Выражения и управление процессом выполнения программы в PHP

В предыдущем уроке уже упоминались темы, которые более полно будут здесь рассмотрены, например, выбор (ветвление) и создание сложных выражений. Здесь мне хотелось сконцентрировать внимание на наиболее общих вопросах синтаксиса и работы в PHP, но при этом невозможно было не затронуть темы более высокого уровня. А вот теперь можно преподнести вам основы, необходимые для полноценного использования всех сильных сторон PHP.

В этом уроке будет заложен фундамент практики программирования на PHP и рассмотрены основные способы управления процессом выполнения программы.

Содержание

Выражения

Начнем с базовой части любого языка программирования — выражения. Выражение представляет собой сочетание значений, переменных, операторов и функций, в результате вычисления

которого выдается новое значение. Оно знакомо всем, кто когда-либо имел дело с обыкновенной школьной алгеброй:

$$y = 3({\mid 2x \mid} + 4)$$

что в PHP приобретает следующий вид:

```
$y = 3 * (abs(2*$x) + 4);
```

Возвращаемое значение (в данном случае **y** или $y) может быть числом, строкой или булевым (логическим) значением (названным так в честь Джорджа Буля, английского математика и философа XIX века). Первые два типа значений вам уже должны быть знакомы, поэтому я объясню, что такое третий тип.

TRUE или FALSE?

Элементарное булево значение может быть либо истинным — TRUE, либо ложным — FALSE. Например, выражение 20 > 9 (20 больше 9) является истинным (TRUE), а выражение 5 == 6 (5 равно 6) — ложным (FALSE). (Булевы, или логические, операции могут быть объединены путем использования таких операторов, как И, ИЛИ и исключающее ИЛИ, то есть AND, OR и XOR, которые будут рассмотрены в этой главе.)

> *Обратите внимание, что для имен TRUE и FALSE я использую буквы верхнего регистра. Это обусловлено тем, что в PHP они являются предопределенными константами. При желании можно также применять и их версии, составленные из букв нижнего регистра, поскольку они также являются предопределенными константами. Кстати, версия, в которой задействуются буквы нижнего регистра, более надежна, потому что PHP не допускает ее переопределения, а версия, использующая буквы верхнего регистра, может быть переопределена, и это нужно иметь в виду при импортировании чужого кода.*

В примере показаны некоторые простые выражения: два, о которых уже упоминалось, плюс еще два выражения. Для каждой строки

выводится буква от a до d, за которой следуют двоеточие и результат выражения (тег
 используется в HTML для переноса и разбивает выходную информацию на четыре строки).

> *Теперь, когда HTML5 уже полностью вошел в обиход, и XHTML не планируется на замену HTML, больше уже не нужно использовать самозакрывающуюся форму
 тега
, или любых пустых элементов (не имеющих закрывающих тегов), поскольку теперь символ / необязателен. Поэтому в данной книге мой выбор пал на использование более простого стиля. Если же где-нибудь сделать непустые теги HTML самозакрывающимися (например,), в HTML5 они не сработают, потому что символ / будет проигнорирован, и их нужно будет, к примеру, заменить структурой ... . Но при работе с XHTML нужно по-прежнему пользоваться формой HTML-синтаксиса
.*

```
<?php
        echo "a: [" . (20 > 9) . "]
";
        echo "b: [" . (5 == 6) . "]
";
        echo "c: [" . (1 == 0) . "]
";
        echo "d: [" . (1 == 1) . "]
";
?>
```

Этот код выведет следующую информацию:

```
<?php
        a: [1]
        b: []
        c: []
        d: [1]
?>
```

Обратите внимание, что результаты вычисления обоих выражений, a: и d:, являются истинными (TRUE), имеющими значение 1. А

результаты вычисления выражений b: и c: ложны (FALSE) и вообще не показывают никакого значения, поскольку в PHP константа FALSE определена как NULL (ничто). Чтобы убедиться в этом, можно ввести код, приведенный в следующем примере:

```
<?php
        echo "a: [" . TRUE  . "]
";
        echo "b: [" . FALSE . "]
";
?>
```

Этот код выведет следующую информацию:

```
<?php // test2.php
echo "a: [" . TRUE . "]
";
echo "b: [" . FALSE . "]
";
?>
```

Этот код выведет следующую информацию:

```
a: [1]
b: []
```

Кстати, в некоторых языках константа FALSE может быть определена как 0 или даже как –1, поэтому в каждом языке ее определение стоит проверить.

Литералы и переменные

Простейшей формой выражения является литерал, означающий нечто, вычисляющееся само в себя, например число 73 или строка Hello. Выражение может также быть просто переменной, которая вычисляется в присвоенное этой переменной значение. Обе формы относятся к типам выражений, поскольку они возвращают значение.

В примере показаны три литерала и две переменные, все они возвращают значения, хотя и разных типов.

```
<?php
        $myname = "Brian";
        $myage = 37;
        echo "a: " . 73      . "
"; // Числовой литерал
        echo "b: " . "Hello" . "
"; // Строковый литерал
        echo "c: " . FALSE   . "
"; // Литерал константы
        echo "d: " . $myname . "
"; // Строковая переменная
        echo "e: " . $myage  . "
"; // Числовая переменная
?>
```

Как и ожидалось, в выходной информации вы увидите возвращаемое значение всех этих выражений, за исключением выражения c:, результат вычисления которого является FALSE и ничего не возвращает:

```
a: 73
b: Hello
c:
d: Brian
e: 37
```

Объединив простейшие выражения с операторами, можно создать более сложные выражения, результатом вычисления которых будут какие-нибудь полезные результаты. При объединении присваивания или управляющей конструкции с выражениями получается инструкция. В примере ниже показано по одной инструкции каждого вида.

В первой из них осуществляется присваивание результата выражения 366 - $day_number переменной $days_to_new_year, а во второй выводится приветственное сообщение, если выражение

$days_to_new_year < 30 вычисляется как TRUE.

```
<?php
         $days_to_new_year = 366 - $day_number;         //
Выражение
        if ($days_to_new_year < 30)
        {
                                echo "Скоро Новый год!";
// Инструкция
        }
?>
```

Операторы

В PHP имеется множество мощных операторов, от арифметических, строковых и логических до операторов присваивания, сравнения и многих других операторов
(см. таблицу).

Таблица: Типы операторов PHP

Оператор	Описание	Пример
Арифметический	Элементарная математика	$a + $b
Для работы с массивом	Слияние массивов	$a + $b
Присваивания	Присваивание значений	$a = $b + 23
Поразрядный	Манипуляция битами в байте	12 ^ 9
Сравнения	Сравнение двух значений	$a < $b

Выполнения	
Выполнение содержимого, заключенного в обратные кавычки	
`ls -al`	
Инкремента/декремента	
Добавление или вычитание единицы	
$a++	
Логический	
Выполнение булевых сравнений	
$a and $b	
Строковый	
Объединение строк	
$a . $b	

Различные типы операторов воспринимают разное количество операндов.

- Унарные операторы, такие как оператор инкремента ($a++) или изменения знака числа (-$a), воспринимают только один операнд.
- Бинарные операторы, представленные большим количеством операторов PHP, включая операторы сложения, вычитания, умножения и деления, воспринимают два операнда.
- Один трехкомпонентный оператор, имеющий форму x ? y : z. По сути, это состоящая из трех частей однострочная инструкция if, в которой осуществляется выбор между двумя выражениями, зависящий от результата вычисления третьего выражения.

Приоритетность операторов

Если бы у всех операторов был один и тот же уровень приоритета, то они обрабатывались бы в том порядке, в котором встречались интерпретатору. Фактически многие операторы имеют одинаковый уровень приоритета, что и показано здесь:

```
1 + 2 + 3 - 4 + 5
2 - 4 + 5 + 3 + 1
5 + 2 - 4 + 1 + 3
```

Из примера видно, что, несмотря на перестановку чисел (и предшествующих им операторов), результат каждого выражения имеет значение **7**, поскольку у операторов «**плюс**» и «**минус**» одинаковый уровень приоритета. Можно проделать то же самое с операторами умножения и деления, что показано чуть ниже.

```
1 * 2 * 3 / 4 * 5
2 / 4 * 5 * 3 * 1
5 * 2 / 4 * 1 * 3
```

В этом примере получаемое значение всегда равно **7,5**. Но все меняется, когда в выражении присутствуют операторы с разными уровнями приоритета. Вот три выражения, в которых присутствуют операторы с разными уровнями приоритета:

```
1 + 2 * 3 - 4 * 5
2 - 4 * 5 * 3 + 1
5 + 2 - 4 + 1 * 3
```

Если бы не существовало приоритетности операторов, то в результате вычисления этих выражений получались бы числа **25**, **–29** и **12** соответственно. Но поскольку операторы умножения и деления имеют более высокий уровень приоритета по сравнению с операторами сложения и вычитания, вокруг частей выражения с их участием предполагается наличие скобок, и если их сделать видимыми, выражения будут выглядеть так, как в трёх выражениях, в которых отображены предполагаемые скобки

```
1 + (2 * 3) - (4 * 5)
2 - (4 * 5 * 3) + 1
5 + 2 - 4 + (1 * 3)
```

Очевидно, что PHP должен сначала вычислить подвыражения, заключенные в скобки, чтобы получились частичные вычисленные, показанные в выражениях после вычисления подвыражений в скобках

```
1 + (6) - (20)
2 - (60) + 1
5 + 2 - 4 + (3)
```

Окончательный результат вычисления этих выражений равен соответственно **–13**, **–57** и **6** (что абсолютно отличается от результатов **25**, **–29** и **12**, которые мы увидели бы при отсутствии приоритетности операторов). Разумеется, исходную приоритетность операторов можно отменить, расставив собственные скобки, и принудительно получить результаты, показанные в самом начале, которые были бы получены в отсутствие приоритетности операторов:

```
((1 + 2) * 3 - 4) * 5
(2 - 4) * 5 * 3 + 1
(5 + 2 - 4 + 1) * 3
```

Теперь, если скобки расставлены правильно, мы увидим значения **25**, **–29** и **12** соответственно. В таблице перечислены операторы в порядке их приоритетности от самого высокого до самого низкого уровня.

Таблица: Операторы PHP, расположенные по уровню их приоритетности (сверху вниз)

Оператор(ы)	Тип
()	Скобки
++ —	Инкремент/декремент
!	Логический
* / %	Арифметические
+ -	Арифметические и строковые
<>	Побитовые
< <= > >= <>	

Сравнения	
== != === !==	
Сравнения	
&	
Поразрядный (и ссылочный)	
^	
Поразрядный	
|	
Поразрядный	
&&	
Логический	
||	
Логический	
? :	
Трехкомпонентный	
= += –= *= /= .= %= &= != ^= <>=	
Присваивания	
and	
Логический	
xor	
Логический	
or	
Логический	

Взаимосвязанность операторов

Мы рассматривали обработку выражений слева направо, за исключением тех случаев, в которых вступала в силу приоритетность операторов. Но некоторые операторы могут также потребовать обработки справа налево. Направление обработки обусловливается взаимосвязанностью операторов. Для отдельных операторов взаимосвязанность отсутствует.

Взаимосвязанность приобретает большое значение в тех случаях, когда вы явным образом не меняете приоритетности. Для этого вам нужно знать о действиях операторов по умолчанию. В табице перечислены все операторы и их взаимосвязанность:

Таблица: Взаимосвязанность операторов

Оператор	Описание	Взаимосвязанность
CLONE NEW	Создание нового объекта	Отсутствует
< <= >= == != === !== <>	Сравнение	Отсутствует
!	Логическое НЕ	Правая
~	Поразрядное НЕ	Правая
++ --	Инкремент и декремент	Правая
(int)	Преобразование в целое число	Правая
(double) (float) (real)	Преобразование в число с плавающей точкой	Правая
(string)	Преобразование в строку	Правая
(array)	Преобразование в массив	Правая
(object)	Преобразование в объект	Правая
@	Подавление сообщения об ошибке	Правая
= += -= *= /=	Присваивание	Правая
.= %= &= |= ^= <>=	Присваивание	

Правая		
+	Сложение и унарный плюс	Левая
–	Вычитание и отрицание	Левая
*	Умножение	Левая
/	Деление	Левая
%	Модуль	Левая
	Конкатенация строк	Левая
<> & ^ |	Поразрядные операции	Левая
? :	Операция с тремя операндами	Левая
|| && and or xor	Логические операции	Левая
,	Разделение	Левая

Рассмотрим оператор присваивания, показанный в примере, где всем трем переменным присваивается значение **0**.

```
<?php
$level = $score = $time = 0;
?>
```

Такое множественное присваивание возможно только в том случае, если сначала вычисляется самая правая часть выражения, а затем

процесс продолжается справа налево.

Новичкам следует научиться в процессе работы с PHP избегать потенциальных просчетов в вопросах взаимосвязанности операторов и **всегда принудительно задавать порядок вычислений, заключая подвыражения в круглые скобки**. Это поможет и другим программистам, которые будут обслуживать ваш код, понять, что в нем происходит.

Операторы отношения

Операторы отношения проверяют значения двух операндов и возвращают логический результат, равный либо TRUE, либо FALSE. Существует три типа операторов отношения: операторы равенства, сравнения и логические операторы.

Операторы равенства

С оператором равенства == (двойным знаком равенства) мы уже не раз встречались в этой книге. Его не следует путать с оператором присваивания = (одинарным знаком равенства). В примере ниже первый оператор присваивает значение, а второй проверяет его на равенство:

```
<?php
$month = "Март";
if ($month == "Март") echo "Весна наступила";
?>
```

Как видно из примера, возвращая значение TRUE или FALSE, оператор сравнения позволяет проверять условия, используя инструкцию if. Но это еще не все, поскольку PHP является языком со слабой типизацией. Если два операнда выражения равенства имеют разные типы, PHP преобразует их к тому типу, который имеет для него наибольший смысл.

К примеру, любые строки, составленные полностью из цифр, при сравнении с числами будут преобразованы в числа. В примере ниже переменные $a и $b являются двумя разными строками, и

поэтому вряд ли стоило ожидать, что какая-то из инструкций if выведет результат.

```
<?php
$a = "1000";
$b = "+1000";
if ($a == $b) echo "1";
if ($a === $b) echo "2";
?>
```

Но если запустить этот пример, то он выведет число. Это означает, что результат вычисления первой инструкции if является TRUE. Причина в том, что обе строки сначала конвертируются в числа, и **1000** имеет такое же числовое значение, что и **+1000**.

В отличие от первой, во второй инструкции if используется **оператор тождественности** — тройной знак равенства, который удерживает PHP от автоматического преобразования типов. Поэтому переменные $a и $b сравниваются как строки и теперь считаются отличающимися друг от друга, и на экран ничего не выводится. Как и в случае с принудительным заданием уровня приоритетности операторов, если возникнут сомнения в том, будет ли PHP конвертировать типы операндов, для отмены такого поведения интерпретатора можно воспользоваться оператором тождественности.

Аналогично применению оператора равенства для определения равенства операндов можно проверить их на неравенство, используя оператор неравенства !=. В примере ниже операторы равенства и тождественности были заменены противоположными им операторами.

```
<?php
$a = "1000";
$b = "+1000";
if ($a != $b) echo "1";
if ($a !== $b) echo "2";
```

```
?>
```

Как, наверное, и ожидалось, первая инструкция if не выводит на экран число **1**, потому что в коде ставится вопрос о неравенстве числовых значений переменных $a и $b. Вместо этого будет выведено число 2, поскольку вторая инструкция if ставит вопрос о нетождественности прежнего типа операндов переменных $a и $b, и ответом будет TRUE, потому что они не тождественны.

Операторы сравнения

Используя операторы сравнения, можно расширить круг проверок, не ограничивая его только равенством и неравенством. PHP предоставляет вам для этого операторы > (больше), < (меньше), >= (больше или равно) и <= (меньше или равно). Здесь показано использование этих операторов.

```
<?php
        $a = 2; $b = 3;
        if ($a > $b)  echo "$a больше $b
";
        if ($a < $b)  echo "$a меньше $b
";
        if ($a >= $b) echo "$a больше или равно $b
";
        if ($a <= $b) echo "$a меньше или равно $b
";
?>
```

Этот пример, в котором переменная $a имеет значение **2**, а переменная $b — значение **3**, выведет на экран следующую информацию:

```
2 меньше 3
2 меньше или равно 3
```

Попробуйте самостоятельно запустить этот пример, меняя значения переменных $a и $b, чтобы увидеть результаты. Присвойте им одинаковые значения и посмотрите, что из этого

получится.

Логические операторы

Логические операторы выдают истинные или ложные результаты. Всего имеется четыре таких оператора (см. таблицу "Логические операторы").

Таблица: Логические операторы

Логический оператор	Описание
AND	Возвращает истинное значение (TRUE), если оба операнда имеют истинные значения
OR	Возвращает истинное значение (TRUE), если любой из операндов имеет истинное значение
XOR	Возвращает истинное значение (TRUE), если один из двух операндов имеет истинное значение
NOT	Возвращает истинное значение (TRUE), если операнд имеет ложное значение, или ложное значение (FALSE), если он имеет истинное значение

Использование этих операторов показано в примере ниже. Обратите внимание, что PHP требует использовать вместо слова **NOT** символ !. Кроме того, операторы могут быть составлены из букв нижнего или верхнего регистра.

```
<?php
        $a = 1; $b = 0;
        echo ($a AND $b) . "
";
        echo ($a or $b)  . "
";
        echo ($a XOR $b) . "
";
        echo !$a         . "
";
```

```
?>
```

Этот пример выводит на экран **NULL, 1, 1, NULL**. Это значит, что только вторая и третья инструкции echo получают в результате вычисления значение TRUE. (Следует помнить, что NULL, или ничто, отображает значение FALSE.) Такой результат получается, потому что оператору AND, чтобы вернуть значение TRUE, нужно, чтобы оба операнда имели истинное значение, а четвертый оператор проводит над значением переменной $a операцию NOT, превращая его из TRUE (значения, равного единице) в FALSE. Если есть желание поэкспериментировать, запустите этот код, присваивая переменным $a и $b разные значения, выбранные из 1 и 0.

Занимаясь программированием, следует помнить, что у операторов AND и OR более низкий уровень приоритета, чем у других версий этих операторов – && и ||. Поэтому в сложных выражениях более безопасным будет, наверное, применение операторов && и ||.

Использование в инструкции if оператора OR может стать причиной непредвиденных проблем, поскольку второй операнд не будет вычисляться, если в результате вычисления первого операнда уже получено значение TRUE. В примере функция getnext никогда не будет вызвана, если переменная $finished имеет значение **1**.

```
<?php
if ($finished == 1 OR getnext() == 1) exit;
?>
```

Если нужно, чтобы функция getnext вызывалась для каждой инструкции if, следует внести в код изменения, показанные в примере изменения в инструкции if ... OR, гарантирующие вызов функции getnext

```
<?php
$gn = getnext();
if ($finished == 1 OR $gn == 1) exit;
```

?>

В этом случае код в функции getnext будет выполнен и возвращенное значение сохранится в переменной $gn еще до выполнения инструкции if. Другое решение заключается в том, чтобы обеспечить выполнение функции getnext за счет простой перестановки условий местами, поскольку тогда вызов функции будет появляться в выражении первым.

В таблице ниже показаны все допустимые варианты использования логических операторов. Следует заметить, что !TRUE является эквивалентом FALSE, а !FALSE — эквивалентом TRUE.

Таблица: Все логические выражения, допустимые в PHP

Входные данные	Операторы и результаты			
a	b	AND	OR	XOR
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE

```
FALSE
TRUE
FALSE
TRUE
TRUE
```

```
FALSE
FALSE
FALSE
FALSE
FALSE
```

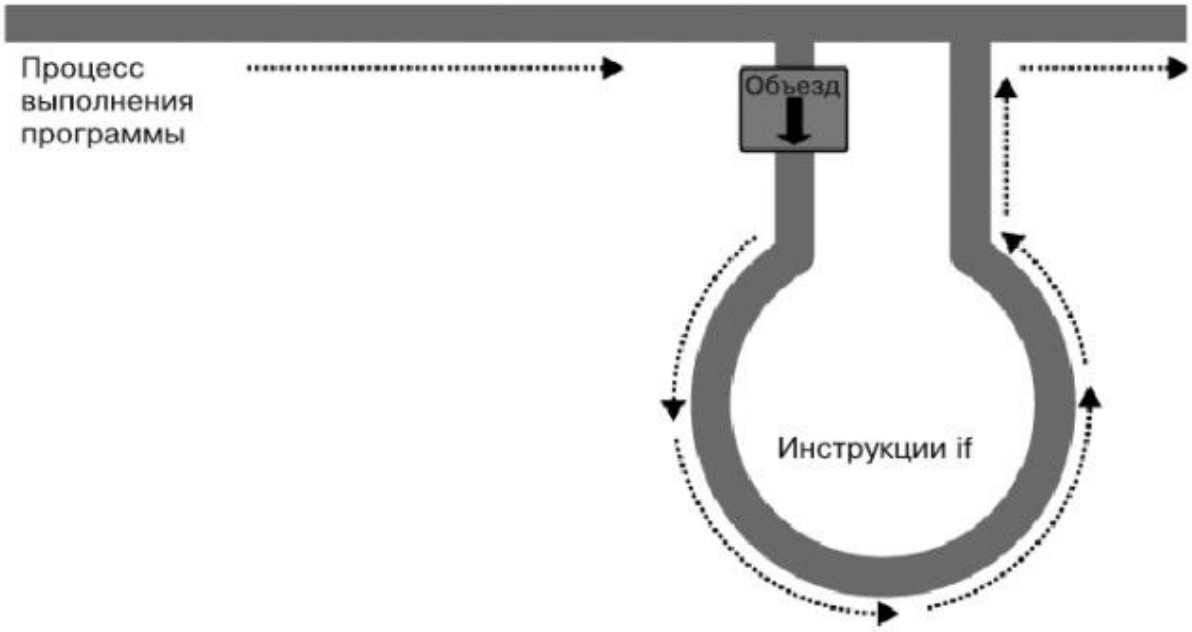
Условия

Условия изменяют процесс выполнения программы. Они позволяют задавать конкретные вопросы и по-разному реагировать на полученные ответы. Условия играют важную роль при разработке динамических веб-страниц — основной цели использования PHP, поскольку облегчают создание разных вариантов выводимой при каждом просмотре веб-страницы информации. Существует три типа нециклических условных инструкций: if, switch и ?. Называя их нециклическими, я имел в виду, что после действий, инициированных инструкцией, процесс выполнения программы продолжается, а при использовании циклических условных инструкций (которые еще предстоит рассмотреть) код выполняется снова и снова до тех пор, пока не будет соблюдено определенное условие.

Инструкция if

Процесс выполнения программы можно представить себе как езду на машине по однополосной магистрали. Эта магистраль большей частью прямолинейна, но иногда встречаются различные дорожные знаки, задающие направление движения. Когда встречается инструкция if, можно представить, что машина подошла к знаку объезда, предписаниям которого необходимо следовать, когда

определенные условия вычисляются как TRUE. При этом вы съезжаете с магистрали и следуете по объездному пути до тех пор, пока не вернетесь снова на магистраль и не продолжите движение по исходному маршруту. Или же, если условие не вычисляется как TRUE, вы игнорируете объезд и продолжаете ехать по магистрали как ни в чем не бывало (см. рисунок).



Процесс выполнения программы похож на движение по однополосной магистрали

Содержимым условной инструкции if может быть любое допустимое PHP-выражение, включая равенство, сравнение, проверку на нуль и NULL и даже значения, возвращаемые функциями (как встроенными, так и созданными самостоятельно). Действия, предпринимаемые при вычислении условия в TRUE, помещаются, как правило, в фигурные скобки { }. Но эти скобки можно опустить, если нужно выполнить всего одну инструкцию. Тем не менее, если всегда использовать фигурные скобки, можно избежать «охоты» на трудно отслеживаемые ошибки, возникающие, к примеру, когда к условной инструкции добавляется еще одна строка, но забывается о необходимости добавить фигурные скобки, из-за чего строка не вычисляется.

(Учтите, что в целях экономии места и доходчивости материала,

если в примерах, приводимых в книге, была всего одна исполняемая инструкция, я не следовал этому совету и опускал фигурные скобки.) В примере следует представить, что подошел конец месяца и нужно платить по всем счетам, поэтому вы проводите некоторые операции с банковским счетом.

```
<?php
if ($bank_balance < 100)
{
$money
= 1000;
$bank_balance += $money;
}
?>
```

В этом примере проверяется, не стал ли баланс ниже **$100** (или **100** единиц другой используемой вами валюты). Если баланс стал ниже этой суммы, вы платите сами себе **$1000**, а затем прибавляете их к балансу. (Хорошо бы так просто зарабатывать деньги!) Если баланс счета в банке равен **$100** или превышает эту сумму, условные инструкции игнорируются и процесс выполнения программы переходит на следующую строку кода (которая здесь не показана). Одни разработчики предпочитают ставить первую фигурную скобку справа от условного выражения, а другие начинают с нее новую строку. В этой книге открывающая фигурная скобка располагается обычно на новой строке. Подойдет любой из этих вариантов, поскольку PHP позволяет оставлять на ваше усмотрение какие угодно свободные пространства (пробелы, символы новых строк и табуляции). Но код будет легче читаться и отлаживаться, если у каждого уровня условий будет свой отступ, сформированный с помощью символа табуляции.

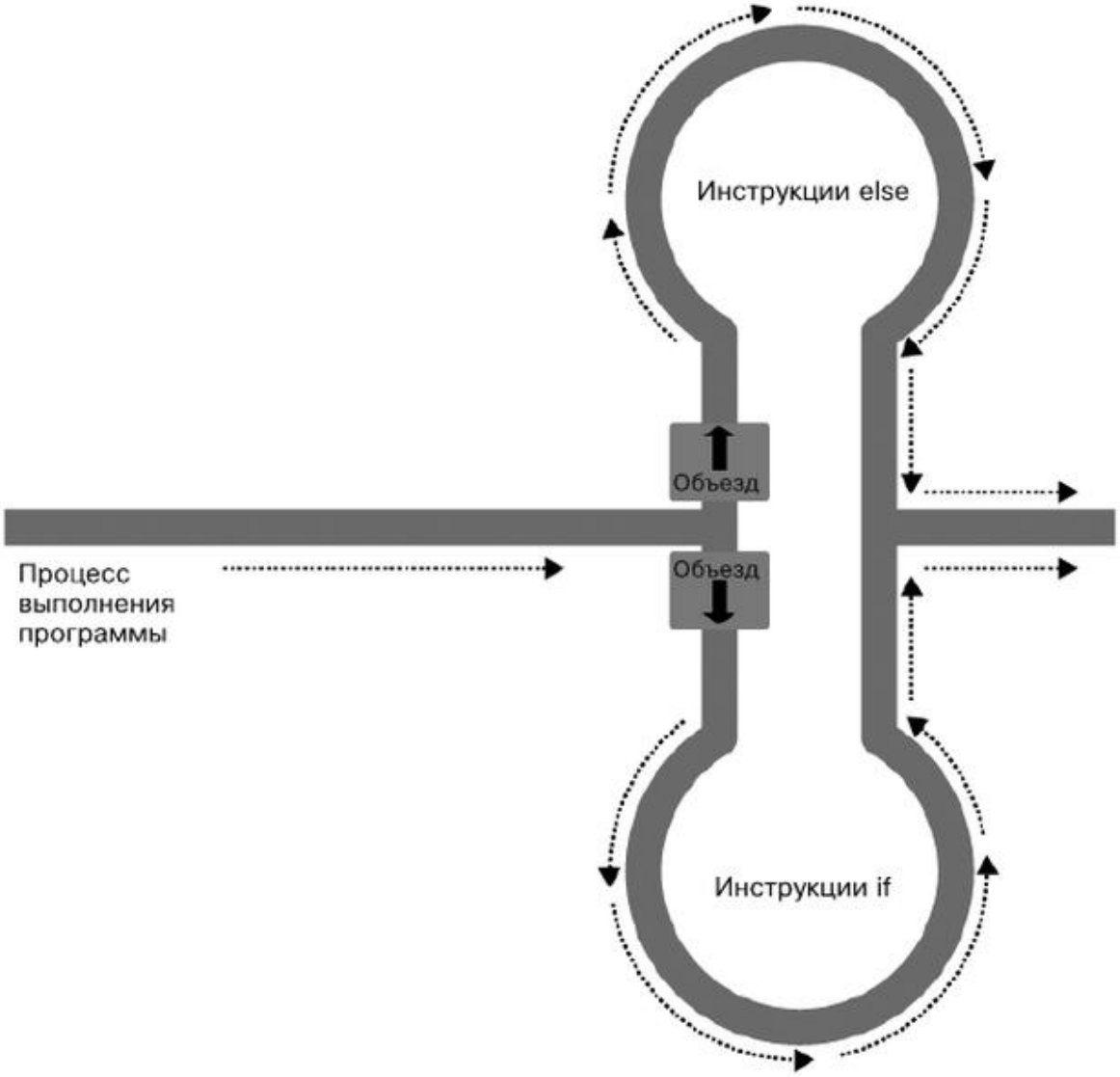
Инструкция else

Бывают случаи, когда условие не вычисляется как TRUE, но вам не хочется сразу же продолжать выполнение основного кода программы, а вместо этого нужно сделать что-либо другое. Тогда пригодится инструкция else. С ее помощью на вашей магистрали

можно организовать второй объезд, показанный на рисунке. Если при использовании конструкции if...else условие вычисляется как TRUE, то выполняется первая условная инструкция. Но если это условие вычисляется как FALSE, то выполняется вторая условная инструкция. Для выполнения должна быть выбрана одна из этих двух инструкций, но обе сразу они не будут выполнены ни при каких условиях, и обязательно будет выполнена хотя бы одна из них.

Использование конструкции if...else показано в примере:

```
<?php
if ($bank_balance < 100)
{
$money
= 1000;
$bank_balance += $money;
}
else
{
$savings
+= 50;
$bank_balance -= 50;
}
?>
```



Теперь у магистрали есть объезд if и объезд else

Если в этом примере будет установлено, что в банке лежит **$100** или более, то выполняется инструкция else, с помощью которой часть этих денег перемещается на ваш сберегательный счет. Точно так же, как и у if, если у инструкции else есть только одна условная инструкция, то фигурные скобки можно не ставить. (Хотя фигурные скобки рекомендуется использовать в любом случае. Во-первых, при их наличии проще разобраться в коде, а во-вторых, они облегчают последующее добавление инструкций к этому ветвлению.)

Инструкция elseif

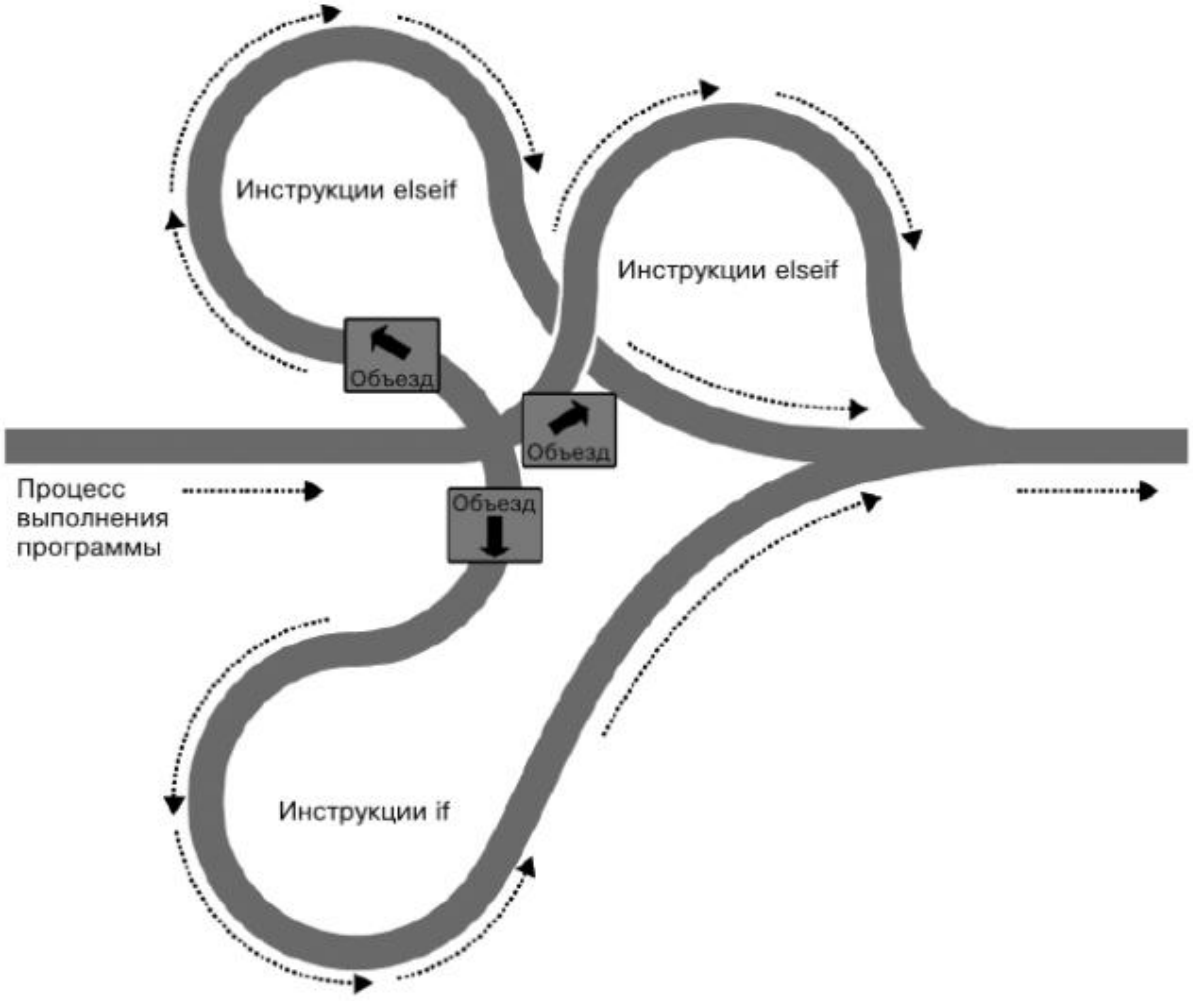
Случается, что на основе последовательности условий нужно

осуществить сразу несколько действий. Достичь желаемого результата можно, используя инструкцию elseif. Можно предположить, что она похожа на инструкцию else, за исключением того, что до кода условия вставляется еще одно условное выражение. Полноценная конструкция if...elseif...else показана в примере:

```
<?php
if ($bank_balance < 100)
{
$money
= 1000;
$bank_balance += $money;
}
elseif ($bank_balance > 200)
{
$savings
+= 100;
$bank_balance -= 100;
}
else
{
$savings
+= 50;
$bank_balance -= 50;
}
?>
```

В этом примере инструкция elseif была вставлена между инструкциями if и else. Она проверяет, не превышает ли баланс банковского счета сумму **$200**, и если превышает, принимается решение о том, что в этом месяце можно позволить себе положить на сберегательный счет **$100**. Это все можно представить в виде набора объездов в нескольких направлениях (cv рис.). Инструкция else завершает либо конструкцию if...else, либо конструкцию if...elseif...else. Если она не нужна, то финальную инструкцию else можно опустить, но ни одна из этих инструкций не должна стоять перед инструкцией elseif, точно так же, как ни одна инструкция elseif не должна стоять перед

инструкцией if.



Магистраль с объездами if, elseif и else

Количество используемых инструкций elseif не ограничено. Но по мере роста количества этих инструкций будет лучше, наверное, обратиться к инструкции switch, если, конечно, она отвечает вашим запросам. Именно ее мы сейчас и рассмотрим.

Инструкция switch

Инструкция switch применяется в тех случаях, когда у одной переменной или у результата вычисления выражения может быть несколько значений, каждое из которых должно вызывать особую функцию. Рассмотрим, например, управляемую кодом PHP систему меню, которая в соответствии с пожеланием пользователя передает отдельную строку коду основного меню. Предположим, что есть следующие варианты: Home, About, News, Login и Links,

а переменная $page принимает одно из этих значений в соответствии с информацией, введенной пользователем.

Код реализации этого замысла с использованием конструкции if...elseif...
else может иметь вид, показанный в примере 4.22.

```
<?php
        if     ($page == "Home")  echo "Вы выбрали Home";
        elseif ($page == "About") echo "Вы выбрали About";
        elseif ($page == "News")  echo "Вы выбрали News";
        elseif ($page == "Login") echo "Вы выбрали Login";
        elseif ($page == "Links") echo "Вы выбрали Links";
?>
```

Код, в котором используется инструкция switch, показан в примере:

```
<?php
  switch ($page)
  {
      case "Home":
          echo "Вы выбрали Home";
         break;
      case "About":
          echo "Вы выбрали About";
         break;
      case "News":
          echo "Вы выбрали News";
         break;
      case "Login":
          echo "Вы выбрали Login";
         break;
      case "Links":
          echo "Вы выбрали Links";
         break;
  }
?>
```

Как видите, переменная $page используется только один раз — в

начале инструкции switch. После этого все соответствия проверяются командой case. Когда найдено соответствие, выполняется его условная инструкция. Разумеется, в настоящей программе в этом месте будет применяться код отображения или перехода на страницу, а не простое сообщение пользователю о том, что именно он выбрал.

В инструкциях switch внутри команд case фигурные скобки не используются. Вместо этого инструкции начинаются с двоеточия и заканчиваются командой break. Тем не менее весь перечень команд case в инструкции switch заключается в фигурные скобки.

Прекращение работы инструкции switch

Если нужно, чтобы инструкция switch прекратила свою работу из-за выполнения условия, используется команда break. Она предписывает PHP прекратить работу инструкции switch и перейти к выполнению следующей инструкции.

Если в примере, приведённом выше, не расставить команды break и результат вычисления команды case, проверяющей условие Home, получится TRUE, то будут выполнены все пять условных инструкций, следующих за командами case. Или же, если переменная $page имела значение News, то, начиная с этого места, будут выполнены все оставшиеся команды case. Это сделано преднамеренно для расширения возможностей программирования, но в большинстве случаев не следует забывать ставить команду break во всех местах, где набор условных инструкций, следующих за командами case, завершает свою работу. Надо сказать, что случайный пропуск команд break является весьма распространенной ошибкой.

Действие по умолчанию

Типичным требованием для инструкции switch является переход к действию по умолчанию, если не будет выполнено ни одно из условий, содержащихся в командах case. Например, к коду меню можно непосредственно перед закрывающей фигурной скобкой добавить код, показанный здесь:

```
default: echo "Нераспознанный выбор";
break;
```

Хотя здесь ставить команду break не требуется, поскольку default является заключительной внутренней инструкцией и процесс выполнения программы автоматически продолжится после закрывающей фигурной скобки, но если вы решите поставить инструкцию default выше этого места, ей определенно понадобится команда break, для того чтобы процесс выполнения программы не затронул все стоящие ниже условные инструкции. Лучше перестраховаться и в конце этой инструкции всегда ставить команду break.

Альтернативный синтаксис

Открывающую фигурную скобку инструкции switch можно заменить двоеточием, а закрывающую — командой endswitch (см. пример ниже). Такой вариант используется довольно редко, и здесь он упоминается на тот случай, если придется столкнуться с ним в коде, созданном кем-нибудь другим.

```
<?php
        switch ($page):
                case "Home":
                        echo "Вы выбрали Home";
                        break;
                // и т.д. ...
                case "Links":
                        echo "Вы выбрали Links";
                        break;
        endswitch;
?>
```

Оператор ?

Использование трехкомпонентного оператора ? позволяет избежать многословности инструкций if и else. Необычность этого оператора заключается в том, что он использует не два, как большинство других операторов, а три операнда. У нас уже

состоялось краткое знакомство с этим оператором при выяснении разницы между print и echo, где он приводился в качестве примера оператора, который хорошо работает с print, но не работает с echo. Оператору ? передаются выражение, которое он должен вычислить, и два выполняемых оператора: один для выполнения, когда результат вычисления выражения TRUE, а другой — когда FALSE.

В примере показан код, который может использоваться для вывода предупреждения об уровне топлива в автомобиле на его панель приборов.

```
<?php
echo $fuel <= 1 ? "Требуется дозаправка" : "Топлива еще достаточно";
?>
```

Если топлива остается всего 1 галлон или меньше (иными словами, переменная $fuel имеет значение, равное единице или меньше ее), то этот оператор возвращает предыдущей команде echo строку «Требуется дозаправка». В противном случае он возвращает строку «Топлива еще достаточно». Значение, возвращаемое оператором ?, можно также присвоить какой-нибудь переменной:

```
<?php
$enough = $fuel <= 1 ? FALSE : TRUE;
?>
```

В этом примере переменной $enough будет присвоено значение TRUE только в том случае, если в баке более 1 галлона топлива, в противном случае ей будет присвоено значение FALSE. Если вы считаете синтаксис оператора ? слишком запутанным, то можете вместо него воспользоваться инструкцией if, но о нем все равно нужно знать, поскольку он может встретиться в программном коде, созданном другим программистом. Чтение кода, в котором используется этот оператор, может быть сильно затруднено из-за частого применения в нескольких местах одной и той же

переменной. Например, весьма популярен код такого вида:

```
$saved = $saved >= $new ? $saved : $new;
```

Понять, что он делает, можно только после тщательного разбора:

```
$saved =                    // Присваивание значения переменной
$saved
        $saved >= $new     // Сравнение $saved и $new
    ?                       // Если сравнение выдает истинный
результат ...
         $saved             // ... ей присваивается текущее
значение $saved
    :                       // Если сравнение выдает ложный
результат ...
         $new;              // ... ей присваивается значение
переменной $new
```

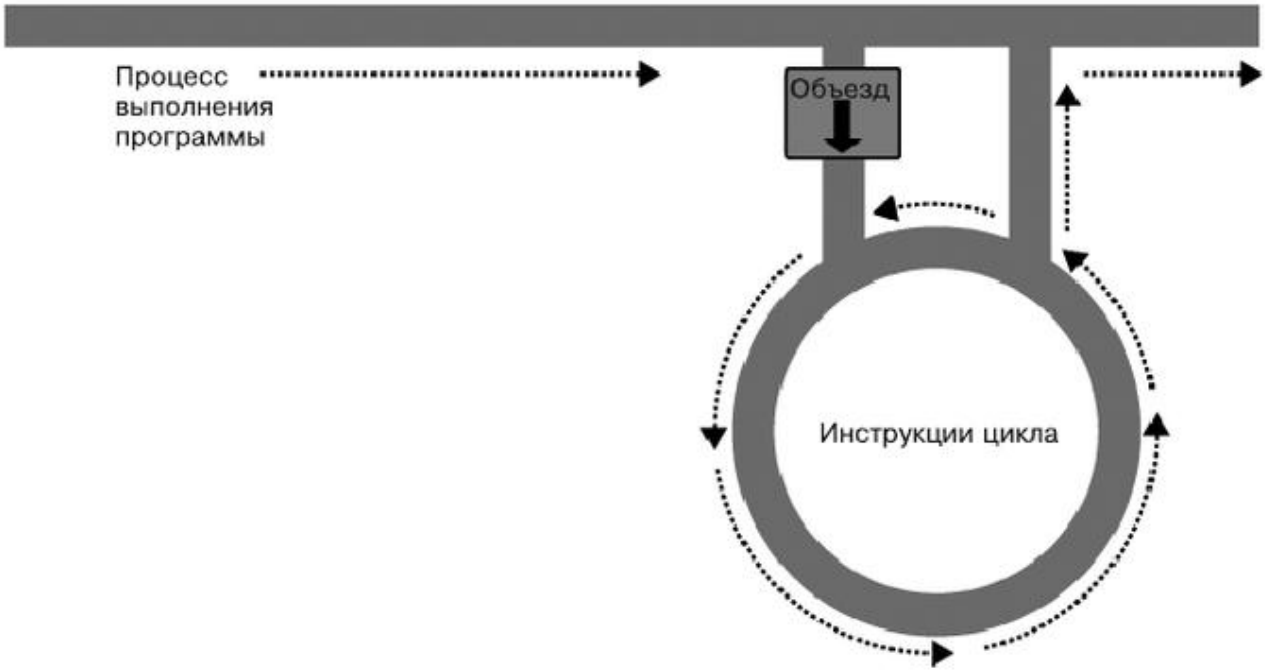
Это весьма компактный способ отслеживания самого большого значения, которое может встретиться в процессе выполнения программы. Самое большое значение содержится в переменной $saved и при поступлении нового значения сравнивается со значением переменной $new. Программисты, освоившие оператор ?, считают, что для таких коротких сравнений его удобнее применять, чем инструкции if. Если этот оператор не используется для создания компактного кода, то он обычно применяется для принятия решений, умещающихся на одной строке, например для проверки того, установлено ли значение переменной, перед передачей ее функции.

Организация циклов

Компьютеры славятся своей способностью быстро и неутомимо повторять вычисления. Зачастую от программы требуется снова и снова повторять одну и ту же последовательность кода, пока не произойдет какое-нибудь событие, например ввод значения пользователем или достижение программой своего естественного

окончания. Имеющиеся в PHP разнообразные структуры организации циклов предоставляют великолепные способы решения подобных задач.

Чтобы представить, как это работает, посмотрите на рисунок. Он очень похож на метафору с магистралью, которая использовалась для иллюстрации работы инструкции if, за исключением того, что у объезда также есть замкнутый участок, из которого машина может выйти только при соблюдении определенных программных условий.



Представление цикла как части программы магистральной разметки

Циклы while

Превратим автомобильную панель приборов из предыдущего примера в цикл, постоянно проверяющий уровень топлива при езде на машине, в котором используется инструкция цикла while:

```
<?php
        $fuel = 10;
        while ($fuel > 1)
        {
```

```
                // Продолжение поездки...
                echo "Топлива еще достаточно";
        }
?>
```

Вообще-то, вы можете предпочесть выводу текста горящий зеленый сигнал, но суть в том, что любая разновидность позитивной индикации об уровне топлива помещается в цикл while. Кстати, учтите, что если вы запустите этот пример на выполнение, то он будет постоянно выводить строку, до тех пор пока вы не остановите работу браузера. Здесь, как и в случае с инструкциями if, для хранения инструкций внутри цикла while используются фигурные скобки, если только в этом цикле не задействована лишь одна инструкция.

В примере показан еще один вариант использования цикла while, в котором выводится таблица умножения на **12**.

```
<?php
$count = 1;
while ($count <= 12)
{
        echo "Число $count, умноженное на 12, равно " . $count
* 12 . "
";
        ++$count;
}
?>
```

В этом примере переменной $count присваивается начальное значение **1**, а затем запускается цикл while, в котором используется выражение сравнения $count <= 12. Цикл будет выполняться до тех пор, пока значение переменной не станет больше **12**.

Данный код выведет следующий текст:

```
Число 1, умноженное на 12, равно 12
Число 2, умноженное на 12, равно 24
```

Число 3, умноженное на 12, равно 36

и т.д.

Внутри цикла осуществляется вывод строки, а также значения переменной $count, умноженного на **12**. Чтобы упорядочить вывод, после всего этого использован тег
, вызывающий переход на новую строку. Затем перед закрывающей фигурной скобкой, предписывающей PHP вернуться к началу цикла, значение переменной $count увеличивается на единицу.

Теперь значение переменной $count опять проверяется, чтобы узнать, не превышает ли оно число **12**. Оно не превышает этого числа, но теперь оно равно **2**, и после **11** последующих прохождений цикла оно станет равно **13**. Когда это произойдет, код, находящийся внутри цикла while, будет пропущен и станет выполняться код, следующий за циклом, в данном случае это будет завершение программы. При отсутствии оператора ++$count (вместо которого с таким же успехом может быть применен оператор $count++) этот цикл будет похож на первый, показанный в этом разделе. Он никогда не закончится и будет снова и снова выводить один и тот же результат 1 · 12.

Но есть и более изящный способ написания этого цикла, который должен вам понравиться. Посмотрите:

```
<?php
$count = 0;
while (++$count <= 12)
echo "Число $count, умноженное на 12, равно " . $count * 12 .
"
";
?>
```

В этом примере оператор ++$count был удален из тела цикла while и помещен непосредственно в выражение условия цикла. Теперь PHP вычисляет значение переменной $count в начале каждого прохода цикла (итерации) и, заметив, что перед именем переменной стоит оператор инкремента, сначала увеличивает

значение переменной на 1 и только потом сравнивает его с числом 12. Следовательно, теперь переменной $count присваивается начальное значение 0, а не 1, поскольку это значение увеличивается сразу же, как только происходит вход в цикл. Если оставить начальное значение, равное 1, то будут выведены результаты для чисел между 2 и 12.

Циклы do...while

Цикл do...while представляет собой небольшую модификацию цикла while, используемую в том случае, когда нужно, чтобы блок кода был исполнен хотя бы один раз, а условие проверялось только после этого. В примере показана модифицированная версия таблицы умножения на 12, в которой использован этот цикл:

```
<?php
$count = 1;
do
        echo "Число $count, умноженное на 12, равно " . $count
* 12 . "
";
while (++$count <= 12);
?>
```

Заметьте, что теперь мы вернулись к присваиванию переменной $count начального значения **1** (а не 0), потому что код выполняется сразу же, без увеличения значения переменной на 1. Во всем остальном этот код очень похож на показанный в предыдущем примере. Разумеется, если внутри цикла do...while находится несколько инструкций, то не следует забывать ставить вокруг них фигурные скобки, как показано:

```
<?php
$count = 1;
do {
        echo "Число $count, умноженное на 12, равно " . $count
* 12;
        echo "
```

```
";
} while (++$count <= 12);
?>
```

Циклы for

Цикл for, являющийся последней разновидностью инструкций цикла, к тому же еще и самый мощный из них, поскольку в нем сочетаются возможности установки значения переменных при входе в цикл, проверки соблюдения условия при каждом проходе цикла (итерации) и модификации значений переменных после каждой итерации. В примере показнаа возможность вывода таблицы умножения с использованием цикла for.

```
<?php
for ($count = 1 ; $count <= 12 ; ++$count)
echo "Число $count, умноженное на 12, равно " . $count * 12 .
"
";
?>
```

Как видите, весь код сведен к одной инструкции for, в которой содержится одна условная инструкция. И вот что из этого получается. Каждая инструкция for воспринимает три параметра:

- выражение инициализации;
- выражение условия;
- выражение модификации.

Эти три выражения отделяются друг от друга точкой с запятой: for (выражение1 ; выражение2 ; выражение3). В начале первой итерации выполняется выражение инициализации. В нашем коде таблицы умножения переменная $count инициализируется значением **1**. Затем при каждой итерации проверяется выражение условия (в данном случае $count <= 12), и выход из цикла осуществляется только в том случае, если результат вычисления условия будет TRUE. И наконец, в завершение каждой итерации выполняется выражение модификации. В случае с таблицей умножения значение переменной $count увеличивается на **1**.

Эта структура в явном виде исключает любые требования по размещению управляющих элементов цикла в его собственном теле, освобождая его для инструкций, требующих циклического выполнения. Если в теле цикла for содержится несколько инструкций, не забудьте воспользоваться фигурными скобками:
Пример 4.34. Цикл for из примера 4.33 с добавлением фигурных скобок

```
<?php
for ($count = 1 ; $count <= 12 ; ++$count)
{
echo "Число $count, умноженное на 12, равно " . $count * 12;
echo "
";
}
?>
```

Сравним условия, при которых следует использовать циклы for, с условиями, при которых нужно применять циклы while. Цикл for явно создавался под отдельное значение, изменяющееся на постоянную величину. Обычно мы имеем дело с увеличивающимся значением — это похоже на то, как если бы вам был передан перечень того, что выбрал пользователь, и от вас требуется обработать каждый его выбор по очереди. Но переменную можно видоизменять по вашему усмотрению. Более сложная форма инструкции for позволяет даже осуществлять со всеми тремя параметрами сразу несколько операций:

```
for ($i = 1, $j = 1 ; $i + $j < 10 ; $i++ , $j++)
{
// ...
}
```

Но новичкам использовать такую сложную форму не рекомендуется. Здесь главное — отличать запятые от точки с запятой. Все три параметра должны быть отделены друг от друга точкой с запятой. Несколько операторов внутри каждого параметра должны быть отделены друг от друга запятыми. Первый и третий параметры в

предыдущем примере содержат по два оператора:

```
$i = 1, $j = 1   // Инициализация переменных $i и $j
$i + $j < 10     // Условие окончания работы цикла
$i++ , $j++     // Модификация $i и $j в конце каждой итерации
```

Главное, что следует уяснить из этого примера, — три секции параметров должны разделяться точкой с запятой, а не запятыми (которые могут использоваться только для разделения операторов внутри каждой секции параметров). Тогда при каких условиях следует отдавать предпочтение инструкциям while перед инструкциями for? Когда ваше условие не зависит от простого изменения переменной на постоянной основе. Например, инструкция while применяется в том случае, если нужно проверить, не введено ли какое-то определенное значение или не возникла ли какая-то конкретная ошибка, и завершить цикл сразу же, как только это произойдет.

Прекращение работы цикла

Прекратить работу цикла for можно точно так же, как и работу рассмотренной уже инструкции switch, — используя команду break. К примеру, это может понадобиться, когда одна из ваших инструкций вернет ошибку и продолжать выполнение цикла станет небезопасно. Один из таких случаев может произойти, когда при записи файла возникнет ошибка, возможно, из-за нехватки места на диске.

```
<?php
$fp = fopen("text.txt", 'wb');
for ($j = 0 ; $j < 100 ; ++$j)
{
        $written = fwrite($fp, "data");
        if ($written == FALSE) break;
}
fclose($fp);
?>
```

Это наиболее сложный из всех ранее приведенных фрагментов кода, но вы уже готовы к его пониманию. Команды обработки файлов будут рассмотрены в одной из следующих глав, а сейчас нужно лишь знать, что в первой строке кода открывается файл **text.txt** для записи в двоичном режиме, а затем переменной $fp возвращается указатель на него, который в дальнейшем используется для ссылки на этот открытый файл. Затем осуществляется 100 проходов цикла (от 0 до 99), записывающих строку data в файл. После каждой записи функция fwrite присваивает переменной $written значение, представляющее собой количество успешно записанных символов. Но если происходит ошибка, то функция fwrite присваивает этой переменной значение FALSE.

Поведение функции fwrite облегчает коду проверку переменной $written на наличие значения FALSE, и если она имеет такое значение, код прекращает работу цикла и передает управление инструкции, закрывающей файл.

При желании улучшить код можно упростить строку:

```
if ($written == FALSE) break;
```

за счет использования оператора NOT:

```
if (!$written) break;
```

Фактически пара инструкций, находящихся внутри цикла, может быть сокращена до одной:

```
if (!fwrite($fp, "data")) break;
```

Но команда break обладает более широкими возможностями, чем можно было бы предположить, поскольку, если нужно прекратить работу кода, вложенного глубже, чем на один уровень, после команды break можно поставить число, показывающее, работу скольких уровней нужно прекратить, например:

```
break 2;
```

Инструкция continue

Инструкция continue немного похожа на команду break, только она предписывает PHP остановить процесс текущего цикла и перейти непосредственно к его следующей итерации, то есть вместо прекращения работы всего цикла PHP осуществляет выход только из текущей итерации.

Этот прием может пригодиться в тех случаях, когда известно, что нет смысла продолжать выполнение текущего цикла и нужно сберечь процессорное время или избежать ошибки путем перехода сразу к следующей итерации цикла. В примере ниже инструкция continue используется для того, чтобы избежать ошибки деления на нуль за счет ее вызова в тот момент, когда переменная $j имеет значение **0**.

```
<?php
$j = 10;
while ($j > –10)
{
        $j--;
        if ($j == 0) continue;
        echo (10 / $j) . "
";
}
?>
```

Для всех значений переменной $j в диапазоне чисел между **10** и **–10**, за исключением **0**, отображается результат деления числа **10** на значение переменной $j. Но для конкретного случая, когда значение $j равно **0**, вызывается инструкция continue и дальнейшее выполнение итерации сразу же пропускается с переходом к следующей итерации цикла.

Неявное и явное преобразование типов

PHP является языком со слабой типизацией, который позволяет объявлять переменную и ее тип путем простого использования этой переменной. При необходимости он также осуществляет автоматическое преобразование одного типа в другой. Этот процесс называется неявным преобразованием типов. Однако могут возникнуть ситуации, когда присущее PHP неявное преобразование типов станет совсем нежелательным действием. Рассматривая пример ниже, обратите внимание на то, что входные данные для операции деления являются целыми числами. По умолчанию PHP осуществляет преобразование выходных данных к числу с плавающей точкой, чтобы получалось наиболее точное значение — **4,66** и **6** в периоде.

```
<?php
$a = 56;
$b = 12;
$c = $a / $b;
echo $c;
?>
```

Но что делать, если вместо этого нужно получить значение переменной $c в виде целого числа? Этого можно добиться разными способами, одним из которых является принудительное преобразование результата $a/$b в целое число путем использования оператора преобразования (int):

```
$c = (int) ($a / $b);
```

Такой способ называется явным преобразованием типов. Обратите внимание, что для обеспечения преобразования в целое число значения всего выражения это выражение помещено в круглые скобки. В противном случае преобразованию подверглось бы только значение переменной $a, что не имело бы никакого смысла, поскольку деление на значение переменной $b все равно

вернуло бы результат в виде числа с плавающей точкой. Можно провести явное преобразование значений в те типы, которые показаны в таблице но обычно его можно избежать, используя преобразование за счет вызова одной из встроенных функций PHP. Например, для получения целочисленного значения можно использовать функцию intval. Этот раздел, как и многие другие в данной книге, предназначен в основном для того, чтобы помочь разобраться с чужим кодом, который может вам встретиться.

Таблица: Типы преобразований, доступных в PHP

Тип преобразования	Описание
(int) (integer)	Преобразование в целое число путем отбрасывания десятичной части
(bool) (boolean)	Преобразование в логическое значение
(float) (double) (real)	Преобразование в число с плавающей точкой
(string)	Преобразование в строку
(array)	Преобразование в массив
(object)	Преобразование в объект

Динамическое связывание в PHP

Поскольку PHP является языком программирования и получаемая в результате его работы выходная информация может быть совершенно разной для различных пользователей, есть возможность запускать целый сайт из одной веб-страницы, созданной с помощью PHP. При каждом щелчке пользователя на каком-нибудь элементе подробности могут отправляться назад той же веб-странице, которая будет принимать решение, что делать дальше, в соответствии с различными объектами cookie и/или данными сессии, которые могут быть сохранены.

Но несмотря на возможность создания таким способом целого сайта, этого делать не рекомендуется, поскольку исходный код будет все время разрастаться и приобретет громадные размеры по мере того, как ему придется принимать во внимание разнообразные действия пользователя.

Будет куда более благоразумно разделить разработку сайта на несколько разных частей. Например, один автономный процесс будет заниматься подпиской на сайт со всеми вытекающими отсюда проверками допустимости адреса электронной почты, незадействованности имени пользователя и т.д.

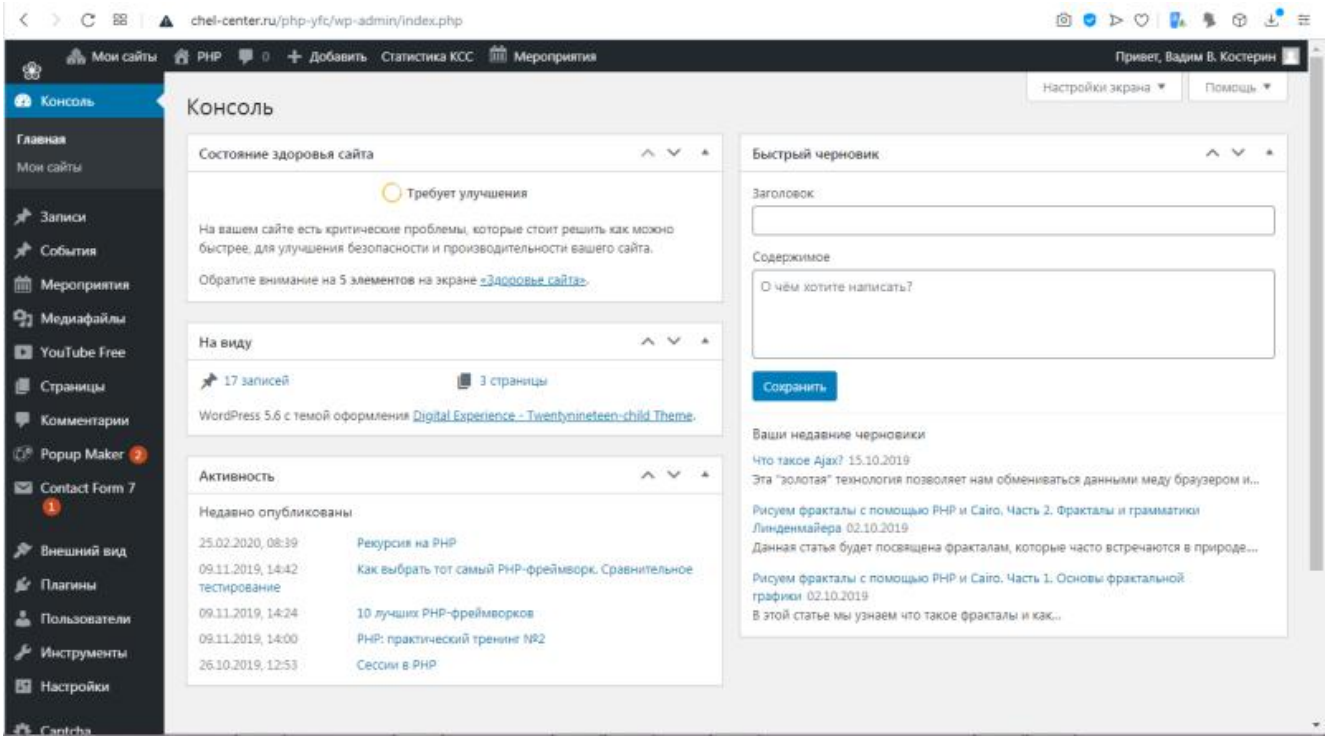
Второй модуль неплохо было бы создать для регистрации пользователей, предшествующей их допуску к основной части вашего сайта. Затем можно создать модуль вывода сообщений, в котором пользователи могли бы оставлять свои комментарии, модуль, содержащий ссылки и полезную информацию, еще один модуль, позволяющий загружать на сайт фотографии, и т.д. Как только будет создано средство для отслеживания действий пользователя на вашем сайте, использующее объекты cookie или переменные сессии (оба этих средства будут более подробно рассмотрены в следующих главах), можно разделить сайт на удобные секции PHP-кода, каждая из которых будет независима от других.

Таким образом, вы существенно облегчите себе будущую разработку каждого нового свойства и обслуживание уже имеющихся.

Динамическое связывание в действии

Одним из наиболее популярных в настоящее время приложений, управляемых PHP, является платформа для ведения блогов WordPress (см. рисунок). При ведении или чтении блога этого можно и не понять, но для каждой основной секции выделен свой основной PHP-файл, а огромное количество совместно используемых функций помещено в отдельные файлы, которые

включаются основными PHP-страницами по мере необходимости.



Платформа WordPress, предназначенная для ведения блогов, написана на PHP

Вся платформа держится на закулисном отслеживании сессии, поэтому вы вряд ли знаете о том, когда осуществляется переход от одной подчиненной секции к другой. Поэтому, если веб-разработчик хочет провести тонкую настройку WordPress, ему не трудно найти конкретный файл, который для этого применяется, и выполнить его проверку и отладку, не теряя понапрасну времени на не связанные с ним части программы. Когда в следующий раз будете использовать WordPress, проследите за адресной строкой своего браузера, особенно при управлении блогом, и тогда вы сможете заметить обращения к разнообразным PHP-файлам, которые используются в этом приложении.

В текущей главе были рассмотрены обширные сведения, закладывающие основу для дальнейшего изучения материала книги. Теперь вы уже должны уметь составлять свои собственные небольшие PHP-программы. Но перед тем, как перейти к следующей главе, посвященной функциям и объектам, можете проверить приобретенные знания, ответив на следующие вопросы.