Lecture 5: Introduction of Programming Language PHP

(02-24,03-03-2010)

(Reading: Lecture Notes)

Lecture Outline

1. Introduction
2. Basics
3. Control constructs
4. Functions
5. Arrays and associative arrays
6. Objects
7. Examples

1. Introduction

(1) What is PHP?

a. Originally PHP was designed as a language that helps programmers/web practioners maintain persoal home pages (PHP).

(a) It was first developed by *Rasmus Lerdorf* in 1994 as a set of tools that were used as a parsing engine to interpret a set of macros.

(b) The first PHP was combined with a form interpretation (FI) package and released as PHP/FI as an open source to the public.

(c) Many professional began their contribution to PHP/FI. By 1997, PHP had been used in over 50,000 web sites. By then, PHP had been added functionalities like connecting to databases and performing dynamic web information processing.

b. In June 1998, led by two developers (Zeev Suraski and Andi Gutmans) PHP3 was released. PHP3 was the first release that included supports of multiple platforms (no longer just restricted to Linux), databases, SNMP, and IMAP (Internet Message Access Protocol).

c. Over the years, PHP has grown into a full-featured programming language capable of managing huge database-driven online environment. Its official name is now *HyperText Preprocessor*.

d. PHP4: released in May 2000. It includes multiple important enhancements and improvements over PHP3:

http://us3.php.net/manual/en/faq.general.php#faq.general.differences-34

e. Current stable version (Sept. 2003, PHP4.3.33). Next 'major version: PHP5. PHP5 is now in beta1.

(2) What PHP does?

a. Web programming language:

(a) CGI programming

(b) Integrated web based information processing

(c) Database access/manipulation

(d) Web application authentication/authorization

(e) Dynamic documents (PS,PDF,GIF and etc) generation

(f) Cookies management

b. System application/management

(a) File system access/management
(b) Data encryption/decryption
(c) Form/report generation

(3) Main advantages of PHP

a. Open source, widely available
   (a) Free to use
   (b) Increasingly large user/developer base
   (c) Increasingly more features

b. Facilitation of web applications. PHP allows quick deployment of web applications. PHP allows web practitioners to separate HTML code from scripted elements (processing elements). This decreases the development cycle of web applications.

c. Performance
   (a) PHP is optimized by the so called *Zend engine* to perform well under multiple platforms and database systems. For small applications PHP compared favovorably over other scripting language such as Perl, Java Servlets, and ASP.
   (b) For large web applications, PHP's performance advantage is not so clear. But still compared favorably with other scripting languages.

d. Portability. Available in multiple platforms and database systems.

2. Basics

(1) First PHP script: Hello world

```
1:  <?php
2:       print "Hello world!\n";
3:  ?>
```

(2) PHP start and end tags: Table 1

(3) PHP in HTML documents

```
1:  <html>
2:  <head>
3:       <title>Listing a PHP script including HTML </title>
```

Tag Style	*Start Tag*	*End Tag*
Standard tags	<?php	? >
Script tags	<SCRIPT LANGUAGE="php">	</SCRIPT>
Short tags	<?	? >
ASP tags	<%	%>

Table 1: PHP Start and End Tags

```
 4:  </head>
 5:  <body>
 6:  <b>
 7:  <?php
 8:       print "Hello world!\n";
 9:  ?>
10:  </b>
11:  </body>
12:  </html>
```

(4) Comments in PHP

   a. Single line comments: begin with two forward slashes (//) or a single hash sign (#).

   b. Multiple line comments: begin with /∗ and end with ∗/

(5) Variables: PHP variables begin with a dollar sign $, followed by any alphanumeric characters or the underscore character _:

```
$a
$a12
$331a
$a_b_c
```

(6) Data types: PHP is a *loosely typed* language: the data type of a variable is determined when a value is assigned to the variable.

   a. PHP data types: Table 2

   b. The *settype*, *gettype* functions and type casting.

      (a) The *settype* function takes two arguments: first is a variable, second is a type name. It set the type of that variable to the one specified by the second argument.

Type	Example	*Description*
Integer	5	A whole number
Double	3.14	A floating-point number
String	"hello, world!"	A sequence of characters
Boolean	true, false	Binary values
Object		Class and OO supported type
Array		A sequence of values of the same type
NULL		An uninitialized, unknown value

Table 2: PHP Data Types

(b) Explict type casting of the form:

(integer) $abc

can be used to cast the type of variable or expression value.

(7) Operators

a. Assignment operator: =
It is the same as in C/C++

b. Arithmetic operators:

$$+, -, *, /, \ \%, ++, \text{and} \ --$$

They again have the same semantics as in C/C++. The last two are incremental and decremental operators.

c. String concatenation operator: .

d. Combined assignment operators:

$$+=, -=, /=, *=, \%=, \ \text{and} \ .=$$

e. Comparison operators:

$$==, !=, >, >=, <, <=, ===$$

The last one is the *identical* operator, and will return true if its left operand is equivalent to its right operand and the two operands are of the same type.

f. Logical operators:

(a) ||, or: logical or

(b) &&, and: logical and

(c) xor: exclusive or

(d) !: logical negation

g. Operator precedence: conventional:

(a) $++, --$, (cast), !
(b) /, $*$, %
(c) +, -
(d) $<, <=, =>, >$
(e) $==, ===, != $
(f) &&
(g) ||
(h) $=, += , -=, /=, \%=, .=$
(i) and
(j) xor
(k) or

Notice that although && and *and* are the same operators they have difference precedences. Same to || and *or*.

(8) PHP constants: they can be defined using the built-in function *define*:

define("constant_name", constant_value)

3. Constrol constructs

(1) Conditional branching mechanism

a. The *if* statement and the *elseif* clause:

(a) The *if* statement follows closely the syntax and semantics in C/C++.

```
if ($i == 1) {
$this_entry_num = $forms->vars['Total_entry_1'];
} else if ($i == 2) {
        $this_entry_num = $forms->vars['Total_entry_2'];
}
```

(b) The *elseif* clause is simply a short hand for *else if*.

b. The *switch* statement: again it is very close to the one in C/C++.

```
switch ($code) {
       case '01': $this->process_case_1();
                  break;
       case '02': prepare_case_2();
                  $this->process_case_1();
```

```
                    break;
        default:    print "Sorry, wrong code!\n";
                    break;
}
```

c. The ? operator:

(exp) ? returned_if_exp_true: returned_if_exp_false

```
$capability == 0 ? $lang['Not_capability'] : $lang['Has_capability'];
```

(2) Loops

a. The *while* statement:

```
while ($result = $DB->fetch_row() )
{
     if ($result['member_id'] == 1)
     {
            $active['member_id']++;
     }
}
```

b. The *do ... while* statement:

```
$result = $DB->fetch_row();
do
{
     if ($result['member_id'] == 1)
     {
            $active['member_id']++;
     }
} while ( !empty($result = $DB->fetch_row() ) );
```

Here the function *DB− >fetch_row()* is a user-defined function (using PHP's *mysql_fetch_array*()).

c. The *for* statement:

```
for ($i = 1; $i <= $total_service; $i++)
       ......
{
```

d. The *foreach* statement. Similar to Perl's

```
foreach ($array as $temp) {
   ...
}
```

Example:

```
$users = array ("Bert', "Sharon", "Betty", "Harry");
foreach ($users as $val) {
    print "$val<br>";
}
```

e. The *continue* and *break* statements

4. Functions

(1) Function definition: like in C/C++

```
function parse_member($member) {
         global $forms, $std, $DB;
         ......
}
```

a. Return values from a function:

return expr;

b. Dynamic function calls

```
 1:  <html>
 2:  <head>
 3:      <title>Dynamic function calls </title>
 4:  </head>
 5:  <body>
 6:  <?php
 7:  function SayHello() {
 8:       print "Hello world!\n";
 9:  }
10:  $function_holder = "SayHello";
11:  $function_holder();
12:  ?>
13:  </body>
14:  </html>
```

c. Scope of variables.

(a) Variables used in a function are usually *local*.

(b) Variables can be declared *global*. They can be referenced anywhere in the entire program.

d. Setting default values for arguments

```
function parse_member($member="") {
         global $forms, $std, $DB;
         ......
}
```

In this case the argument $member takes empty string "" as the default value.

(2) Dynamic number of function arguments

a. (Starting in PHP4) Two functions *func_num_args* and *func_get_arg* are supported. They allow a caller to pass any number of functions.

b. Example: adding a sequence of numbers.

(a) First version: adding two numbers:

```
function addNums ( $num1, $num2 ) {
    $result = $num1 + $num2;
    $ret = "<table border=\"1\">";
    $ret .= "<tr><td>number 1: </td><td>$num1 </td></tr>";
    $ret .= "<tr><td>number 2: </td><td>$num2 </td></tr>";
    $ret .= "<tr><td>result: </td><td>$result </td></tr>";
    $ret .= "</table>";
    return $ret;
}
```

This function can only add two integers. Not flexible.

(b) Second version:

```
function addNums ( ) {
    $ret = "<table border=\"1\">";
    for ( $x=0; $x < func_num_args(); $x++) {
        $arg = func_get_arg($x);
        $result += $arg;
        $ret .= "<tr><td>number " . ($x+1) .
                                ":</td><td>$arg</td></tr>";
    }
    $ret .= "<tr><td>result: </td><td>$result </td></tr>";
    $ret .= "</table>";
    return $ret;
}
```

With this new version, we can call the function *addNums* with any number of numbers: addNums(100,40,70,30).

5. Arrays and associative arrays

(1) PHP supports arrays (single and multiple dimension), just like in C/C++

a. Arrays can be defined using the *array()* construct:

```
$actions = array( 'MOVE_TOPIC', 'CLOSE_TOPIC', 'OPEN_TOPIC',
            'DELETE_TOPIC', 'EDIT_TOPIC', 'UNSUBBIT',
            'MERGE_TOPIC', 'SPLIT_TOPIC' );
```

This creates an array variable *$actions* with initial values of eight elements.

(a) Like in C, arrays created like this have integer subscripts that start with subscript 0.

b. Arrays can also be created using individual assignment statements:

```
$actions[] = 'MOVE_TOPIC';
$actions[] = 'CLOSE_TOPIC';
$actions[] = 'OPEN_TOPIC';
 ... ...
```

c. Arrays created above have consecutive subscripts, starting from 0. But subscripts do not have to be consecutive. For example, the following assignment statement would add one more element to the array $action at position 100:

```
$actions[100] = 'TEST_TOPIC';
```

It left a hole of 92 elements (from 8 to 99). This is legal, although it may cause confusion.

(2) Associative arrays

a. Associative arrays are those arrays that can be referenced by "names", instead of subscripts. In other words, their subscripts are character strings (i.e. names):

```
$persons = array(
            "name" => "bob",
            "occupation" => "college professor",
            "date_of_birth" => "10-09-1950"
            );
```

b. They above definition can be changed to direct creation, similar to normal array case:

```
$persons["name"]  = "bob";
$persons["occupation" = "college professor";
$persons["date_of_birth"] = "10-09-1950";
```

c. Looping through associative arrays

```
 1:  <html>
 2:  <head>
 3:  <title>Listing single dimensional array</title>
 4:  </head>
 5:  <body>
 6:  <?php
 7:  $character = array (
 8:        "name" => "bob",
 9:        "occupation" => "superhero",
10:        "age" => 30,
11:        "special power" => "x-ray vision"
12:   );
13:  foreach ($character as $key => $val ) {
14:       print "$key = $val<br>";
15:  }
16:
17:  ?>
18:  </body>
19:  </html>
```

(3) Multidimensional arrays

a. Example

```
$persons = array(
              array(
                  "name" => "bob",
                  "occupation" => "college professor",
                  "date_of_birth" => "10-09-1950"
                ),
              array(
                  "name" => "mary",
                  "occupation" => "high school teacher",
                  "date_of_birth" => "10-09-1960"
                ),
              array(
                  "name" => "jones",
                  "occupation" => "IT manager",
                  "date_of_birth" => "11-11-1959"
                )
  );
```

b. Loop through multidimensional arrays: nested loops:

```
 1:  <html>
 2:  <head>
 3:  <title>Listing multidimensional associative array</title>
 4:  </head>
 5:  <body>
 6:  <?php
 7:  $personals = array (
 8:          array(
 9:              "name" => "bob",
10:              "occupation" => "college professor",
11:              "date_of_birth" => "10-09-1950"
12:            ),
13:          array(
14:              "name" => "mary",
15:              "occupation" => "high school teacher",
16:              "date_of_birth" => "10-09-1960"
17:            ),
18:          array(
19:              "name" => "jones",
20:              "occupation" => "IT manager",
21:              "date_of_birth" => "11-11-1959"
22:            )
23:     );
24:  foreach ( $personals as $val ) {
25:       foreach ( $val as $key => $final_val ) {
26:            print "$key: $final_val<br>";
27:       }
28:       print "<br>";
29:  }
30:
31:  ?>
32:  </body>
33:  </html>
```

(4) Functions on arrays

a. Function *count*: return the number of elements in the array.

(a) For multi-dimensional arrays, this function only return the size of first dimension.

(b) Example usage: to be given.

b. Function *array_merge*

(a) Accepts two *or more* arrays as arguments and returns a merged array combining all their elements.

(b) Example:

```
<?php
$first_array = array ("red", "black", "white");
$second_array = array (1, 2, 3 );
$third_array = array_merge($first_array, $second_array);
foreach ( $third_array as $val ) {
     print "$val<br>";
}
?>
```

(c) Notes:

- The original arrays are not altered.
- If the array arguments passed have same string index, for two array arguments, the first will be overwritten. For more than two, those of the previous arrays will be overwritten.

c. Function *array_push*

(a) Accepts an array and any number of further arguments and add those further arguments to the array.

(b) Example:

```
<?php
$first_array = array ("red", "black", "white");
$final = array_push($first_array, "green", "blue", "brown");
print "There are $final elements in \$first_array<p><br>";

foreach ( $first_array as $val ) {
     print "$val<br>";
}
?>
```

(c) Notes:

- The original array is altered.
- The function returns the number of elements in the final transformed array.

d. Function *array_shift*

(a) Accepts an array argument and removes the first element of the array as return value.

(b) Example:

```
<?php
```

```
$first_array = array ("red", "black", "white");

while ( count ($first_array) ) {
     $val = array_shift ($first_array);
     print "$val<br>";
     print "There are " . count ($first_array) .
                          " elements in \$first_array <br>";
}
?>
```

(c) Notes:

- The original array is altered.
- The function returns the remove element

e. Function *array_slice*

(a) Extracts a chunk of a given array. Accepts an array argument, a starting position (offset), and an (optional) length argument. If the last argument is missing, the function will return all elements from the starting position.

(b) Example:

```
<?php
$first_array = array ("red","black","white","green","blue","brown");
$second_array = array_slice ($first_array, 2, 3);

foreach ( $second_array as $val ) {
print "$val<br>";
}
?>
```

This would print "white", "green", and "blue".

(c) Notes:

- The original array is not altered.
- A value 0 of the offset argument means starting from first element.

f. Function *sort*

(a) Sort numerically indexed arrays

(b) Example:

```
<?php
$first_array = array ("red","black","white","green","blue","brown");
sort ($first_array);

foreach ( $first_array as $val ) {
print "$val<br>";
```

```
}
?>
```

This would print "black", "blue", "brown", ...

(c) Notes:

· The original array is altered.

g. Function *asort*

(a) Sort an associative array by its values

(b) Example:

```
<?php
$first_array = array ("red"=>10,"black"=>8,"white"=>7,"green"=>6,
             "blue"=>4,"brown"=>15);
asort ($first_array);

foreach ( $first_array as $key = > $val ) {
print "$key =  $val<br>";
}
?>
```

This would print blue = 4, green = 6, white = 7, ...

(c) Notes:

· The original array is altered.

h. Function *ksort*

(a) Sort an associative array by its keys

(b) Example:

```
<?php
$first_array = array ("red"=>10,"black"=>8,"white"=>7,"green"=>6,
                     "blue"=>4, "brown"=>15);
ksort ($first_array);

foreach ( $first_array as $key = > $val ) {
     print "$key = $val<br>";
}
?>
```

This would print black = 8, blue = 4, brown = 15, ...

(c) Notes:

· The original array is altered.

6. Objects

(1) Class definition and object creation

a. PHP classes are created using syntax similar to that in C++

```
class my_class {
     // class attributes here
};
```

b. Objects of a defined class can be created using the PHP *new* statement:

```
$obj1 = new my_class();
$obj2 = new my_class();
print "\$obj1 is a " . gettype($obj1). "<br>\n";
print "\$obj2 is a " . gettype($obj2). "<br>\n";
```

(2) Object properties/attributes

a. A property of an object can be a value, an array, a function (i.e. a method), or even another object.

(a) A value property:

```
Class my_class {
      var $first_prop = "Hi, there!";
};
```

(b) Such a property can be referenced using the $->$ operator:

```
$obj1 = new my_class();
$obj2 = new my_class();
$obj2->first_prop = "I am fine!";

print "$obj1->first_prop"<br>\n";
print "$obj2->first_prop"<br>\n";
```

b. Object method properties

(a) Example

```
 1:  <html>
 2:  <head>
 3:      <title>Object method example 1</title>
 4:  </head>
 5:  <body>
 6:  <?php
 7:  class first_class {
 8:      function SayHello() {
 9:          print "Hello world! My name is harry\n";
10:      }
11:  }
12:
```

```
13:  $obj1 = new first_class();
14:  $obj1->sayHello();
15:  // outputs "Hello, world! My name is harry"
16:  ?>
17:  </body>
18:  </html>
```

(b) Accessing a property from within a method

```
 1:  <html>
 2:  <head>
 3:      <title>Object method example 2</title>
 4:  </head>
 5:  <body>
 6:  <?php
 7:  class first_class {
 8:      var $name = "harry";
 9:      function SayHello() {
10:          print "Hello world! My name is $name\n";
11:      }
12:  }
13:
14:  $obj1 = new first_class();
15:  $obj1->name = "mary";
15:  $obj1->sayHello();
16:  // outputs "Hello, world! My name is mary"
17:  ?>
18:  </body>
19:  </html>
```

(c) Changing the value of a property from within a method

```
 1:  <html>
 2:  <head>
 3:      <title>Object method example 3</title>
 4:  </head>
 5:  <body>
 6:  <?php
 7:  class first_class {
 8:
 8:      var $name = "harry";
 9:
10:      function setNam( $n ) {
11:          $this->name = $n;
```

```
12:      }
13:
14:      function SayHello() {
15:          print "Hello world! My name is $this->name\n";
16:      }
17:  }
18:
19:  $obj1 = new first_class();
20:  $obj1->setNam( "william" );
21:  $obj1->sayHello();
22:  // outputs "Hello, world! My name is william"
23:  ?>
24:  </body>
25:  </html>
```

(d) A class with a constructor

```
 1:  <html>
 2:  <head>
 3:      <title>Object method example 4</title>
 4:  </head>
 5:  <body>
 6:  <?php
 7:  class first_class {
 8:
 8:      var $name = "harry";
 9:
10:      function first_class( $n = "anon" ) {
11:          $this->name = $n;
12:      }
13:
14:      function SayHello() {
15:          print "Hello world! My name is $this->name\n";
16:      }
17:  }
18:
19:  $obj1 = new first_class("bob");
20:  $obj2 = new first_class("harry");
21:  $obj1->sayHello();
22:  // outputs "Hello, world! My name is bob"
23:  $obj2->sayHello();
24:  // outputs "Hello, world! My name is harry"
```

```
25:  ?>
26:  </body>
27:  </html>
```

(3) A complete example

a. A class that can maintain a table of fields, organized in named columns.

b. The HTML and PHP code:

```
 1:  <html>
 2:  <head>
 3:      <title>Object method example 5</title>
 4:  </head>
 5:  <body>
 6:  <?php
 7:  class Table {
 8:     var $table_array = array();
 9:     var $headers = array();
10:     var $cols;
11:     function Table ( $headers ) {
12:         $this->headers = $headers;
13:         $this->cols    = count ($headers);
14:     }
15:
16:     function addRow ( $row ) {
17:         if ( count ($row) != $this->cols )
18:            return false;
19:         array_push ( $this->table_array, $row );
20:         return true;
21:     }
22:
23:     function addRowAssocArray ( $row_assoc ) {
24:         $row = array();
25:         foreach ( $this->headers as $header ) {
26:              if ( ! isset ($row_assoc[$header] ) )
27:                 $row_assoc[$header] = "";
28:              $row[] = $row_assoc[$header];
29:         }
30:         array_push($this->table_array, $row);
31:         return true;
32:     }
33:
```

```
24:     function output ( ) {
35:         print "<pre>";
36:         foreach ( $this->headers as $header )
37:              print "<b>$header</b> ";
38:         print "\n";
39:         foreach ( $this->table_array as $y ) {
40:              foreach ( $y as $xcell )
41:                   print "$xcell ";
42:               print "\n";
43:         }
44:         print "</pre>";
45:     }
46: }
47:
48: $test = new table ( array ("a", "b", "c") );
49: $test->addRow ( array (1, 2, 3 ) );
50: $test->addRow ( array (5, 6, 7 ) );
51: $test->addRowAssocArray ( array (b=>0, a=>6, c=>3 ) );
52: $test->output();
53: ?>
54: </body>
55: </html>
```

7. Examples (to be given later)