

```

print("location of 'Piper' in the reversed side is " .
      strrpos($twister, 'Piper') . '<BR>');
print("location of 'q' is " . strpos($twister, 'q') . '<BR>');
?>

```

<p>Change the content of a string.</p>

```

<?php
  $myString="abcdefg";
  echo("\$myString\n<BR>");
  $myString[5]='X';
  echo("\$myString\n<BR>");
  $myString[4]="Y";
  echo("\$myString\n<BR>");
?>

```

<p>Test the comparator.</p>

```

<?php
  $s1="hey";
  $s2="HEY";
  $s3="het";
  echo("\$s1\n<BR>");
  echo("\$s2\n<BR>");
  echo("\$s3\n<BR>");

  if(!strcmp($s1, $s2))
    print("$s1 ". "is the same as ". "$s2");
  else print("<BR>$s1 ". "is not the same as ". "$s2");

  if(strcasecmp($s1, $s2))
    print("$s1 ". "is the same as ". "$s2");
  else print("<BR>$s1 ". "is not the same as ". "$s2");

  if(!strcmp($s1, $s3))
    print("$s1 ". "is the same as ". "$s3");
  else print("<BR>$s1 ". "is not the same as ". "$s3");
?>

```

<p>Test the searching function strstr(source, target), which sends back the portion in source that contains the first occurrence of target.</p>

```

<?php
  $s1="showsuponceshowstwtice";
  $s2="up";

```

```

    $s3="down";
    print("Result of looking for $s2: ". strstr($s1, $s2). "<BR>");
    print("Result of looking for $s3: ". strstr($s1, $s2). "<BR>");
?>

```

<p>Test the string slicing function substr(source, startPosition, length), which sends back the portion in source that contains length characters, starting from startPosition.

When startPosition is negative, it means the first character should be counted from the right-hand-side, starting with 1. When the length is negative, it means that the last character should be counted from the right-hand-side, starting with 0.

</p>

```

<?php
    $s1="Take what you need, and leave the rest behind<BR>";
    print $s1;
    print ( substr($s1, 23));
    print (substr($s1, 5, 13))."<BR>";

    $alphabetTest="abcdefghijklmnop";
    echo "<BR>Given the following string: ". "$alphabetTest<BR><BR>";
    print("3: " . substr($alphabetTest, 3). "<BR>");
    print("-3: " . substr($alphabetTest, -3). "<BR>");
    print("3, 5: " . substr($alphabetTest, 3, 5). "<BR>");
    print("3, -5: " . substr($alphabetTest, 3, -5). "<BR>");
    print("-3, -5: " . substr($alphabetTest, -3, -5). "<BR>");
    print("-3, 5: " . substr($alphabetTest, -3, 5). "<BR>");
?>

```

<p>Compare strstr(\$containing, \$contained) and substr(\$containing, strpos(\$containing, \$contained).</p>

```

<?php
    $containing="showsuponceshowsuptwice";
    $contained="up";
    print("Result of strstr($containing, $contained): ".
        strstr($containing, $contained). "<BR>");
    print("<BR>Result of substr($containing, strpos($containing, $contained): ".
        substr($containing, strpos($containing, $contained)). "<BR>");
?>

```

<p>String clean-up functions. Notice that the browser will automatically clean up the mess, but it can be observed by viewing the source sent back by the php.</p>

```
<?php
    $original="    More than meets the eye    ";
    $chopped=chop($original);
    $trimmed=ltrim($original);
    $trimmed=trim($original);
    print("The original is '$original'<BR>");
    print("Its length is " . strlen($original). "<BR>");
    print("The chopped version is '$chopped'<BR>");
    print("Its length is " . strlen($chopped). "<BR>");
    print("The trimmed version is '$trimmed'<BR>");
    print("Its length is " . strlen($ltrimmed). "<BR>");
    print("The trimmed version is '$trimmed'<BR>");
    print("Its length is " . strlen($trimmed). "<BR>");
?>
```

<p>String replacement functions. Notice that the browser will automatically clean up the mess, but it can be observed by viewing the source sent back by the php.</p>

```
<?php
    $firstEdition="Burma is similar to Rhodesia in at least one way.";
    $secondEdition=str_replace("Rhodesia", "Zimbabwe", $firstEdition);
    $thirdEdition=str_replace("Burma", "Myanmar", $secondEdition);
    print($firstEdition . "<BR>");
    print($secondEdition . "<BR>");
    print($thirdEdition . "<BR>");

    $trickyString="ABA is part of ABABA";
    $maybeTricked=str_replace("ABA", "DEF", $trickyString);
    print("<BR>Substitution result is '$maybeTricked'<BR>");
?>
```

<p>Let's check out a few other functions, such as substr\_replace(target, replacement, startPosition, length), strrev(toBeReversed), and str\_repeat(toBeRepeated).</p>

```
<?php
    print(substr_replace("ABCDEFGH", "-", 2, 3));
    echo "<BR>";
    print(strrev("ABCDEFGH"));
    echo "<BR>";
```

```
    print(str_repeat("ABCDEFGH", 3));
?>
```

<p>Let’s check out a few case related functions, such as strtolower(toBeLowered), strtoupper(toBeUppered), ucfirst(theFirstToBeUppered), and ucwords(theFirstOfEachWordToBeUppered).</p>

```
<?php
    $original="THEY DON'T KNOW THEY are SHOUTING";
    echo strtolower($original);
    $s1="make this link stand out";
    echo "<BR><B>strtoupper($s1)</B><BR>";
    $s2="polish is a word for which pronunciation depends on capitalization";
    echo  ucfirst($s2);
    $s3="truth or consequences";
    echo "<BR>While $s3 is a parlor game, ". ucwords($s3) . "
        is a town in New Mexico.<BR>";
    sprintf("<BR>While $s3 is a parlor game, ". ucwords($s3) . "
        is a town in New Mexico.<BR>");
?>
```

<p>Test out the quotemeta(string) function, which adds slashes to many special symbols.</p>

```
<?php
    $literalString='these characters ($ *) are very special to me \n<BR>';
    $qmString=quotemeta($literalString);
    echo $qmString;
?>
```

<p>Test out formatted printing mechanism.</p>

```
<pre>
    <?php
        $value=3.1415926;
        printf("%f, %10f, %-010f, %2.2f\n", $value, $value, $value, $value);
    ?>
</pre>
</body>
</HTML>
```

### Labwork 3.6:

1. Carefully read through the scripts, run the scripts. In particular, play with the “here-doc” feature, which we will use quite a bit later. Make sure you understand that it has to start with the very first position of a line.

2. Revise the scripts as you like to show your understanding of many of the string related features, both functions and operations, as used within.
3. Send in the URL of the revised script as well as the script itself, perhaps as a .txt file.

### 3.7 Functions

As we will see, it is very important to be able to define functions, method in Java, so that we can apply the divide and conquer technique to cut a problem to a bunch of smaller and simpler problems.

Below shows `testFunction.php`, which shows how to define a function and use it.

```
<HTML> <HEAD>
<TITLE>Test out function composition</TITLE>
</HEAD>

<BODY>
<p> Function composition is essentially the
      same as that in Java, except the parameter
      list does not specify with types.</p>

<?php
function better_deal($amount_1, $amount_2,
                    $price_1, $price_2){
    //Calculate the respective amount/price ratio
    $per_amount_1=$price_1/$amount_1;
    $per_amount_2=$price_2/$amount_2;

    return($per_amount_1 < $per_amount_2);
}

$liters_1=1.0; $liters_2=1.5;
$price_1=1.59; $price_2=2.09;

if(better_deal($liters_1,$liters_2,
               $price_1, $price_2))
    print("The first is better!<BR>");
else print("The second is better!<BR>");
?>
</BODY>
</HTML>
```

**Labwork 3.7:**

1. Test out the above function to understand this piece.
2. Write a complete script containing a couple of meaningful functions and their invocation. For example, you can implement the exponential operation, and demonstrate how to invoke this function in calculation.
3. Send in the URL of your script, as well as the script itself, perhaps as a `.txt` file.

### 3.8 The scope of variables

Just as in Java, all the variables defined in a function are **local**, i.e., its *scope* is only within the function where it is defined. It no longer exists once the execution of the function completes. Technically, such variables are allocated in a system stack when the function is called. When the function completes its execution, the space allocated to such variables are retracted.

```
<?php
function SayMyABC(){
    //Define a local variable $count and give it
    //an initial value of 0.
    //The scope of this one is only with the
    //function SayMyABC
    $count=0;
    while ($count<10){
        //Notice the following conversion from
        //number to characters and the function chr
        print(chr(ord('A')+$count));
        $count++;
    }
    //When the loop ends, the value of $count is 10.
    print("<BR>Now, I know my ABC better<BR>");
}
//This $count has nothing to do with that $count
$count=1;
SayMyABC();
print("<BR>Now, we have made ($count) calls.<BR>");

$count++;
//Now the value of $count becomes 2
SayMyABC();
print("Now, we have made ($count) calls.<BR>");
?>
```

**Labwork 3.8.1** Read and get the output of the above script, then explain *why* it generates such an output. For example, assume the following is the output. Explain *how* it is generated by this script.

```
ABCDEFGHIJ
```

```
Now, I know my ABC better
```

```
Now, we have made (1) function calls .
```

```
ABCDEFGHIJ
```

```
Now, I know my ABC better
```

```
Now, we have made (2) function calls .
```

If we do want a connection between two pieces of code, we make it **global**, its scope then is with the php *page* where it is defined. For example, the following `testFunctionScopeGlobal.php` shows how to do it.

```
<HTML>
  <HEAD>
    <TITLE>Test the function scope</TITLE>
  </HEAD>
  <BODY>
    <p> Scope of variables in php function is essentially the same as that in Java.
    The values of the actual parameters are copied into the formals. The connection is
    then cut. </p>

    <?php
    function SayMyABC(){
      //Define a global variable, w/o giving it an initial.
      global $count;

      while ($count<10){
        //Notice the conversion from number to characters and the function chr
        print(chr(ord('A')+$count));
        $count++;
      }
      //When the loop stops for the first invocation,
      // $count contains a 10
      print("<BR>Now, I know $count letters<BR>");
    }

    //The following variable is the same as that in
    //the SayMyABC function.
```

```

$count=0;
SayMyABC();
//Now, $count contains a 10
print("<BR>Now, we have made $count function calls .<BR>");

$count++;
//$count now contains 11.
print('$count now contains 11. Once in, it will not print out anything');
print("since its value is larger than 10.");
SayMyABC();
//The $count still contains 11
print("Now, we have made $count function calls .<BR>");
?>
</BODY>
</HTML>

```

**Labwork 3.8.2** Read and get the output of the above script, then explain *why* it generates such an output. For example, assume the following is the output. Explain *how* it is generated by this script.

```

ABCDEFGHIJ
Now, I know 10 letters

```

```

Now, we have made 10 function calls .
$count now contains 11.
Now, I know 11 letters
Now, we have made 11 function calls .

```

As aforementioned, a local variable disappears once the function completes. If we want to *remember its value in the previous invocations*, we make it **static**. Static variables are different from global ones in the sense that its scope is still with the function its if defined, thus it is still local. But, it is deferent from local, in the sense that it is defined through out the execution of the whole program.

Let's see another example.

```

<?php
function SayMyABC3(){
//The following $count is initialized only once when the function is
//called for the first time.
static $count=0;
//$limit is local, it is set to the current value of $count plus 10.
$limit=$count+10;
//When the function is called first time, $limit is set to 10.
//When it is called a second time, it is set to 20

```

```

while ($count<$limit){
    print(chr(ord('A')+$count));
    $count++;
}
//When the loop ends, $count contains the same value as that for $limit
//Thus, for the first time, it is 10, then 20
print("<BR>Now, I know $count letters");
}
//The following count is different from the $count in SayMyABC3
$count=0;
SayMyABC3();
//This $count still contains 0
$count++;
//Now, $count contains 1
print("Now, we have made $count calls .<BR>");
SayMyABC3();
$count++;
//It contains 2
print("Now, we have made $count calls .<BR>");
?>

```

**Labwork 3.8.3** Read and get the output of the above script, then explain *why* it generates such an output. For example, assume the following is the output. Explain *how* it is generated by this script.

ABCDEFGHIJ

Now, I know 10 letters

Now, we have made 1 function calls .

KLMNOPQRST

Now, I know 20 lettersNow, we have made 2 function calls .

### 3.9 Recursion

A function is *recursive* if the same function is used in defining itself. For those recursion fan, *PHP* does it, too. Below is an example, `testRecursion.php`.

```

<?php
function countdown($num_arg){
    if ($num_arg>0){
        print("counting down from $num_arg<BR>");
        countdown($num_arg-1);
    }
}

```

```
}  
  
    countdown(10);  
?>
```

**Question:** What should be the output?

**Answer:**

```
counting down from 10  
counting down from 9  
counting down from 8  
counting down from 7  
counting down from 6  
counting down from 5  
counting down from 4  
counting down from 3  
counting down from 2  
counting down from 1
```

### Labwork 3.9:

1. Test out the above code to understand the way to do it.
2. Write a function to calculate the factorial of a given number, and test it out by printing out the factorial of 1 through 20.

## 4 Data passing in *PhP*

One of the most important thing about WEB is that it is *stateless*, i.e., it does not have memory for the values of all the variables as contained in any page. Each http call is independent of all the others. Thus, values are not transmitted except explicitly arranged. On the other hand, we certainly wish to transmit values between simpler and smaller pages, if we don't want to work with just one **huge** web page.

There are two ways to passing values between pages, **GET** and **POST**, each has its own advantage and disadvantage.

### 4.1 The GET method

The **GET** approach passes arguments between pages as part of the URI (UR Indicator) by adding the value to the end of a URL. Let's look at an example, which selects a team and then display a corresponding message.

In the following script, we use a structure called *form*. A form is a section of an *HTML* document that contains normal content, markup, special elements called *controls*, such as

checkboxes, radio buttons, menus, etc., and labels on those controls, together with some blanks for users to complete by, e.g., entering text, and/or selecting check boxes, etc., before it is submitted to a Web server for processing<sup>1</sup>

```
<HTML><HEAD>
  <!--This is the testGet.php-->
  <TITLE>A GET method example</TITLE></HEAD>
  <BODY>
    <!--Below defined how to pass, with whose help-->
    <Form Method="GET" Action="baseball.php">
      <!--Start with a normal paragraph-->
      <p>root, root, root for the:<BR>  </p>
      <!--Below is a Select list, or rather the familiar Java drop box,
        with two options-->
      <Select Name="Team" Size="2">
        <!--The first option has the index being "Chicago Cubs"
          and the value being "Cubbies"-->
        <!--If we select Chicago Cubs, the value of -->
        <!--"Cubbies" will be selected -->
        <Option Value="Cubbies">Chicago Cubs</Option>
        <Option Value="Pale Hose">Chicago White Sox</Option>
      </Select>
      <!-- The selected value will be sent out via $_GET['Team'] -->
      <p><Input Type="submit" Name="Submit" Value="Select"></p>

    </Form>
  </BODY></HTML>
```

Here is the `baseball.php`, the *form handler*.

```
<HTML>
  <HEAD>
    <TITLE>A GET method example(II)</TITLE>
    <Style type="text/css">
      <!-- Body {font-size:24pt;}-->
    </Style>
  </HEAD>
  <BODY>
    <!-- The 'Team's have to match-->
    <p>Go, <?php echo $_GET['Team']; ?> </p>
  </BODY>
</HTML>
```

---

<sup>1</sup>I took a definite of the *form* from [6, §17.1], and did some rewriting.

If everything is in the right place, we should open the following:

```
http://turing.plymouth.edu/~j_doe/PhPFiles/newsletter_signup.html.
```

We notice the following in the browser Address line:

```
http://localhost/baseball.php?Team=Cubbies&Submit=Select, where the stuff after the '?' shows the Get string.
```

**Question:** Do you really want to show the public this information?

The **GET** method for data passing constructs a new and different URL, which can be bookmarked, while the result obtained via the other method, **Post**, can't be. But, there are at least two issues with the **Get** method which we have to know.

- The **GET** method is not appropriate in a security sensitive environment such as logins because....
- This method assigns a value to a server environment variable `$_GET`, thus the length of the URL is limited as the type of such a variable specifies, currently at 255 characters.

#### Labwork 4.1:

1. Test out the script.
2. Write a pair of scripts. One, similar to the above `testGet.php`, contains just one input box to receive an integer, and a submit button, plus necessary explanation message. The other, similar to the above `baseball.php`, gets, using the **GET** method, a value passed over from the first file, then calculates the factorial of this passed value. The calculation part can look like the following:

```
<html>
Factorial calculation<BR><BR>
<?php

    function factorial($arg){
        if($arg==0)
            return (1);
        else return (factorial($arg-1)*$arg);
    }

    $count=1;

    while ($count<=20){
        print("The factorial of $count is ".factorial($count));
        print("<BR>");
        $count++;
    }
?>
</html>
```

3. Write a pair of scripts. One contains a list of courses, and uses the GET method to pass your choice to another one, which prints out some associated message about the chosen course, e.g., its description.

## 4.2 The Post method

In contrast, the POST approach will never make the user entered string visible in the URL query string or any where else. Thus, it is much more secure. Moreover, with this method, much larger chunk of string can be passed.

But, its result can't be bookmarked, and will be gone when you click the back button in a browser.

Finally, it can be incompatible with certain firewall setups, which strip the form data as a security measure.

As an example, let's apply the Post method to the previous application. We begin with the form.

```
<HTML>
  <HEAD>
    <!--This is the testGetPost.php-->
    <TITLE>A GET method example</TITLE></HEAD>
    <BODY>
      <!--We now send out the message with the
        the Post method with a different handler -->
      <Form Method="POST" Action="baseballPost.php" >
        <p>root, root, root for the:<BR>  </p>

        <Select Name="Team" Size="2">
          <!--If we select Chicago Cubs, the value of -->
          <!--"Cubbies" will be selected -->
          <Option Value="Cubbies">Chicago Cubs</Option>
          <Option Value="Pale Hose">Chicago White Sox</Option>
        </Select>
        <p><Input Type="submit" Name="Submit" Value="Select"></p>
        <!-- The selected value will be sent out via $_POST['Team'] -->
      </Form>
    </BODY>
</HTML>
```

Here is the baseballPost.php, the form handler.

```
<?php
  $team=$_POST['Team'];
?>
```

```

<HTML>
  <HEAD><TITLE>A Post method example(II)</TITLE>
    <Style type="text/css">
      <!--Body {font-size:24pt;}-->
    </Style></HEAD>

  <BODY>
    <p>Go, <?php echo $team; ?> </p>
  </BODY>
</HTML>

```

When we run the above script, we don't see any information about the data just sent over.

**Labwork 4.2:** Redo Labwork 4.1 with the POST method.

### 4.3 An example

Let's check out a retirement calculation script, `calcRetirement.php`, to see how to use the POST method to pass multiple values. Moreover, we will see how to combine the interface and the handler into one page.

```

<HTML>
  <HEAD>
    <TITLE>Retirement Savings Worksheet</TITLE>
    <Style type="text/css">
      <!--Body {font-size:14pt;}
        .heading {font-size: 18 pt; color: red}-->
    </Style>
  </HEAD>
  <?php
    //Set defaults in case of error, when the 'Submit' button
    //is not clicked, or if the value sent over is not
    //'Calculate'
    if(!isset($_POST['Submit']))
      || $_POST['Submit'] != 'Calculate'){
      $_POST['CurrentAge'] = "";
      $_POST['RetireAge'] = "";
      $_POST['Contrib'] = "";
      $Total=0; $AnnGain=7;
    }
    else {
      //Get the values via the POST method
      $AnnGain=$_POST['AnnGain'];

```

```

//Remaining years
$Years=$_POST['RetireAge']
        -$_POST['CurrentAge'];
$YearCount=0;
//How much saved so far
$Total=$_POST['Contrib'];

//Calculate the total
while($YearCount<=$Years){
    $Total=round($Total*(1.0+$AnnGain/100)+$_POST['Contrib']);
    $YearCount++;
    $Total;
}\\\end of while
}\\\ end of else
?>
<BODY>
<!--The <Div> tag defines a division/section in a document, with its own
characteristic. The specification applies to everything until a new
division starts.
-->
<Div Align="CENTER" ID="Div1" class="heading">
    A retirement-savings calculator
</Div>

<p class=blurb>Fill in all the values (except "Nest Egg") and see how much
money you'll have for your retirement.</p>

<!--How to contain its own handler-->
<Form Method="POST" Action="<?php echo $_SERVER['PHP_SELF']; ?>" >
    <p>Your age now:
        <Input Type="text" Size=5 Name="CurrentAge"
            Value="<?php echo $_POST['CurrentAge']; ?>">
    <p>The age at which you plan to retire:
        <Input Type="text" Size=5 Name="RetireAge"
            Value="<?php echo $_POST['RetireAge']; ?>">
    <p>Annual contribution:
        <Input Type="text" Size=5 Name="Contrib"
            Value="<?php echo $_POST['Contrib']; ?>">
    <p>Annual return:
        <Input Type="text" Size=5 Name="AnnGain"
            Value="<?php echo $_POST['AnnGain']; ?>">%
<BR><BR>

```

```

<p><B>Nest Egg</B>: <?php echo $Total; ?>
<p><Input Type="submit" Name="Submit"
      Value="Calculate">
</Form>
</BODY>
</HTML>

```

One of the benefits of combining interface and the handler together is that we can get access to all the variables defined for that page, change them without causing any confusion.

### Labwork 4.3:

1. Test out this program.
2. Use this program as an example to write an application that calculate the mortgage payment. You have to get the amount of the loan, interest rate, and the length of the loan, to find out *how much principle are left at what time (which month, which is also an input. For example, if the loan starts in October, 2006, and if you want to know how much you will still owe to the bank in October, 2007, you have to find out how much you still owe after 12 months.*

Below gives the formula, taken from [2], to calculate mortgage for U.S. based properties, assuming a typical conventional loan where the interest is compounded monthly.

We need the following inputs:

- *Principal, P*, the initial amount of the loan;
- *Annual interest rate, I* (from 1 to 100 percent);
- *Length, L*, the length (in years) of the loan.

We can then derive two more quantities:

- *monthly interest in decimal form,*

$$J = I / (12 \times 100);$$

- *number of months over which loan is amortized,*

$$N = L \times 12.$$

With the above quantities, the *monthly payment formula*, M, is given as follows:

$$M = P \left[ \frac{J}{1 - (1 + J)^{-N}} \right].$$

I once asked you to implement the exponential function in Labwork 3.7 to calculate `exponential(a, n)`, you can now implement the mortgage calculation as the following:

$$M = P * ( J / (1 - (1/\text{exponential}(1 + J, N))))).$$

We can go through the following process to find the remaining principle for the month you are interested, or until the principle has come down to 0:

- (a) Set  $P$  to  $P_0$ , the starting principle.
- (b) Find out,  $r$ , the number of times you have to repeat the following process, then repeat the following steps:
- (c) Calculate  $H = P \times J$ , this is your monthly interest.
- (d) Calculate  $C = M - H$ , this is your monthly payment minus your monthly interest, so it is the amount of principal you have to pay for the following month
- (e) Let  $Q = P - C$ , this is the new balance of your principal.
- (f) Set  $P$  equal to  $Q$  and go back to Step (c).

#### 4.4 Another example: a quiz

Let's look at an example, in `geekquiz.php`, which combines data passing with the `Post` method, the usage of `Check Boxes`, the `array` type, and a *massive* usage of *heredoc* strings,

```
<?php
/*****
 * "How geeky are you?" script, showing with screens. *
 * Screen 1: quiz form. Screen 2: results page      *
 *****/

//We start by defining three strings, for the header, the footer
//and for the quiz panel.

//the header which appears in both cases
//-----

$header_str = <<<EOHEADER
<HTML
<HEAD>
<Style Type="text/css">
<!--
    Body, P, TD {color: black; font-family: verdana; font-size: 9 pt}
    H1   {color: black; font-family:arial; font-size: 12pt}
-->
</Style>
</HEAD>

<Body>
```

```

<Table Border=0 CellPadding=10 Width=100%>
  <Tr>
    <!--the width of the light blue stuff-->
    <Td BgColor="#F0F8FF" Align=Center VAlign=top Width=150>
      </Td>

    <!--The second item, the main panel-->
    <Td BgColor="FFFFFF" Align=Left VAlign=Top width=83%>
      <table cellspacing=0 cellpadding=20 border=0 width="530">
        <tr>
          <td valign=top>
EOHEADER;

//The footer which appears in both cases
//-----
$footer_str=<<< EOFooter
      </td>
    </tr>
  </table>
</Td>
</Tr>
</Table>

</Body>
</HTML>
EOFOOTER;

//Screen 1: quiz form
//-----

$quiz_str=<<< EQQUIZ
<h2>How geeky are you?</h2>
<form action="geek_quiz.php" method="POST">
<br /><br />

  <!-- If checked, the value of affirm[0] is set to 1, otherwise, set to 0 -->
  0. Have you ever had a dream in which you were debugging?<br />
    Yes <input type="checkbox" name="affirm[0]" value="1"/>
    <!--one spacing -->
    <br /><br />
  1. Do you know the name of the company founded by Danny Hillis?<br />
    Yes <input type="checkbox" name="affirm[1]" value="1"/>
    <br /><br />
  2. can you edit a file in both emacs and vi without recourse to any documentation?<br />

```

```

    Yes <Input type="checkbox" name="affirm[2]" value="1"/>
    <br /><br />
3. Is the computer you're using at this moment hooked up to a IVM switch?<br />
    Yes <Input type="checkbox" name="affirm[3]" value="1"/>
    <br /><br />
4. Are you wearing a logowear T-shirt?<br />
    Yes <Input type="checkbox" name="affirm[4]" value="1"/>
    <br /><br />
5. Have you ever written a chess program?<br />
    Yes <Input type="checkbox" name="affirm[5]" value="1"/>
    <br /><br />
6. Have you ever set up an SMTP server?<br />
    Yes <Input type="checkbox" name="affirm[6]" value="1"/>
    <br /><br />
7. Have you ever discussed the merits of a commercial LISP implementation?<br />
    Yes <Input type="checkbox" name="affirm[7]" value="1"/>
    <br /><br />
8. Have you ever used the phrase "I can do that in two lines of code" in public?<br />
    Yes <Input type="checkbox" name="affirm[8]" value="1"/>
    <br /><br />
9. Have you ever refused an otherwise welcome sexual advance because you
    were debugging?<br />
    Yes <Input type="checkbox" name="affirm[9]" value="1"/>
    <br /><br />
    <input type="submit" name="submit" value="Evaluate">
</form>
EOQUIZ;

//-----
// Now the script itself
//-----

//Show the header first
echo $header_str;

//If the submit button has yet to be clicked, show them the quiz panel
if (!isset($_POST['submit'])) {
    //First time, show the quiz form
    echo $quiz_str;
}
//Otherwise, do some statistic, then construct the appropriate result string.
elseif ($_POST['submit']=='Evaluate'){
    //Count up the yes answers, by counting up all those with a 1 inside.
    $num_affirm=count($_POST['affirm']);

```

```

//come up with 4 different blurbs, which can certainly be improved.
if($num_affirm>=0 && $num_affirm<=3)
    $result_str="<p>Why even pretend to be something you are so clearly not?</p>\n";
elseif ($num_affirm>=4 && $num_affirm<=6)
    $result_str= "<p>Come back when you have learned more craft, Grasshopper.</p>\n";
elseif ($num_affirm>=7 && $num_affirm<=8)
    $result_str= "<p>Pretty geeky, but not yet a Code God.</p>\n";
elseif ($num_affirm>=9 && $num_affirm<=10)
    $result_str= "<p>We are not worthy to be in the presence of your bad geeky self!</p>\n";

//Send back the result
echo $result_str;
}
//Now the footer
echo $footer_str;
?>

```

#### Labwork 4.4:

1. Test out all the scripts.
2. Use the last piece as an example to come up with an evaluation with a subject of your choice with 10 questions, and based on the number of correct answers, give out appropriate summary results.

## 4.5 More data structures

We have so far looked some *PHP* programming examples, with the most complicated data structure being an array of check boxes. Although there is no way we can show everything, let's look at several other structures that we can use to solve a rather complicated problem related to an exercise calculator. We can use either *radio buttons*, as shown in a pair of files `workout_calc_radio.html`, and `wc_handler_ckbx.php`; or *check boxes* as shown in `workout_calc_radio2.html`, and `wc_handler_arrays.php`; or an *multi-dimensional array* as shown in `workout_calc_multi_array.html`, and `wc_handler_multi_array.php`.

This example also shows how to present a more descriptive and user friendly output format.

These files are too big to be included here, and can be found in the Lab work page in the course page.

#### Labwork 4.5:

1. Test out all the scripts.
2. Use the last piece as an example to come up with a quiz with 10 questions, and based on the number of correct answers, give out a grade, "Excellent" if she gets at least 8 right; "Good" if at least 6 right, "Fair" if at least 4 right; "Work harder" otherwise.

## 5 Work together with *MySQL*

*MySQL*[3] is an implementation of SQL, and its current version, as I am writing down this words on June 1, 2006, is 5.1 Beta. Please refer to [5] for much more details of using *PHP* with *MySQL* to do database programming.

For some basic *MySQL* commands, please refer to the *MySQL* labnotes. For a comprehensive discussion of the system, please refer to [3].

### 5.1 A basic procedure

Besides using *MySQL* in the command line interface, it is more realistic to work with *MySQL* based databases, with *HTML* based interface, and using a *PHP* script to pass data and set up some basic control logic. A typical process could be the following:

1. Establish a database connection with the database(s) you want to work with.
2. Select the table(s) you want to work with.
3. Construct a query to send to the database.
4. Get the result.
9. Close of the database connection.

You also need to go through the following steps in between Step 4 and 9, if using *HTML* to generate a more formatted output.

5. Start an *HTML* table, using the `<Table>` tag.
6. Loop through the database result rows, with a cursor, and place it into each row of the table, using a `<Tr></Tr>` structure.
7. In each row, retrieve the successive fields and place it into that row, using a `<Td></Td>` structure.
8. Close off the *HTML* table, using the `</Table>` tag.

### 5.2 An example: a sign-up service

Let's assume that we want to provide a service that users can sign up so that, when we have something to post, we could send over the content. This is an example of static SQL programming, in the sense that before the processing starts, we have a fixed query in our mind.

We first have to create the following database table for the `test` database, in *MySQL*, as follows:

```
mysql>create table mailinglist (
  ID int not null auto_increment primary key,
  Email varchar(30),
  Source varchar(50)
);
```

Once entered, its structure should look like the following:

```
mysql> desc mailinglist;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID    | int(11)       |      | PRI | NULL    | auto_increment |
| Email | varchar(30)   | YES  |     | NULL    |                |
| Source | varchar(50)   | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

To let our users easily use this sign-up page, we have to write up a user friendly interface and, maybe a *PhP* script to hook up this interface and the mailinglist database. There are a few ways to do it. A simpler one is to cut the task explicitly into two pieces: an *HTML* script for the interface, and an *handler* in *PhP* as the go-between, besides the database as already defined.

Below presents a possible design of the interface, `newsletter_signup.html`.

```
<HTML>
<Head>
  <Style Type="text/css">
    <!--
      Body, P, TD {color: black; font-family: verdana; font-size: 10 pt}
      H1 {color: black; font-family: arial; font-size:12 pt}
    -->
  </Style>
</Head>

<!--A table with only one row, consisting of two cells, the first being the -->
<!--left edge, 1/6; and the other contains the form, 5/6 -->
<Table Border=0 cellPadding=10 Width=100%>
  <!--Now define the row-->
  <Tr>
    <!--The following cell gives the left cushion edge-->
    <Td BgColor="F0F8FF" Align=Center VAlign=top Width=17%> </Td>

    <!--The following gives the right entry form part, completely white-->
```

```

<Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>

<H1>Newsletter sign-up form</H1>
<p>Enter your email address and we will send you our weekly newsletter.</p>

<!--Data will be transmitted via the post method, and will be handled by the-->
<!-- specific file. We will talk in detail about such methods, later -->
<Form Method="post" Action="formhandler1.php">
  <!--First input box, with a name "email", thus earmarked with the name -->
  <!-- $_POST['email'] in the handler. The value will be filled in by the user -->
  <Input Type="text" Size=25 Name="email">
  <BR><BR>
  <!--The submit button, with the name being $_POST['submit'], -->
  <!--and the value shown being 'Submit'-->
  <Input Type="submit" Name="submit" Value="Submit">
</Form>

</Td>
<!--end of the row definition>
</Tr>
</Table>

</Body>
</HTML>

```

Now, we have the following as a *The handler* of the above interface, `formhandler.php`,

```

<?php
  //Display an error message if either the email
  //input box is empty or contains an empty
  //string, or too long

If(!isset($_POST['email'])||$_POST['email']=="" or strlen($_POST['email'])>30)
  echo '<p>Did you enter a proper address? </p>';
else {
  //1. open connection to the database
  $my_connection=mysql_connect("localhost","zshen","password")
  or die("Fail to communicate with database");

  //2. Select the test database
  mysql_select_db("zshen");

  //Add backslash to every quotes.
  $as_email = addslashes($_POST['email']);

```

```

//Get rid of all the spaces
$str_email=trim($as_email);

//3. A query to enter a record in the mailinglist table.
$query="Insert Into mailinglist(Email, Source)
Values
( '$str_email',
'www.example.com/newsletter_signup.html')";

//4. Execute the query. This is where MySQL kicks in.
$result=mysql_query($query);

//Here is the post processing part. In this case, we just
//check to see if the query is done properly, namely,
//if only one row is affected.
if(mysql_affected_rows()==1)
    echo '<p>Your information has been recorded. </p>';
else {
    error_log(mysql_error());
    echo mysql_error();
    echo '<p>Something went wrong with
        your signup attempt.</p>';
} //end if inserted
//9. close this connection
mysql_close($my_connection);
}
//end if entry error
?>

```

## Labwork 5.2:

1. Set up the database table in your group database.
2. Put in all the scripts to you-know-where.
3. Modify the scripts so that it runs.
4. Log into your group's mysql account to verify it indeed adds in the data into the database table `mailinglist`.

## 5.3 Find out the best and the worst

We have repeatedly shown how to find out the students with the highest GPA and the ones with the lowest. This is also an example of static SQL programming.

Here is how to get it done. Below is the `sendSpecificQuery.html` file.

```

<Html>
<!--This is to send a query, which will be picked up and processed by -->
<!--another script.php -->
<Head>
  <Style Type="text/css">
    <!--
      Body, P, TD {color: black; font-family: verdana; font-size: 10 pt}
      H1 {color: black; font-family: arial; font-size:12 pt}
    -->
  </Style>
</Head>

<!--A table with only one row, consisting of two cells, the first being the -->
<!--left edge, 1/6; and the other contains the form, 5/6 -->
<Table Border=0 cellPadding=10 Width=100%>
  <!--Now define the row-->
  <Tr>
    <!--The follwoing cell shows the left cushion edge-->
    <Td BgColor="FOF8FF" Align=Center VAlign=top Width=17%> </Td>

    <!--The following gives the right entry form part, completely white-->
    <Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>

    <H1>This html script shows how to generate results for simple queries.</H1>
    <p>The following button finds the student(s) with the highest GPA.</p>

    <p>The associated SQL code is the following:

    <em> Select S.Name, S.Id From Student S
      Where S.GPA >= (Select Max(S1.GPA)
        From Student S1)</em></p>
    <!--The following says that this section is going to be -->
    <!--handled by a separate php script.-->
    <Form Method="post" Action="highestGPA.php">

    <BR><BR>
    <!--The sumbit button, with the name being $_POST['submit'], -->
    <!--and the value shown being 'Highest GPA'-->
    <Input Type="submit" Name="submit" Value="Highest GPA">
  </Form>

```

```
<hr><hr>
```

```
<p>The following button finds the student(s) with the lowest GPA.</p>
```

```
<p>The associated SQL code is the following:
```

```
<em> Select S.Name, S.Id From Student S
      Where S.GPA <= (Select Min(S1.GPA)
                    From Student S1)</em></p>
```

```
<!--The following says that this section is going to be-->
```

```
<!--handled by another php script.-->
```

```
<Form Method="post" Action="lowestGPA.php">
```

```
<BR><BR>
```

```
<!--The submit button, with the name being $_POST['submit'], -->
```

```
<!--and the value shown being 'Lowest GPA'-->
```

```
<Input Type="submit" Name="submit" Value="Lowest GPA">
```

```
</Form>
```

```
</Td>
```

```
<!--end of the row definition>
```

```
</Tr>
```

```
</Table>
```

```
</Body>
```

```
</Html>
```

Now, we present the two scripts that are associated with the two buttons, each of which is for a static SQL statement. First, the `highestGPA.php`:

```
<?php
```

```
//This segment gets in a function used to print out the content of
```

```
//a single table
```

```
include("displayQueryResult.inc");
```

```
?>
```

```
<Html>
```

```
<Head>
```

```
<Title>Student(s) with highest GPA</Title>
```

```
</Head>
```

```
<Body>
```

```
<!--The following setting ensures the interface consistency -->
```

```
<Table Border=0 cellPadding=10 Width=100%>
```

```
<!--Now define the row-->
```

```

<Tr>
  <!--The follwoing cell gives the left cushion edge-->
  <Td BgColor="F0F8FF" Align=Center VAlign=top Width=17%> </Td>

  <!--The following gives the right entry form part, completely white-->
  <Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>
    <!--Now the content of this cell-->
    <?php
      //This is the query to get the highest GPA student(s)
      $query_string="Select S.Name, S.Id, S.GPA From Student S
                    Where S.GPA >= (Select Max(S1.GPA)
                    From Student S1)";

      print("The following displays students with the highest GPA.<BR><BR>");

      //Call the predefined function to print out the cellar table, together
      //with column titles and an appropriate border
      display_db_query($query_string, $global_dbh, TRUE, "Border=2");
      mysql_close($global_dbh);
    ?>
  </Td>
</Tr>
</Table>
</Body>
</Html>

```

Now, the other part lowerGPA.php:

```

<?php

  //Include all the functions needed to print out the content of a single table
  include("displayQueryResult.inc");
?>

<Html>
  <Head>
    <Title>Student(s) with highest GPA</Title>
  </Head>
  <Body>

```

<!--A table with only one row, consisting of two cells, the first being the -->

```

<!--left edge, 1/6; and the other contains the form, 5/6 -->
<Table Border=0 cellPadding=10 Width=100%>
  <!--Now define the row-->
  <Tr>
    <!--The follwoing cell gives the left cushion edge-->
    <Td BgColor="F0F8FF" Align=Center VAlign=top Width=17%> </Td>

    <!--The following gives the right entry form part, completely white-->
    <Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>
<?php

    $query_string="Select S.Name, S.Id, S.GPA
                    From Student S
                    Where S.GPA <= (Select Min(S1.GPA)
                                   From Student S1)";

    print("The following displays students with the lowest GPA.<BR><BR>");

    //Call the predefined function to print out the cellar table,
    //together with column titles and an appropriate border
    display_db_query($query_string, $global_dbh, TRUE, "Border=2");
?>
</Td>

    </Td>
  <!--end of the row definition>
</Tr>
</Table>

</Table>
</Body>
</Html>

```

## 5.4 Another example: on countries and cities

Let's demonstrate the above procedure by using a database of countries and cities. We have collected some information about countries, the continent they are located, and a few cities in each of those countries.

ID	continent	country
1	Africa	Kenya
2	South America	Brazil
3	North America	USA
4	North America	Canada

Its structure can be created as follows in *MySQL*:

```
mysql>Create table Country (
  ID Int(11) Not Null Auto_Increment Primary key,
  continent Varchar(50),
  countryname Varchar(50))
```

Notice *Auto\_Increment* is a special feature provided by *MySQL*, which provides an automatically incremented counter for the field.

This is how to check out its structure as recorded by *MySQL*:

```
mysql> desc country;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID         | int(11)       |      | PRI | NULL    | auto_increment |
| continent  | varchar(50)   | YES  |     | NULL    |                |
| countryname | varchar(50)  | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Below shows how to create an *MySQL* structure for the *City* table,

```
mysql>Create table City (
  ID Int(11) Not Null Auto_Increment Primary key,
  countryID Varchar(50),
  cityname Vachar(50))
```

Once created, the following is what *MySQL* knows about this table.

```
mysql> desc city;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID         | int(11)       |      | PRI | NULL    | auto_increment |
| countryID  | int(11)       |      |     | 0       |                |
| cityname   | varchar(50)   | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Assume it comes with the following data, you should know how to insert them into the `city` table. If you don't, check out Part (II) of the lab notes, *A Gentler Introduction to MySQL Database Programming*.

ID	countryID	cityname
1	1	Nairobi
2	1	Meru
3	1	Mombasa
4	2	Rio
5	2	San Paulo
6	2	Salvador
7	3	Boston
8	3	Chicago
9	3	Houston
10	4	Windsor
11	4	Montreal
12	4	Winnipeg

As an example of a query, we often have to print out the stuff from a table. We collect the relevant code in a file `singleTableDisplay.inc`. Notice that we don't do database connection in this function. Otherwise, if we call such a function multiple times, we will get into an inefficient situation.

```
<?php
include("home/phpbook/phpbook-vars.inc");

function display_db_table($tablename,$connection)
{
//3. A query to get all the stuff for the table
$query_string="Select * from $tablename";

//4. Get the restful by executing the query
$result_id=mysql_query($query_string,$connection);
//How many columns?
$column_count=mysql_num_fields($result_id);

//5. Start a table
print("<Table border=1>\n");

//6. For each row contained in the result
while($row= mysql_fetch_row($result_id)){
print("<Tr Align=left Valign=top>");

//7. Get columns for each row
```

```

for($column_num=0; $column_num< $column_count;
    $column_num++)
    print("<Td>$row[$column_num]</Td>\n");

//8. Finish off
print("</Tr>\n");
}
print("</Table>\n");
}
?>

```

The following file is named `testSingleTableDisplayClass.php`.

```

<?php
include("singleTableDisplay.inc");

$db="geography";

//1. Set up the db connection
$global_dbh=mysql_connect($hostname,
    $user, $password);

//2. Connect to the database
mysql_select_db($db, $global_dbh);
?>

<HTML><Head><Title>Cities</Title>
</Head>
<Body>
<Table><Tr><Td>
<?php
    //Call the function
    display_db_table("city", $global_dbh);
?>
</Td></Tr>
</Table>
<?php
    //9. close off this connection
    mysql_close($global_dbh);
?>
</Body>
</HTML>

```

**Question:** What is contained in the file `home/phpbook/phpbook-vars.inc`?

**Answer:** This is where some of the account information is kept. Its content could be like this:

```
<?php
//Critical data to make the connection
$hostname='localhost';
$user='root';
$password='Colt45';
?>
```

Let's check out the whole thing.

## 5.5 Work with multiple tables

We just did some work with only one table. How about two? Let' assume we want to get a list grouping all the cities for each country. Since in the city table, we don't have the name of that country, only a code. Thus, we have to list all the countries, and, during the process, for each country, we have to look for all the cities stored for that country, using its id. Technically, it is just a join.

Depending on if we have already got to that part, you might or might not understand the following SQL code.

```
Select continent, countryname, cityname
From country, city
Where city.countryID=country.id
Order By continent, countryname, cityname;
```

This one works, but contains too much duplicated information.

continent	countryname	cityname
Africa	Kenya	Meru
Africa	Kenya	Mombasa
Africa	Kenya	Nairobi
North America	Canada	Montreal
North America	Canada	Van Couva
North America	Canada	Winnipig
North America	USA	Boston
North America	USA	Chicago
North America	USA	Houston
North America	USA	New York
South America	Brazil	Belo Horizonte

```
| South America | Brazil      | Rio de Janeiro |
| South America | Brazil      | Salvador       |
| South America | Brazil      | San Paulo      |
+-----+-----+-----+
```

14 rows in set (0.05 sec)

Thus, we can't just print out the data we get from the database, but we have to do some preprocessing to get rid of the stuff we do not want, which is what the following `multipleQuery.php` does.

```
<?php
//This is for the example as given in pp. 303 of the Bible
include("phpbook-vars.inc");

//1. Open data base connection
$global_dbh=mysql_connect($hostname, $user, $password)
                or die("Could not connect to database");

//2. Select the database to work with

$db="geography";

mysql_select_db($db, $global_dbh) or die("Could not select database");

//Define a function that will take care of the redundancy
function display_cities($db_connection){
    //3. Pass in the query to displays table of cities and countries.
    $country_query = "Select id, continent, countryname from country
                    order by continent, countryname";
    //4. Get the result
    $country_result=mysql_query($country_query, $db_connection)
                    or die("display_cities:". mysql_error());

    //5. Begin table, print hard-coded table header
    print("<Table border=1>\n");
    print("<Tr><Th>Continent</Th><Th>Country</Th> <Th>Cities</Th></Tr>");

    //6. Loop through countries via a cursor
    while($country_row=mysql_fetch_row($country_result)){
        //Set up country information
        $country_id=$country_row[0];
        $continent=$country_row[1];
        $country_name=$country_row[2];
```

```

print("<Tr Align=left VAlign=top>");

//7. Fill the first two cells of a row
print("<Td>$continent</Td>");
print("<Td>$country_name</Td>");

//Use another query to get data and fill the last cell
print("<Td>");
//3(b): get all the cities for a country
$city_query="Select cityname from city
             where countryID=$country_id
             order by cityname";
//4(b): execute this query
$city_result=mysql_query($city_query, $db_connection)
              or die(mysql_error());

//5(b) Loop through cities via another cursor
//and print out cities one by one
while($city_row=mysql_fetch_row($city_result)){
    $city_name=$city_row[0];
    //Every city name is on a separate line
    print("$city_name<BR>");
} //end of the city_row loop
//close cities cell, thus country row
print("</Td></Tr>");
} //end of the country_row loop

//8. Finish off the table
print("</Table>");
} //end of the function
?>

<HTML>
  <Head>
    <Title>Cities by Country</Title>
  </Head>
  <Body>
    <?php
      display_cities($global_dbh);
      //9. Close off this connection
      mysql_close($global_dbh);

```

```

?>
>
</Body>
</HTML>

```

Its output should look like the following:

Continent	country	Cities
Africa	Kenya	Meru Mombasa Nairobi
North America	Canada	Montreal Van Couva Winnipeg
North America	USA	Boston Chicago Houston New York
South America	Brazil	Belo Horizonte Rio de Janeiro Salvador San Paulo

:

1. Test out both programs.
2. Write an application that only print out part of the given table with a restricting condition passed into a function.

## 5.6 A comprehensive example: rate your boss

In the mailinglist case, we only need to select one of two states, submitted or not. Thus, only one submit button suffices. What happens if we have to select one out of many choices, one out of many cars, one of many power drills, one of many rates for your boss, ...? In this rather complicated example, we will see how to use radio buttons to collect our answers to such questions, and how to pass on to the processing part within the same script, i.e., following a *self submission* approach, which we saw earlier with the `calcRetirement.php`, where a form and its handles are combined in one script. This practice may cut down the clicks, thus making multiple submission a lot easier: you just stay in one form, but not go through with the handlers multiple times.

Another point is that we don't need to always have a separate message handler. We can do a self-submission, using the following option:

```
<Form method=post Action=
    "<?php echo $_SERVER['PHP_SELF']?>"
```

Let's assume you want to give your boss a rate, which is entered into a database table `ratings` for future reference. Below is its *MySQL* structure:

```
mysql> desc ratings;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID    | int(11)       |      | PRI | NULL    | auto_increment |
| Rating | int(6)        | YES  |     | NULL    |                |
| Boss  | varchar(20)   | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

To prevent somebody from rating again, the program will also save your information into another table `raters`, which looks like the following:

```
mysql> desc raters;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID    | int(11)       |      | PRI | NULL    | auto_increment |
| Name  | varchar(20)   | YES  |     | NULL    |                |
| Email | varchar(30)   | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

The program `rate_boss.php` then makes use of these two tables to collect the rating for the bosses.

```
<!--Below comes from Figure 17.4 of the Bible-->
<?php

    $reg_form= <<< EOREGFORM
<p>We must ask for your name and email address to ensure that no one votes more
    than once, but we do not associate your personal information with your rating. </p>

<Form Method="post" Action="$thisfile">
    Name: <Input Type="text" Size=25 Name="name"><BR><BR>
    Email: <Input Type="text" Size=25 Name="email">
    //The following box will not show, but when clicked
    //once, its value is sent along with the input
    //and will be used to prevent repeated checking
```

```

    <input Type="hidden" Name="stage" Value="register"><br><br>
    <input Type="submit" Name="submit" Value="Submit">
</Form>
EOREGFORM;
//The last line must begin from the very first column. Otherwise, error will occur.
//See page 140 of the Bible

//construct the rate form as a string
$rate_form=<<<EORATEFORM
<p>My boss is:</p>

//The following line indicates that this form will
//call itself to process the message
<Form Method="post" Action="$_SERVER['PHP_SELF']">
    <input Type="radio" Name="rating" Value=1><br>
    Driving me to look for a new job. <br><br>

    <input Type="radio" Name="rating" Value=2><br>
    Not the worst, but pretty bad. <br><br>

    <input Type="radio" Name="rating" Value=3><br>
    Just so-so. <br><br>

    <input Type="radio" Name="rating" Value=4><br>
    Pretty good. <br><br>

    <input Type="radio" Name="rating" Value=5><br>
    A pleasure to work with. <br><br>

    Boss name: <input Type="text" Size=25 Name="boss"><br>
    <input Type="hidden" Name="stage" Value="rate"><br><br>
    <input Type="submit" Name="submit" Value="Submit">
</Form>
EORATEFORM;
//The last line must begin from the very first column. Otherwise, error
//will occur. See page 140 of the bible

if (!$_POST['submit']) {
    //first step, just show the registration form
    $message=$reg_form;
}
elseif ($_POST['submit']=='Submit' && $_POST['stage']=='register'){
    //second step, show the registration form again on error,
    //rating form on successful insert

```

```

if(!$_POST['name']||$_POST['name']==""|| strlen($_POST['name'])>30
    || !$_POST['email'] || $_POST['email']=="" || strlen($_POST['email'])>30){
    $message='<p>There is a problem. Did you enter a name and email address?</p>';
    $message.=$reg_form;
}
else { //Put in your log in information
    mysql_connect("localhost", "zshen", "PASSWORD")
        or die("Failure to communicate with database during the registration
            stage<BR>");

    //Select the test database
    mysql_select_db("test") or die("Can't select the database");

    //Check to see this name and email have not appeared before
    $as_name = addslashes($_POST['name']);
    $tr_name = trim($as_name);
    $as_email = addslashes($_POST['email']);
    $tr_email=trim($as_email);

    //A query to search for the just entered name and email address.
    $query = "select ID from raters where Name='$tr_name' and Email='$tr_email' ";
    $result=mysql_query($query);

    //If there exists at least one such record
    if(mysql_num_rows($result)>0){
        error_log(mysql_error());
        $message='Some one with this name and password has already rated.
            If you think a mistake was made, please email help@example.com.';
    }
    else {
        //She is new, insert her name and email address so that she can't rate
        //again in the future
        $query = "Insert into raters (Name, Email)
            Values
            ('$tr_name', '$tr_email')";

        //Execute the query
        $result=mysql_query($query);

        //Check to see if it is done properly. See notes on pp. 281 on the
        //combination of mysql_affected and insert.
        //She is registered, now give her the chance to rate.
        if(mysql_affected_rows()==1)
            $message=$rate_form;
        else {

```

```

        error_log(mysql_error());
        echo mysql_error();
        $message='<p>Some thing went wrong with your sign up attempt.</p>';
        $message.=$reg_form;
    }//else of the database insert query
} //else of the num_row checking
} //end of inproper value checking
} //end of elseif stage is register
elseif ($_POST['submit']=='Submit' && $_POST['stage']=='rate'){
    //Third step, store the rating and boss' name

    //open connection to the database
    mysql_connect("localhost", "zshen", "PASSWORD")
        or die("Failure to connect with database during the rate stage");

    mysql_select_db("zshen");

    //Insert rating and boss' name
    $as_boss=addslashes($_POST['boss']);
    $tr_boss=trim($as_boss);
    $rating = $_POST['rating'];

    //Notice that the type of $rating is number
    $query="Insert into ratings (Rating, Boss)
        Values
        ('$rating', '$tr_boss')";

    $result=mysql_query($query);

    //Check to see if it is done properly. See notes on pp. 281 on the
    // combination of mysql_affected and insert.
    if(mysql_affected_rows()==1)
        $message="<p>Your rating has been successfully submitted.</p>";
    else {
        error_log(mysql_error());
        $message='<p>Some thing went wrong with your rating attempt.
            Try again. </p>';
        $message.=$rate_form;
    } //end of rating insertion
} //endif stage is rate
?>

<HTML>
<Head>
    <Style Type="text/css">

```

```

        <!--
        Body, P, TD {color: black; font-family: verdana; font-size: 10 pt}
        H1 {color: black; font-family: arial; font-size: 12 pt}
        -->
    </Style>
</Head>

<Body>
<!--A table with only one row, consisting of two cells, the first being
    the left edge, 1/6; and the other contains the form, 5/6 -->
<Table Border=0 cellPadding=10 Width=100%>
    <Tr>
        <!--The following cell gives the left cushion edge-->
        <Td BgColor="F0F8FF" Align=Center VAlign=top Width=17%> </Td>

        <!--The following gives the right entry form part, completely white-->
        <Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>

            <H1>Rate your boss anonymously</H1>
            <!-- Just show the message that has been prepared. -->
            <?php echo $message; ?>
        </Td>
    </Tr>
</Table>
</Body>
</HTML>

```

### Labwork 5.6:

1. Create the `ratings` and `raters` database tables.
2. Copy in, then revise, the scripts to your own directory.
3. Run the above scripts repeatedly until either you are bored or completely understand it. You certainly should check if the tables are indeed updated accordingly.

## 5.7 More data types

Besides the simple data types as we have seen so far, both *PHP* and *MySQL* also support such familiar data types as

- *textarea*: As an example, the script `comment_edit.php` uses a table `comments`, with the following structure:

```
mysql> desc comments;
```

```

+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID             | int(11)       |      | PRI | NULL    | auto_increment |
| comment_header | varchar(30)   | YES  |     | NULL    |                |
| comment        | varchar(200)  | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.05 sec)

```

- *check boxes*: As an example, `mult_chkbox.php` uses the `date` table with the following structure:

```

mysql> desc date;
+-----+-----+-----+-----+-----+-----+
| Field      | Type   | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| subscribe | int(11) |      | PRI | NULL    | auto_increment |
| tall       | int(11) | YES  |     | NULL    |                |
| dark       | int(11) | YES  |     | NULL    |                |
| handsome   | int(11) | YES  |     | NULL    |                |
| witty      | int(11) | YES  |     | NULL    |                |
| programmer | int(11) | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.05 sec)

```

- *radio buttons*: As an example, `date_prefs.php` uses the `qualities` table, defined as follows:

```

mysql> desc qualities;
+-----+-----+-----+-----+-----+-----+
| Field      | Type   | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| subscriber | int(11) |      | PRI | NULL    | auto_increment |
| height     | int(11) | YES  |     | NULL    |                |
| haircolor  | int(11) | YES  |     | NULL    |                |
| edu        | int(11) | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

```

All the aforementioned scripts can be found in the lab page.

**Labwork:** Set up the tables, populate them, then play with those files. Notice that you have to populate the `comments` table *before* running the script.

## 6 Debugging

In my personal *PHP* programming experience, the most time consuming and silly errors were made when I misspelled a name, such as table name, column name, etc.. Another thing is to put comments where there should not be, particularly with a long definition of a form. Other things could be

- Mix up commenting usage for *HTML* (`<!--comments -->`) and *PHP* (`//`).  
If you use `//` in an *HTML* document, it just a bit annoying. On the other hand, if you use `<!-- -->` in a *PHP* environment, it will lead to an error message.

- Put a comma in a column list within a single quote, such as

```
$query="Update book Set title='$title,'";
```

- Value unquoted: For example

```
$query="Select * from Book Where author=Daniel";
```

Thus, it is often a good idea to break the line into two parts: a query and its invocation.

- Unbounded variable: Given the following segment:

```
$customerID=find_customer_id();  
$result_id=mysql_fetch_query(  
    "Select * From customers  
    Where ID=customer_ID");  
$row=mysql_fetch_row($result_id);
```

there will be a crash.

Here, because of a typo, the variable `$customer_ID` is unbounded, thus, the query becomes

```
"Select * From customers Where ID="
```

- Garbage in, garbage out: Make sure that you have constructed the right queries first. For example,

```
"Select * From families Where kidcount=1  
and kidcount=2"
```

- Understand the functions: For example, `mysql_affected_rows()` only works for Insert, Update and Delete; while `mysql_num_rows()` only works for Select.

Moreover, the former takes an optional query id, while the latter always requires one. Here is an example of this nature.

```
$link_id=mysql_connect($host,$user,$pwd);
mysql_select_db($database,$link_id);

$query="Insert into mytable
        Values (null,'$myVal')";
$result=mysql_query($query);
$test_insert=mysql_affected_rows();

$query1="Select * from mytable";
$result1=mysql_query($query1);
$test_select=mysql_num_rows($result1);

$query2="Delete from mytable";
$result2=mysql_query($query2);
//The following fails
$test_delete=mysql_num_rows($result2);
$test_delete2=mysql_affected_rows();
//The following brings back the same result
$test_select2=mysql_num_rows($result1);
```

## 7 Appendix: An introduction to HTML

HTML (HyperText Markup Language) is used to describe how text, images, and multimedia are displayed by Web browsers. More specifically, it uses various *tags* to describe the layout of a document; the browser then uses these tags to figure out how to display the document.

### 7.1 The basics

Tags are enclosed in angle brackets. For example, `<title>` is a tag that indicates that this section contains the title of the document. Many tags, including `<title>`, have corresponding end tags that indicate where the section ends. The end tags look just like the start tags except that they start with the character `/`, e.g., `</title>`. So the following text indicates that the title of the document is “Introduction to HTML”:

```
<title>Introduction to HTML</title>
```

There are a few tags that almost every document will contain: `<html>`, `<head>`, `<title>`, and `<body>`. Below is an example of a simple HTML document, call it `Sample.html`:

```

<HTML>
  <HEAD>
    <TITLE>Introduction to HTML</TITLE>
  </HEAD>

  <BODY>
    In this lab you will learn about HTML, which is lots of fun
    to use. In particular, you will learn how to use fonts,
    paragraphs, lists, links and applets in a web page. Now you
    can make your own web page for your friends to visit!
  </BODY>
</HTML>

```

To see what this looks like, type it in, and save it in the `Home` folder of your `M:` drive, and then open the file in the web browser. Change the size of the browser window (click and drag any corner) and see how the text is reformatted as the window changes. Note that the title appears on the window, not as part of the document.

The `HEAD` of a document (everything between `<HEAD>` and `</HEAD>`) contains the introduction to the document. The title goes in the head, but for now we won't use the head for anything else. The `BODY` of a document (everything between `<BODY>` and `</BODY>`) contains everything that will be displayed as part of the document. Both the `HEAD` and the `BODY` are enclosed by the `HTML` tags, which begin and end the document.

## 7.2 Add in a picture

Assume that your name is Jane Doe, then your user name should be `jdoe`. Let's assume that you have already saved a picture with the following name<sup>2</sup>, `car.jpg`, in `Home`, the folder where your html file is kept; to add in a picture, you should include the following line into your html file.

```

```

On the other hand, if your name is O. J. Simpson, then your user name should be `ojsimpson`. Assume that you also saved the picture, `car.jpg`, in `Home`, then, to include that picture in your home page, you should include the following line into the file, e.g., `Sample.html`.

```

```

Moreover, if Jane Doe saved the file, `car.jpg`, in a folder `Pics`, that she created inside the `Home` folder, then the line should be the following:

```

```

---

<sup>2</sup>By the name `car.jpg`, we mean that you give it a name `car`, while its type is `jpg`.

## 7.3 Make it look better

This document contains only plain text, but an HTML document can have much more structure: headings, paragraphs, lists, bold and italic text, images, links, tables, and so on. Here is a document containing a heading, two paragraphs, and some fancy fonts:

```
<HTML>
  <HEAD>
    <TITLE>Introduction to HTML</TITLE>
  </HEAD>

  <BODY BGCOLOR="lightgreen">
    <H1 align="center">Introduction to HTML</H1>

    <P>In this lab you will learn about <I>HTML</I>, which
    is lots of fun
    to use. In particular, you will learn how to use fonts,
    paragraphs, lists, links, and colors in a web page. Now you
    can make your <B>own</B> web page for your friends to visit!</P>

    <U>Yippee!</U>
  </BODY>
</HTML>
```

Run the HTML document to see what this looks like in the browser.

In this document the `<H1>` tag creates a level 1 heading. This is the biggest heading; it might be used at the beginning of the document or the start of a new chapter. Level 2 through level 6 headings are also available with the `<H2>` through `<H6>` tags.

The `<P>` tag creates a new paragraph. Most browsers leave a blank line between paragraphs. The `<B>` tag creates bold text, the `<I>` tag creates italic text, and the `<U>` tag creates underlined text. Note that each of these tags is closed with the corresponding end tag. The `BGCOLOR` attribute on the `BODY` tag sets the background color.

Note that line breaks and blank lines in the HTML document do not matter—the browser will format paragraphs to fit the window. If it weren't for the `<P>` tag, the blank line between the paragraphs in this document would not show up in the displayed document.

**Exercise 1:** Write a simple web page about things that interest you. Your page should contain at least the following:

- A title (using the `<TITLE>` tag)
- Two different levels of headings
- Two paragraphs

- Some bold, italic, or underlined text

Your name should appear somewhere in the document.

When you are done, save it in the **home** folder in the **M:** drive with the name of, e.g., `myPage.html`. You can then view your document from a browser, e.g., Microsoft IE, by clicking open your file.

## 7.4 The list structures

We often want to add a list to a document. HTML provides two kinds of lists, ordered (e.g., 1, 2, 3) and unordered (e.g., bulleted).

A list is introduced with the `<OL>` or `<UL>` tag, depending on whether it is ordered or unordered. Each list item is introduced with a `<LI>` tag and ended with the `</LI>` tag. The entire list is then ended with `</OL>` or `</UL>`, as appropriate.

For example, if you want to add in the following list

1. Get up
2. Take shower
3. Have breakfast
4. Go to classes

you would use the following code

```
<ol>
<li> Get up
<li> Take shower
<li> Have breakfast
<li> Go to classes
</ol>
```

If you don't want to have the items enumerated, replace the `<ol>`, `</ol>` pair with the `<ul>`, `</ul>` pair.

**Exercise 2:** Add a list, either ordered or unordered, of at least three elements to your document.

## 7.5 Add in a table

Lists are one-dimensional structures, while tables are two dimensional. Assume that you want to have the following table in your web page:

Saturday	Sunday
Laundry	Study

you can include the following code in your web page

```
<table border="2">
<tr>
<td>Saturday</td>
<td>Sunday</td>
</tr>
<tr>
<td>Laundry</td>
<td>Study</td>
</tr>
</table>
```

Sometimes, you can also play a little trick with, e.g., the following segment.

```
<table border="5">
<td rowspan=2>Saturday</td>
<td>Laundry</td>
</tr>
<tr>
<td>Study</td>
</tr>
</table>
```

**Exercise 3:** Find out what does the last segment do, then add a meaningful table to your document.

## 7.6 Hyperlinks

We use links to connect one document to another. Links are created in HTML with the `<A>` (anchor) tag. When creating a link you have to specify two things:

- The URL of the document to go to when the link is clicked. This is given as the `HREF` attribute of the `A` element.
- How the link should be displayed (that is, what text or image to click on to go to the linked document). This appears between the `<A>` and `</A>` tags.

For example, the code below creates the link shown, which goes to a page about the history of computing:

```
Learn more about <A HREF="http://ei.cs.vt.edu/~history"> the history of computing.</A>
```

Learn more about the history of computing.

**Exercise 4:** Add at least two links that ties in to the material on your page. Have another look at your web page, then save this file with a new name, `home.html`, in your Home folder in the M: drive.

You can add in any other stuff into your page as you see fit. Once you are happy with it, send the address of your web page to me via email, e.g., `oz.plymouth.edu/~jdoe/home.html`, if your username is `jdoe`.

## 7.7 Sky is the limit

There are lots of other stuff you can add into a web page. You can certainly learn more about them in some of the related courses we offer within the Dept. of Computer Science and Technology. On the other hand, if you want to further explore more about the html related stuff on you own, you can begin with, e.g., the following basic introduction to html related material via the following ink: <http://www.cwru.edu/help/introHTML/toc.html>

## References

- [1] Converse, T, Park, J., and Murgan, C., *PhP5 and MySQL Bible*, Wiley Publishing, Inc., Indianapolis, IN, 2004.
- [2] Chou, H., *Hugh's Mortgage and Financial Calculators*, available from <http://www.hughchou.org/calc/formula.html>
- [3] *MySQL 5.1 Reference Manual*, available from <http://dev.mysql.com/doc/refman/5.1/en/index.html>.
- [4] [www.php.org](http://www.php.org).
- [5] *PHP/MySQL Tutorial*, available from <http://dev.mysql.com/usingmysql/php/>.
- [6] *HTML 4.01 Specification*, available from <http://www.w3.org/TR/html4/interact/forms.html>.