Лабораторная № 8

9. Массивы

Массивы и механизм их использования в значительной степени повышают эффективность программ на PHP. Научившись работать с массивами, вы сможете сохранять и обрабатывать сложные структуры данных.

Массив — это набор значений, скрытых под одним именем. Получить доступ к конкретному значению можно по его номеру или текстовой строке.

Как правило, элементы в массиве указываются по номеру, причем нумерация **начинается с нуля**. При этом важно помнить, что номер последнего элемента всегда на 1 меньше числа элементов в массиве.

Указание элементов массива с помощью текстовой строки может быть полезно в том случае, когда вы хотите сохранить одновременно и значение, и имя величины.

В языке PHP существуют развитые средства для работы с массивами и для указания элементов как по номеру, так и по имени.

В простейшем случае массив — это список значений, расположенных в порядке возрастания номера. Значения могут быть присвоены элементам массива двумя способами — непосредственно или с помощью функции `array()`. Мы рассмотрим оба способа.

Определение массива с помощью функции `array()`

Функцией `array()` можно воспользоваться, если нужно присвоить значения сразу нескольким элементам массива. Давайте для примера создадим массив `$colors` и запишем в него 4 строки.

```
$colors = array("red", "green", "blue", "gray");
```

Теперь обратимся к элементу массива по его номеру:

```
print "$colors[2]";
```

В результате будет выведена строка `"blue"`. Номер элемента указывается в квадратных скобках после его имени. Таким способом можно указывать элемент массива как для получения его значения, так и для присваивания ему значения.

Помните о том, что по умолчанию элементы массива нумеруются, начиная с нуля, т.е. начинается массив не с первого элемента, а с нулевого.

Создание элементов массива с помощью идентификатора

Существует возможность создать новый массив или добавить элемент к тому, который уже есть, с помощью идентификатора массива. Для этого нужно указать имя массива и пару пустых квадратных скобок. Давайте еще раз создадим массив `$colors` таким способом.

```
$colors[] "red";

$colors[] "green";

$colors[] "blue";

$colors[] "gray";
```

Обратите внимание на то, что мы не указываем номер элемента в квадратных скобках. PHP автоматически вычисляет его, освобождая вас от необходимости помнить о том, какой следующий элемент свободен.

Мы могли бы указать номера элементов, и результат был бы тот же самый, однако делать это следует с осторожностью. Рассмотрим следующий пример:

```
$colors[0] = "red";

$colors[200] = "green";
```

В массиве есть только 2 элемента, но номер последнего равен 200. Промежуточные элементы не будут инициализированы. Такая ситуация чревата ошибками при попытке обращения к массиву.

После того как массив создан, можно добавлять к нему новые элементы. В следующем фрагменте мы создаем массив с помощью функции `array()` и добавляем к нему новый элемент:

```
$colors = array("red", "green", "blue", "gray");

$colors[] = "white";
```

Ассоциативные массивы

Доступ к элементам массива по номеру удобен тогда, когда вам нужно выбирать их в том порядке, в котором они были созданы, или при сортировке массива. Однако иногда бывает необходимо обратиться к элементу массива по его имени. В ассоциативных массивах элемент указывается не по номеру, а по имени. Представьте себе телефонную книгу. Как удобнее обратиться к полю имени — по его номеру, который вы можете не сразу вспомнить, или назвав это поле непосредственно?

Ассоциативный массив — это массив, к элементу которого можно обратиться по имени.

Как и раньше, ассоциативный массив есть возможность создать непосредственно или с помощью функции `array()`.

Разница между ассоциативными массивами и обычными в языке PHP не является принципиальной. Эти массивы не являются объектами разных типов, как в языке Perl. Однако все же следует обращаться с ними по-разному, потому что они требуют различного подхода и стратегии.

Создание ассоциативного массива с помощью функции `array()`

Для того чтобы создать ассоциативный массив с помощью функции `array()`, нужно задать как имя, так и значение для каждого элемента. В следующем примере создается ассоциативный массив `$sal` из трех элементов.

```
$sal = array (
'name' => "Peel",
'city' => "London",
'comm' => 0.12
);
```

Теперь можно обратиться к любому элементу массива:

```
print $sal['name'];
```

Апострофы у названий ключей при создании ассоциативного массива использовать обязательно.

Непосредственное создание ассоциативного массива

Создать новый массив или добавить к существующему пару имя/значение можно просто присвоив значение элементу массива, указав этот элемент по имени. Например, в следующем фрагменте мы опять создаем массив `$sal`.

```
$sal['name'] = "Peel";
$sal['city'] = "London";
$sal['comm'] = 0.12;
```

Многомерные массивы

До сих пор мы просто говорили, что элементы массива могут иметь некоторые значения. Так, в нашем массиве `$sal` два элемента содержат строки и один элемент — вещественное число. Однако не всегда все так просто. Фактически массив может состоять из значений, объектов или даже из массивов. Многомерный массив как раз состоит из нескольких массивов, т е. каждый его

элемент является в свою очередь массивом. Для того чтобы обратиться ко второму элементу первого массива, нужно написать так:

```
$my_array[1][2]
```

Многомерный массив — это массив, каждый элемент которого является массивом.

Использование многомерных массивов позволяет вам относительно просто создавать довольно сложные структуры данных. Например, в листинге 9.1 мы создаем массив, каждый элемент которого является ассоциативным массивом.

Листинг 9.1. Создание многомерного массива

```
<html> <head>
<title> Листинг 9-1. Создание многомерного массива
</title> </head> <body>
<?php
$sals = array (  #1
    array ('name'=>"Peel",
           'city'=>"London",
           'comm'=>0.12
           ),
    array ('name'=>"Serres",
           'city'=>"San Jose",
           'comm'=>0.12
           ),
    array ('name'=>"Rifkin",
           'city'=>"Barcelona",
           'comm'=>0.15
          )
);  #1
print $sals[0]['city']; // напечатается "London"
?>
</body> </html>
```

Обратите внимание на то, как мы делаем вложенные вызовы функции `array()`. На первом уровне нами создается массив. Затем для каждого из его элементов мы создаем вложенные массивы.

После этого выражение `$sals[2]` дает нам доступ ко 2-му элементу массива верхнего уровня. Продолжая в том же духе, мы можем обратиться к любому элементу ассоциативного массива.

Таким образом, `$sals[2]['name']` вернет нам значение `"Rifkin"`, а `$sals[2]['comm']` вернет 0.15.

Если и на первом уровне нужен ассоциативный массив, то это делается так:

```
$sals = array (  #1
    'first'=>array ('name'=>"Peel",
                    'city'=>"London",
                    'comm'=>0.12
                    ),
    'second'=>array ('name'=>"Serres",
                     'city'=>"San Jose",
                     'comm'=>0.12
                     )
);  #1
```

Тогда `$sals['second']['name']` вернет значение `"Serres"`.

Работа с массивами

Выше мы рассмотрели несколько способов создания массивов и добавления элементов к существующему массиву. В этом разделе будет рассказано, какие средства для получения информации о массивах и для доступа к его элементам предоставляет PHP.

Получение размера массива

К любому элементу массива можно обратиться по его номеру:

```
print $colors[3];
```

Однако механизм работы с массивами настолько гибок, что иногда вы можете не знать, сколько именно элементов содержится в массиве. В таком случае на помощь приходит функция `count()`. Эта функция возвращает количество элементов массива. В следующем примере мы воспользуемся данной функцией для того, чтобы получить доступ к последнему элементу массива.

```
$colors = array("red", "green", "blue", "gray");
print $colors[count($colors)-1];
```

Обратите внимание на то, что для доступа к последнему элементу мы вычитаем 1 из количества элементов. Так приходится делать потому, что номера элементов начинаются с нуля, из-за чего номер последнего элемента не равен их количеству.

Строго говоря, не всегда элементы нумеруются указанным способом, т.е. существует способ изменить эту нумерацию и сделать так, чтобы массив начинался с первого элемента, но делать это не рекомендуется.

Просмотр массива с помощью цикла

Существует много способов просмотреть все элементы массива в цикле. В этой главе мы рассмотрим только наиболее распространенный и мощный оператор `foreach`. Оператор `foreach` появился только в PHP 4.0.

Если мы имеем простой, проиндексированный числами массива, оператор `foreach` используется следующим образом:

```
foreach ($array as $ind)
        {
         // тело цикла
        }
```

В данном случае `$array` — это имя массива, который нужно просмотреть, а `$ind` — переменная, где будет временно храниться значение каждого элемента. Данный способ продемонстрирован в листинге 9.2.

Листинг 9.2. Просмотр массива

```
<html> <head>
<title> Листинг 9-2. Просмотр массива
</title> </head> <body>
<?php
$colors = array("red", "green", "blue", "gray");
foreach ($colors as $ind)
        {print "<p>$ind";
        }
?>
</body> </html>
```

Значение каждого элемента массива временно помещается в переменную `$ind`, а потом выводится на печать. Если вам придется переносить свои программы из языка Perl в PHP, будьте внимательны. Дело в том, что оператор `foreach` работает по-разному в этих двух языках — в Perl любое изменение временной переменной отражается на значении самого элемента массива, а в PHP — нет. Изменение переменной `$ind` в предыдущем примере не привело бы к изменению массива `$colors`.

Просмотр в цикле ассоциативного массива

Для того чтобы просмотреть в цикле ассоциативный массив, нужно написать оператор `foreach` несколько по-другому. В этом случае конструкция должна выглядеть следующим образом:

```
foreach ($array as $key=>$value)
```

Здесь `$array` — это имя массива, `$key` — переменная, в которой сохраняется имя каждого элемента массива, а `$value` — переменная, где временно сохраняется значение каждого элемента.

Вывод многомерного массива

Теперь вы уже можете с помощью рассмотренных способов вывести на печать многомерный массив, который мы создали в листинге 9.1. Пример приведен в листинге 9.3.

Листинг 9.3. Просмотр в цикле многомерного массива

```
<html> <head>
<title> Листинг 9-3. Просмотр многомерного массива
в цикле </title> </head>
<body>
<?php
$sals = array (  #1
             array ('name'=>"Peel",
                    'city'=>"London",
                    'comm'=>0.12
                    ),
             array ('name'=>" Serres",
                    'city'=>"San Jose",
                    'comm'=>0.13
                    ),
             array ('name'=>"Rifkin",
                    'city'=>"Barcelona",
                    'comm'=>0.15
                    )
               );  #1
foreach ($sals as $ind)
   {  #2
    foreach ($ind as $key=>$val)
       {
```

```
        print "$key:  $val<br>";
       }
    print "<br>";
   }  #2
?>
</body> </html>
```

Вывод программы из листинга 9.3 выглядит так:

name: Peel
city: London
comm: 0.12

name: Serres
city: San Jose
comm: 0.13

name: Rifkin
city: Barcelona
comm: 0.15

Мы создаем два цикла `foreach`. Внешний цикл перебирает элементы массива `$sals`, который упорядочен по номерам. Значение каждого элемента помещается в переменную `$ind`. Поскольку каждый элемент, помещенный в данную переменную, сам по себе является массивом, его тоже можно просмотреть в цикле. Это и делает внутренний цикл `foreach`, помещая имя и значение каждого элемента в переменные `$key` и `$val`.

Для того чтобы эта программа работала так, как мы ожидаем, нам необходимо убедиться, что в переменной `$ind` всегда содержится массив. Для повышения надежности мы могли бы воспользоваться функцией `is_array()`, которая возвращает значение `true`, если ее аргумент является массивом, и `false` — в противном случае.

Управление массивами

Вы уже умеете сохранять значения в массиве и получать доступ к его элементам, но в PHP есть функции, позволяющие делать гораздо больше.

Объединение массивов функцией `array_merge()`

Функция `array_merge()` принимает аргументами несколько массивов и возвращает результат их объединения (т.е. сперва идут элементы первого массива, потом второго и т.д.). В листинге 9.4 мы создаем два массива,

присоединяем второй к первому и просматриваем в цикле результат этого объединения.

Листинг 9.4. Объединение массивов

```
<html> <head>
<title> Листинг 9-4. Объединение массивов
</title> </head>
<body>
<?php
$first  = array("a", "b", "c");
$second = array("1", "2", "3");
$third = array_merge($first, $second);

foreach ($third as $val)
   {
    print "$val<br>";
   }
?>
</body> </html>
```

В массиве `$third` содержатся копии всех элементов, входящих в массивы `$first` и `$second`. Цикл `foreach` выводит все эти элементы на печать, разделяя их тегом `<br>`. Помните о том, что исходные массивы никак при этом не изменяются.

Добавление элементов к массиву с помощью функции `array_push()`

Функция `array_push()` принимает аргументом массив и еще несколько параметров, которые все к этому массиву присоединяются. Обратите внимание на то, что данная функция, в отличие от предыдущей, преобразует массив, переданный ей в первом аргументе. Функция `array_push()` возвращает количество элементов в результирующем массиве. Давайте для примера создадим массив и добавим к нему несколько элементов (листинг 9.5).

Листинг 9.5. Добавление элементов к массиву

```
<html> <head>
<title> Листинг 9-5. Добавление элементов к массиву
</title> </head>
<body>
<?php
$first = array("a", "b", "c");
$total = array_push($first, 1,2,3);
print "В массиве \$first всего $total элементов <p>";
foreach ($first as $val)
   {print "$val<br>";
   }
?>
</body> </html>
```

Поскольку функция `array_push()` возвращает количество элементов в результирующем массиве, мы можем запомнить это число в переменной и затем вывести его на экран браузера. В массиве `$first` теперь содержатся те элементы, которые в нем были вначале, и три целых числа, переданных функции `array_push()`. Все эти элементы выводятся на печать с помощью цикла `foreach`.

Обратите внимание на то, что для вывода на печать строки `"$first"` был поставлен управляющий символ обратной косой черты. Если бы мы этого не сделали, то PHP попытался бы вывести не строку, а значение переменной `$first`. Но в данном случае нам нужно было вывести именно знак доллара просто как символ, а не значение переменной, поэтому была использована обратная косая черта перед знаком доллара (см. главу 2).

Будьте внимательны: при передаче функции `array_push()` во втором аргументе массива данный массив будет добавлен как элемент, т.е. будет создан двумерный массив. Для слияния двух массивов следует пользоваться функцией `array_merge()`.

Удаление первого элемента с помощью функции `array_shift()`

Функция `array_shift()` удаляет из переданного ей массива первый элемент и возвращает этот удаленный элемент. В следующем примере (листинг 9.6) мы воспользуемся данной функцией в цикле, причем каждый раз с

помощью функции `count()` будем проверять, остались ли в массиве еще какие-нибудь элементы.

Листинг 9.6. Удаление первого элемента массива

```
<html> <head>
<title> Листинг 9-6. Удаление первого элемента
        массива</title> </head> <body>
<?php
$an_array = array("a", "b", "c");
while (count($an_array))
   {
    $val = array_shift($an_array);
    print "$val<br>";
    print "В массиве \$an_array есть"
          .count($an_array)." элементов<br>";
   }
?>
</body> </html>
```

Результат работы этого фрагмента выглядит так:

a
В массиве $an_array есть 2 элементов

b
В массиве $an_array есть 1 элементов

c
В массиве $an_array есть 0 элементов

Функция `array_shift()` может оказаться полезной, когда вам нужно создать очередь и обслуживать ее до опустошения.

Выделение части массива с помощью функции `array_slice()`

С помощью функции `array_slice()` можно выделить часть массива. Для этого функции передается массив в первом аргументе, начальная позиция, или смещение от начала массива, и необязательный аргумент — длина участка. Если длина опущена, то считается, что нужно выделить часть массива от начальной позиции до его конца. Функция `array_slice()` не изменяет

исходного массива и возвращает новый массив, состоящий из выделенной части.

В приведенном ниже примере (листинг 9.7) мы создаем массив и выделяем из него новый массив, состоящий из трех элементов.

Листинг 9.7. Выделение части массива

```
<html> <head>
<title> Листинг 9-7. Выделение части массива
</title> </head> <body>
<?php
$first = array ("a", "b", "c", "d", "e", "f");
$second = array_slice($first, 2,3);
foreach ($second as $val)
   {print "$val<br>";
   }
?>
</body> </html>
```

В результате будут выведены три элемента, `"c"`, `"d"` и `"e"`, разделенные тегом `<br>`. Обратите внимание на то, что элемент первого массива, номер которого мы указали во втором аргументе функции `array_slice()`, попадает в новый массив.

Если второй аргумент функции `array_slice()` задать отрицательным, то будет выбрано соответствующее количество элементов от конца исходного массива.

Сортировка массивов

В данном разделе будут рассмотрены несколько функций, которые позволяют сортировать как простые, так и ассоциативные массивы.

Сортировка простого массива с помощью функции `sort()`

Функция `sort()` принимает один аргумент — массив — и сортирует его в алфавитном порядке, если хотя бы один из его элементов является строкой, и в числовом порядке, если все его элементы — числа. Эта функция преобразует переданный массив и ничего не возвращает. В листинге 9.8 мы создаем массив,

состоящий из строк длиной в один символ, сортируем его и выводим результат на печать.

Листинг 9.8. Сортировка массива

```
<html> <head>
<title> Листинг 9-8. Сортировка массива
</title> </head> <body>
<?php
$an_array = array("я", "а", "в", "е");
sort($an_array);
foreach ($an_array as $var) {
         print "$var<br>";
}
?>
</body> </html>
```

Не передавайте функции `sort()` ассоциативные массивы — в противном случае вы увидите, что массив отсортирован правильно, но все его имена будут утеряны и вместо них сохранены числа.

Простые массивы можно сортировать в обратном порядке. Для этого существует функция `rsort()`, которая работает точно так же, как и функция `sort()`.

Сортировка ассоциативного массива с помощью функции `asort()`

Функция `asort()` принимает аргументом ассоциативный массив и сортирует его таким же образом, что и функция `sort()`, но при этом не уничтожает имена полей.

Листинг 9.9. Сортировка ассоциативного массива

```
<html> <head>
<title> Листинг 9-9. Сортировка ассоциативного массива
</title> </head> <body>
<?php
$first = array('first'=>5, 'second'=>2, 'third'=>1);
asort($first);
```

```
foreach ($first as $key=>$val)
   {print "$key = $val<br>";}
?>
</body> </html>
```

Для того чтобы отсортировать массив в обратном порядке, нужно воспользоваться функцией `arsort()`, которая работает точно так же.

Сортировка ассоциативного массива по именам полей с помощью функции `ksort()`

Функция `ksort()` принимает аргументом ассоциативный массив и сортирует его по именам полей. Она преобразует сортируемый массив и ничего не возвращает.

Листинг 9.10. Сортировка ассоциативного массива

```
<html> <head>
<title> Листинг 9-10. Сортировка ассоциативного массива
        по именам полей </title>
</head>
<body>
<?php
$first = array('x'=>5, 'a'=>2, 'f'=>1);
ksort($first);
foreach ($first as $key=>$val)
        {print "$key = $val<br>";}
?>
</body>
</html>
```

Для того чтобы отсортировать массив в обратном порядке, нужно воспользоваться функцией `krsort()`, которая работает точно так же.