Глава 9. Ссылки

Учебник по Perl

Содержание [скрыть]

Данные, используемые программой, размещаются в оперативной памяти компьютера. Каждая переменная имеет свой адрес и свое значение, которое хранится по этому адресу. Адрес переменной является информацией, которую также можно использовать в программе.

Ссылка на некоторую переменную содержит адрес этой переменной в оперативной памяти. Говорят, что ссылка *указывает* на переменную. Ссылки широко используются в современных языках программирования, таких как Pascal, C/C++. Вместо слова "ссылка" для обозначения термина может применяться слово "указатель". Основной областью применения ссылок является создание сложных структур данных, способных изменяться во время выполнения программы. Для ссылок используются специальные обозначения. В языке C это символ "*" перед именем переменной. В языке Pascal существует специальный тип данных для описания ссылок-переменных. Признаком этого типа является символ "^" перед идентификатором, описывающим базовый тип данных. Ссылка может быть переменной или константой.

Ссылка в языке Perl – это обычная скалярная величина, в которой хранится некоторый адрес в оперативной памяти.

Виды ссылок

Ссылка в языке Perl может указывать на любой фрагмент данных. Фрагментом данных здесь мы называем любую переменную, константу или часть кода программы. Тип ссылки определяется типом данных, на которые она указывает. Таким образом, существуют следующие типы ссылок: ссылка на скалярную величину, ссылка на массив, ссылка на хеш, ссылка на функцию. Нельзя использовать ссылку одного типа там, где контекст выражения требует присутствия ссылки другого типа, например, использовать ссылку на массив вместо ссылки на хеш-массив. Поскольку сама ссылка является скалярной величиной, то, естественно, существует ссылка на ссылку. Имеется еще один вид ссылок, который мы в свое время рассмотрим подробнее. Это ссылки на данные типа typegiob. Тип typeglob является внутренним типом языка Perl, который служит для обозначения переменных разных типов, имеющих общее имя. Принадлежность к типу typeglob, обозначается префиксом "*". Например, запись *abc обозначает всю совокупность, а также любую из следующих переменных: скаляр $abc, массив @abc, хеш %abc. В данной главе мы не будем рассматривать этот вид ссылок. Отметим только, что он лежит в основе механизма экспорта/импорта модулей.

(Работа с модулями обсуждается в главе 12.)

Тип ссылки можно определить при помощи встроенной функции refo, которая рассматривает свой аргумент как ссылку и возвращает символическое обозначение ее типа. Если аргумент не является ссылкой, возвращается пустая строка. Встроенные типы обозначаются следующим образом:

- REF ссылка на ссылку;
- SCALAR ссылка на скаляр;
- ARRAY ссылка на массив;
- HASH ссылка на ассоциативный массив;
- CODE ссылка на подпрограмму;
- GLOB ссылка на переменную типа typegiob.

Ссылки Е языке Perl бывают *жесткие* и *символические.* Понятия "жесткая ссылка" и "символическая ссылка" вместе с названиями проникли в Perl из мира UNIX, где они используются применительно к файловой системе. Разберем их применение в UNIX, чтобы лучше понимать, для чего они нужны в Perl.

Каждому файлу в UNIX соответствует *индексный дескриптор* - структура данных, имеющая определенный формат, расположенная в специально отведенной области диска и содержащая важнейшую информацию о файле: тип файла, его расположение на диске, права доступа и т. д. Каждый дескриптор имеет числовой номер, соответствующий его положению в таблице индексных дескрипторов. Этот номер и является внутренним именем файла для операционной системы. Для нее

сущность файла заключается в его индексном дескрипторе, а не в его содержимом. Для пользователя, напротив, важно содержимое файла, а о существовании индексного дескриптора он может даже не подозревать. Кроме того, пользователю удобнее работать с именем файла, а не с числовым номером. Для удобства пользователя создается ссылка на файл, которая ставит в соответствие индексному дескриптору имя файла.

Ссылка представляет собой запись в каталоге, который является тоже файлом, выполняющим специальную функцию регистрации других файлов. В простейшем случае эта запись содержит два поля: имя файла и номер индексного дескриптора. Можно создать несколько ссылок с разными именами в одном или нескольких каталогах, указывающих на один файл. Ссылка указывает на индексный дескриптор, но для краткости говорят о ссылке на файл. Следует подчеркнуть, что все ссылки равноправны. Ссылка, созданная первой, не имеет никакого преимущества перед ссылками, созданными позднее. В индексном дескрипторе среди другой важной информации содержится счетчик ссылок. Удаление из каталога ссылки на файл уменьшает значение счетчика ссылок на единицу. Когда значение счетчика ссылок становится равным нулю, файл удаляется, а его индексный дескриптор освобождается для использования новым файлом.

Рассмотренные ссылки называются *жесткими* ссылками. Кроме них, существуют *символические* ссылки. Символическая ссылка является не просто записью в каталоге, а файлом особого типа, содержащим символьное имя другого файла. В качестве файла символьная ссылка имеет свой индексный дескриптор. Поскольку символическая ссылка является самостоятельным файлом, ее существование никак не отмечается в дескрипторе того файла, на который она указывает. В частности, если удалить все символические ссылки на файл, то это не будет означать его удаление.

Жесткие и символические ссылки в языке Perl напоминают одноименные понятия в файловой системе UNIX. Жесткая ссылка в Perl – это скалярная величина, которая содержит адрес некоторой области памяти, являющейся носителем данных. Сами данные будем называть *субъектом* ссылки. Символическая ссылка – это скалярная переменная, которая содержит имя другой скалярной переменной. "Истинной" ссылкой является жесткая ссылка. Она создается одним из способов, перечисленных в разделе 9.1. Внутренняя организация жестких ссылок такова, что для каждого субъекта ссылки поддерживается счетчик ссылок. Область памяти, занимаемая субъектом ссылки, освобождается, когда значение счетчика ссылок становится равным нулю. В большинстве случаев мы имеем дело с жесткими ссылками, а использование символических ссылок будем специально оговаривать.

Главным применением ссылок в языке Perl является создание сложных структур данных. Мы знаем, что основными типами данных в Perl являются скаляры, массивы и хеш-массивы. Многомерные массивы или более сложные структуры данных, аналогичные записям языка Pascal или структурам языка C, в языке Perl отсутствуют. В более ранних версиях языка отсутствовала и возможность создания сложных структур данных на основе имеющихся типов. Такая возможность появилась в версии Perl 5.0 вместе с появлением ссылок. В практике программирования часто встречаются данные, которые

удобно представлять, например, в виде двумерных массивов, реже трехмерных массивов или других подобных структур.

Двумерный массив можно рассматривать как одномерный массив, элементами которого являются также одномерные массивы. Возможность такого представления есть во многих языках программирования. В языке Perl невозможно создать массив с массивами в качестве элементов. То же самое относится и к хешам. Элементом массива или хеша может быть только скалярная величина. Поскольку ссылка является скалярной величиной, можно создать массив или хеш, элементами которого являются ссылки на массивы или хеши, и таким образом получить структуру, которую можно использовать как массив маесивов (соответственно массив хешей, хеш массивов, хеш хешей). Благодаря ссылкам можно на основе массивов и хешей конструировать структуры данных произвольной сложности.

Помимо создания сложных структур данных, ссылки активно применяются для работы с объектами. Слово "объект" здесь обозначает основное понятие объектно-ориентированного подхода к программированию. *(Объекты рассматриваются в главе 13.)*

В этой главе мы рассмотрим основное применение ссылок как средства для конструирования структур данных. Другие применения будут рассмотрены в соответствующих главах.

Создание ссылок

Существует несколько способов порождения ссылок. Рассмотрим их в порядке следования от чаще употребляемых синтаксических конструкций к более редким.

Операция ссылки "\"

Операция "\", примененная к единственному аргументу, создает ссылку на этот аргумент. В качестве последнего может выступать переменная любого типа или константа. Примеры:

```
$a=\5;
$scal_ref=$a; $arr_ref=\@myarray; $hash_ref=\%myhash; $funC_ref=\ anyfunc;
```

В данном примере скалярной переменной $a присваивается значение ссылки на константу 5, т. е. адрес ячейки памяти, в которой хранится число 5. Адрес самой переменной $a хранится в переменной $scai_ref. Переменные $arr_ref, $hash_ref, $func_ref хранят адреса ячеек памяти, являющихся начальными точками размещения соответственно массива gmyarray, хеш-массива %myhash и кода функции myfunc. К переменным, содержащим ссылки, можно применять все операции допустимые для скалярных величин. Их можно присваивать другим переменным, складывать, умножать, делить, выводить на экран и т. д. За исключением присваивания применение подобных операций к ссылкам, как правило, смысла не имеет. Например, вывод рассмотренных выше переменных

```
print $scal_ref, "\n", $arr_ref,"\n", $hash_ref, "\n", $func_ref, "\n";
```

будет состоять иа строк, подобных следующим:

```
SCALAR(0x9b8994) ARRAY(0x9b8a18) HASH(0x9b8a60) CODE(0x9b3d14)
```

Здесь каждая строка содержит слово, обозначающее тип ссылки и ее значение – адрес в виде шестнадцатеричного числа.

Операция, которую действительно имеет смысл применять к ссылкам, это операция *разыменования,* то есть операция получения того значения, на которое указывает ссылка. Синтаксические конструкции, используемые для разыменования ссылок, мы рассмотрим после того, как обсудим способы их создания.

Конструктор анонимного массива

В рассмотренном выше примере операция "\" применялась к переменным, обладающим именами. Perl позволяет создавать ссылки на анонимные массивы при помощи специальной конструкции, использующей квадратные скобки:

```
$arr_ref = [1,2,3];
```

В результате данной операции присваивания будет создан анонимный массив с элементами (1,2,3), а переменной $arr_ref будет присвоено значение ссылки на этот массив.

Компилятор различает случаи использования квадратных скобок для создания ссылки на анонимный массив и для обращения к отдельным элементам массива, как, например, в операции присваивания $a = $myarray[2].

Замечание

Свободный синтаксис языка Perl допускает существование конструкций, смысл которых не очевиден. К рассматриваемой теме имеет отношение следующий пример. Формально выражение \ ($a, $b, $c) представляет собой анонимный массив из трех элементов ($a, $b, $c), к которому применяется операция ссылки "\". Означает ли это, что значением выражения является ссылка на анонимный массив? Нет, это просто сокращенная запись массива, состоящего из трех элементов-ссылок (\$a, \$b, \$c), а для создания ссылки на анонимный массив существует единственный способ, рассмотренный выше.

Конструктор анонимного ассоциативного массива

По аналогии с массивами можно создавать ссылки на анонимные ассоциативные массивы, используя фигурные скобки. Операция присваивания

```
%hash_ref = {
'One'=>1,
'Two'=>2,
"Three'=>3 };
```

создаст анонимный хеш-массив cone'=>i, 'Two'=>2, "rhree'=>3) и присвоит переменной %hash_ref значение ссылки на этот хеш.

Фигурные скобки используются во многих конструкциях, например, для обращения к индивидуальному элементу хеш-массива

```
$a = $myhash{"first"}
```

или для выделения блока операторов. Обычно такие случаи легко различимы, и их нельзя спутать с порождением ссылки на анонимный хеш. Но иногда возникают неоднозначные ситуации, требующие разрешения. Забегая вперед, приведем пример, связанный с определением функции пользователем.

(Желающие могутЛ предварительно прочитать начало главы 11, в которой рассказывается о подпрограммах и функциях.)

Предположим, что необходимо определить функцию, которая создает анонимный хеш и возвращает ссылку на него. Возвращаемое значение можно задать при помощи встроенной функции return. Если конструкция return отсутствует, то в качестве возвращаемого значения по умолчанию принимается значение последнего выражения, вычисленного внутри функции. Таким образом, синтаксически допустимо следующее определение функции

```
sub get_hash_ref { {@_} }
```

В данном примере внутренняя конструкция в фигурных скобках интерпретируется как блок. Для того чтобы она интерпретировалась как ссылка на анонимный хеш, необходимо использовать функцию return или поставить перед внутренней конструкцией знак "+":

```
sub get_hash_ref { return {(?_} } sub get_hash_ref { +{@_} }
```

Другие способы

В предыдущих разделах рассмотрены основные способы создания ссылок:

- применение операции " \ " к объекту ссылки;
- специальные конструкции [ ] и { }, создающие в определенном контексте ссылку соответственно на анонимный массив и анонимный ассоциативный массив.

Эти способы применяются наиболее часто в тех случаях, когда возникает необходимость в использовании ссылок. Но существуют и другие источники появления ссылок, о которых следует упомянуть для полноты изложения.

9.2.4.1. Конструктор анонимной подпрограммы

Мы уже использовали в примерах подпрограммы, не дожидаясь их систематического изучения. Поэтому можем рассмотреть в этой главе такой вид ссылки, как ссылка на анонимную подпрограмму.

Ссылка на анонимную подпрограмму может быть создана при помощи ключевого слова sub, за которым следует блок – последовательность операторов, заключенная в фигурные скобки:

```
$sub_ref = sub { print "Привет!\n");
```

В результате операции присваивания в переменную $sub_ref заносится адрес, по которому размещается код анонимной подпрограммы. В данном примере подпрограмма состоит из единственного обращения к функции print, выводящей строку "Привет!".

Пример, иллюстрирующий данный вид ссылки, будет рассмотрен далее в этой главе.

9.2.4.2. Ссылка, создаваемая конструктором объекта

В версию 5.0 языка Perl была добавлена поддержка объектно-ориентированного программирования. Основой объектно-ориентированного подхода являются понятия *класс* и *объект.*

(Классы и объекты рассматриваются в главе 13.)

Понятие "объект" реализовано в языке Perl таким образом, что объект становится доступным в программе только через ссылку на него. Для создания объекта используется специальная подпрограмма – *конструктор,* которая, в свою очередь, применяет для этого встроенную функцию bless o. Конструктор возвращает ссылку на объект. Таким образом, это еще один способ порождения ссылок, без которого не обойтись тем, кто использует объектно-ориентированный подход в Perl.

9.2.4.3. Ссылки на данные типа typeglob

Компилятор Perl хранит имена всех переменных программы в *таблице символов.* Отдельная таблица символов существует для каждого *пакета,* образуя собственное пространство имен.

(О пакетах и таблицах символов рассказывается в главе 12.)

Каждый идентификатор, встречающийся в пакете, заносится в таблицу символов. Одинаковые идентификаторы, соответствующие переменным разных типов, образуют гнездо, в котором каждому типу соответствует свой элемент, содержащий адрес переменной данного типа. Если, например, в программе имеются следующие строки

```
$a=5;
@a=(1,2,3,4,5);
%a=("one"=>1,"two"=>2,"three"=>3);
sub a {return "Hello, Mike!";};
```

то таблица символов содержит гнездо для идентификатора "a", состоящее из четырех элементов, хранящих адреса: скалярной переменной $a, массива @a, ассоциативного массива %a и кода подпрограммы &a.

В языке Perl существует внутренний тип данных typeglob. Признаком этого типа является наличие префикса "*" в имени переменной. Тип typeglob служит для ссылки на все переменные разных типов с одинаковыми именами. Например, переменная *a обозначает ссылку на гнездо "a" в таблице символов. Используя специальную запись, можно при помощи переменной typeglob получить ссылки на отдельные элементы гнезда: $scalarref = *a{SCALAR}; # эквивалентно $scalarref = \$a; $arrayref = *a{ARRAY}; # эквивалентно $arrayref = \@a; $hashref = *a{HASH}; # эквивалентно $hashref = \%a; $coderef = *a{CODE}; # эквивалентно $coderef = *&a; $globref = *a{GLOB}; # эквивалентно $globref = *a;

9.2.4.4. Неявное создание ссылок

В рассмотренных случаях осуществляется явное создание ссылки при помощи операции "\" или специальных синтаксических конструкций. В них всегда явным образом определяется скалярная переменная, в которую и заносится значение ссылки. Ссылка может также создаваться неявно в случае, когда операция разыменования применяется к ссылке, ранее не созданной в программе явным образом, и в контексте выражения предполагается, что такая ссылка должна существовать. Возможно, последнее предложение звучит не совсем понятно. Его смысл станет ясным в следующем разделе.

Разыменование ссылок

Разыменованием ссылки называется получение объекта, на который указывает эта ссылка. Для разыменования, как и для создания ссылки, применяются различные синтаксические конструкции, подчас достаточно сложные для визуального восприятия. К ним нужно привыкнуть. Вид конструкции зависит от типа ссылки, к которой применяется разыменование. Рассмотрим их по степени возрастания сложности.

Разыменование простой скалярной переменной

Если ссылка на некоторый объект: скалярную переменную, массив, ассоциативный массив и т. д., является простой скалярной переменной без индексов, то для обращения к самому объекту применяется правило: вместо имени объекта подставить в выражение простую скалярную переменную, содержащую ссылку. Например:

```
1 $a = $$scal_ref;
2 @b = @$arr_ref;
3 %c = %$hash_ref;
4 &f - &$code_ref;
5 $$d[0] = 7;
6 $$h{"one"} = 1;
```

Здесь предполагается, что переменная $scal_ref содержит ссылку на скалярную величину, $arr_ref - ссылку на массив, $hash_ref - ссылку на ассоциативный массив, $code_ref – ссылку на

подпрограмму.

Рассмотрим подробно пятую строку.

Во-первых, следует определить, что является ссылкой: скалярная переменная $d, указывающая на *анонимный* массив, или элемент $d[0] массива @d. Ответ содержится в сформулированном выше правиле разыменования. Поскольку в строке 5 применяется именно оно, то индексированная переменная $d[0] ссылкой быть не может. Ссылкой является простая скалярная переменная $d, которая используется в качестве имени. Из контекста видно, что на ее месте должно стоять имя массива, следовательно, $d является ссылкой на анонимный массив

Во-вторых, здесь мы имеем пример неявного создания ссылки, о котором говорилось в предыдущем разделе. Ссылка $d не была ранее создана явным образом, но ее существование предполагается в операции присваивания. Поэтому компилятор создаст ссылку $d на анонимный массив, поместит в нее адрес массива и по этому адресу сохранит значение первого элемента, равное 7.

Все сказанное можно отнести к шестой строке с единственным отличием: вместо ссылки на анонимный массив здесь фигурирует ссылка $ь на анонимный хеш-массив.

Блоки в операциях разыменования ссылок

Если ссылка является не простой скалярной переменной, а, например, элементом массива или ассоциативного массива, то для ее разыменования нельзя применить правило предыдущего раздела. В этом случае следует заключить ссылку в фигурные скобки и полученный блок использовать в качестве имени переменной в выражениях. Вообще, во всех случаях разыменования ссылок в качестве имени объекта можно использовать блок, результатом выполнения которого является ссылка соответствующего типа.

```
${$d[0]} = 7; ${$h{"one"}} = 1; ${&f()}[l] = 3;
```

Разберем первую строку. Начальный символ $ является признаком скалярной переменной, за которым должно следовать ее имя. Вместо имени используется блок, следовательно, выражение внутри блока интерретируется как ссылка. В данном случае осуществляется разыменование ссылки $d[0], являющейся элементом массива @a. Аналогично, во второй строке осуществляется обращение к скалярной переменной, на которую указывает ссылка $h{"one"}, являющаяся элементом ассоциативного массива %ь. В третьей строке блок, возвращающий ссылку, состоит из одного обращения к функции f (). Ее значение интерпретируется как ссылка на массив, и второму элементу этого массива присваивается значение 3.

Операция разыменования "->"

Применение правила разыменования предыдущего раздела может привести к появлению громоздких выражений, содержащих множество вложенных друг в друга блоков, и очень сложных для визуального восприятия. Непростыми являются уже вышеприведенные примеры. При построении же более сложных

структур выражения становятся почти необозримыми. Даже достаточно простые конструкции требуют определенного усилия для того, чтобы понять, что они означают:

```
$<$a[0]'}[1] = 17; , ${$b[0]}{"one"} = 1;
```

В первой строке осуществляется обращение к отдельному элементу массива массивов, во второй – к отдельному элементу массива хеш-массивов.

> **Замечание**
>
> В действительности речь идет соответственно о массиве, элементами которого являются *ссылки* на анонимные массивы и о массиве, элементами которого являются *ссылки* на анонимные хеш-массивы. Но для краткости в подобных случаях мы будем употреблять сочетания "массив массивов", "массив хеш-массивов" и т. д.

Несколько упростить запись и улучшить наглядность можно, используя операцию *"->"* ("стрелка").

Аргумент слева от стрелки может быть любым выражением, возвращающим ссылку на массив или хеш-массив.

Если левосторонний аргумент является ссылкой на массив, то аргумент справа от стрелки – *индекс,* заключенный в квадратные скобки и определяющий элемент этого массива.

Если левосторонний аргумент является ссылкой на хеш-массив, то аргумент справа от стрелки – значение *ключа,* помещенное в фигурные скобки и определяющее элемент этого хеш-массива.

Результатом операции "->" является соответственно *значение элемента* массива или хеш-массива. Предыдущий пример можно более компактно записать в виде

```
$a[0]->[1] = 17; $b[0]->{"one"} = 1;
```

Конструкция $a[0]->[1] обозначает второй элемент массива, определяемого ссылкой $a[0]. Конструкция $b[0]->{"one"} обозначает элемент, соответствующий ключу "one" хеш-массива, задаваемого ссылкой $b[0].

Вообще, если $arr_ref – ссылка на массив, то $arr_ref->[$i] обозначает i-й элемент этого массива. Если $hash_ref – ссылка на хеш-массив, то $hash_ref->{"key"} обозначает элемент этого хеш-массива, соответствующий ключу "key".

Если бы в последнем примере вместо именованных массивов @a и @b использовались ссылки на массив, например, $ref_a и $ref_b, то соответствующие операции присваивания имели вид

```
$ref_a->[0]->[l] = 17; $ref_b->[0]->{"one"} = 1;
```

Здесь мы снова сталкиваемся с неявным созданием ссылок. По контексту элемент массива $ref_a->[0] должен быть ссылкой на массив, а $ref_b->[0] – ссылкой на хеш-массив. Обе ссылки ранее не были определены, но их существование предполагается в контексте выражения. Данные ссылки будут созданы автоматически.

Операция "->" позволяет для обращения к отдельному элементу составного массива или хеш-массива использовать более простые выражения, например,

```
$a[$i]->[$j]->[$k] вместо ${${$a[$i]}[$j]}[$k], $b[$i]->{"key"}->[$j] вместо ${${$b[0]}{"key"}}[$j]
```

и т. д.

Дальнейшее упрощение связано с тем, что при обращении к элементам гожных структур, представляющих собой комбинации вложенных массивов

и хеш-массивов, можно опустить символы "->" между квадратными и/или фигурными скобками, содержащими индексы или ключи элементов. Предыдущие выражения примут еще более простой вид: $a[$i][$j][.$k] и $b[$i] {"key"} [$j]соответственно.

Символические ссылки

Из предыдущего раздела мы знаем, что если ссылка не определена, но ее присутствие требуется контекстом, то она создается автоматически. Если же определенная ранее скалярная величина не является ссылкой, но используется в качестве ссылки, то ее называют *символической* ссылкой. Значение символической ссылки интерпретируется как имя некоторой переменной. Над этой переменной будут выполняться все операции, применяемые к символической ссылке. Вспомним, что значением жесткой ссылки является адрес. В следующем примере переменная $name_a используется как символическая ссылка на переменную $a.

```
1 $name_a = "a";
2 $$name_a = 17;
3 @$name_a = (1,2,3);
4 $name_a->[3] = 4;
5 %$name_a = ("one"=>l, "two"=>2, "three"=>3);
6 &$name_a ();
```

В строке 2 переменной $a присваивается значение 17. В строке 3 определяется и инициализируется массив @a с элементами (1,2,3). В строке 4 к массиву @a добавляется четвертый элемент со значением 4. В строке *5* инициализируется хеш-массив %a. В строке 6 осуществляется вызов функции a o (предположим, что такая функция существует).

Символическая ссылка может указывать только на переменную, имя которой содержится в таблице символов пакета.

(О пакетах и таблицах символов описано в главе 12.)

Лексические переменные, определяемые при помощи функции ту (), в таблицу символов не входят, поэтому их имена невидимы для механизма, реализующего символические ссылки.

(О лексических переменных и применении функции ту () рассказывается в главе П.) Для иллюстрации рассмотрим следующий пример:

```
$name_a="a"; { my $a="Hello!";
print $$name_a; };
```

Здесь переменная $name_a используется в качестве символической ссылки на переменную $a, и можно предположить, что результатом выполнения этой последовательности будет вывод строки "Hello!". В действительности переменная $a является невидимой для символической ссылки, поскольку она определена как лексическая переменная внутри блока {...}. Поэтому в результате выполнения данного фрагмента будет напечатана пустая строка.

Применение символических ссылок является потенциально опасным из-за возможности возникновения смысловых ошибок. Например, может показаться, что в результате выполнения следующей последовательности операторов

```
1 $a[0]="b";
2 #.............
3 $b[0]=2;
4 $b[1]=2;
5 #.............
6 $a[0] [0]=0;
7 #.............
8 $prod = $b[0]*b[1];
```

переменная $prod получит значение 4. Но это не так. В строке 6 мы осуществляем присваивание, рассчитывая на то, что будет применен известный механизм неявного создания жесткой ссылки $a [0]. Мы "забыли" о том, что значение $a[0] уже использовалось в строке 1 и, следовательно, в строке 6 элемент массива $a[0] является символической ссылкой, указывающей на переменную с именем "Ь". Это имя будет подставлено вместо символической ссылки, в результате чего элемент массива ь[0] получит новое значение 0. В итоге значение переменной $prod будет равно 0.

Во избежание подобных ошибок можно запретить использование символических ссылок в пределах текущего блока при помощи директивы

```
use strict 'refs';
```

Это ограничение, если требуется, можно отменить для внутреннего блока при помощи другой директивы

```
no strict 'refs[1];
```

(Директивы use, по рассматриваются в главе 12.)

Еще одно замечание, касающееся символических ссылок. В версии 5.001 появилась новая возможность: если переменную, являющуюся символической ссылкой, заключить в фигурные скобки, то такая конструкция интер-

претируется не как символическая ссылка, а как значение переменной, подобно тому, как аналогичная конструкция интерпретируется командной оболочкой shell операционной системы UNIX. В следующем фрагменте

```
1 use strict 'refs';
2 ${name};
3 ${"name"};
```

вторая строка представляет собой просто значение переменной $name, а третья строка интерпретируется как символическая ссылка, указывающая на переменную $name и вследствие применения директивы use strict 'refs[1] вызывает сообщение об ошибке вида

```
Can't use string ("name") as a SCALAR ref while "strict refs" in use
```

Использование ссылок

В данном разделе мы рассмотрим некоторые примеры, связанные с основным применением ссылок – конструированием структур данных. В качестве первой структуры построим массив массивов или двумерный массив. Для примера рассмотрим массив @calendar, содержащий календарь, например, на 2000 год. Значением элемента $caiendar[$i] [$j] является название дня недели, приходящегося на *(j+l)-ft* день (i+l)-го месяца, i=(0..11),j=(0..30)(p_{H}с. 9.1).

Рис 9.1. Структура массива @calendar

Замыкания

Для заполнения массива ecaiendar нам потребуется функция, которая по заданному году, месяцу и дню месяца вычисляет соответствующий день недели. Читатель может сам написать свой вариант

функции, может быть, более изящный. Мы предлагаем наш вариант (пример 9.1) с основной целью: рассказать об одном интересном свойстве анонимных подпрограмм. Кроме того, он правильно работает для любого года нашей эры.

Вычисление дня недели основано на том, что:

- 1 января 1 года нашей эры было понедельником;
- каждый год, номер которого делится на 4, является високосным, за исключением тех номеров, которые делятся на 100 и не делятся на 4.

(Вопросы создания функций пользователем рассмотрены в главе 11.)

```
sub GetDay (
my $year = shift;
my @days = (Q,31,59,90,120,151,181,212,243,273,304,334);
my @week = ("Monday","Tuesday","Wednesday","Thursday",
"Friday","Saturday","Sunday"); my $previous_years_days = ($year -1 ).*365 + int (($year-l) /4)
- int(($year-l)/100) + int(($year-l)/400); return sub { my ($month, $day)=@_;
my $n « $previous_years_days + $days[$month-l] + $day -1; $n++ if ($year%4 == 0 and $year%100
!= 0 or
$year%400 == 0 and $month > 2) ; return $week[$n%7]; } };.
```

Аргументами функции GetDay о являются номер года, номер месяца и номер дня месяца. Внутри тела функции им соответствуют переменные $yeag, $month и $day. Функция подсчитывает число дней $п, прошедших с 1 января 1 года. Остаток от деления этого числа на 7 — $п%7 — определяет день недели как элемент массива $week[$n%7].

Необходимые пояснения к тексту

Для передачи параметров в подпрограмму используется предопределенный массив @_. Встроенная функция shift·() без параметров, вызванная внутри подпрограммы, возвращает первый элемент массива @_ и осуществляет сдвиг всего массива влево, так, что первый элемент пропадает, второй становится первым и т.д. Элемент массива $days[$l] равен суммарному числу дней в первых i месяцах не високосного года, i = (0..11). В переменной $previous_years_days запоминается вычисленное значение общего количества дней, прошедших с 1 января 1 года до начала заданного года.

Обратите внимание на то, что значением функции GetDayO является не название дня недели, а ссылка на анонимную функцию, которая возвращает название дня недели. Объясним, зачем это сделано.

Если бы функция GetDay (} возвращала день недели, то для заполнения календаря на 2000 год, к ней необходимо было бы сделать 366 обращений, вычисляя каждый раз значение переменной

$previous_years_days. Для каждого года это значение постоянно, поэтому его достаточно вычислить всего один, а не 366 раз.

На время вычисления функции формируется ее *вычислительное окружение,* включающее совокупность действующих переменных с их значениями. После завершения вычисления функции ее вычислительное окружение пропадает, и на него невозможно сослаться позже. Часто бывает полезным, чтобы функция для продолжения вычислений могла запомнить свое вычислительное окружение. В нашем примере полезно было бы запомнить значение переменной $previous_years_days, чтобы не вычислять его повторно. В языках программирования существует понятие *замыкание,* пришедшее из языка Lisp. Это понятие обозначает совокупность, состоящую из самой функции как описания процесса вычислений и ее вычислительного окружения в момент определения функции.

Анонимные процедуры в Perl обладают тем свойством, что по отношению к лексическим переменным, объявленным при помощи функции ту (), выступают в роли замыканий. Иными словами, если определить анонимную функцию в некоторый момент времени при некоторых значениях лексических переменных, то в дальнейшем при вызове этой функции ей будут доступны значения этих лексических переменных, существовавшие на момент ее определения.

В нашем примере указанное свойство анонимных функций используем следующим образом. Чтобы анонимной функцией можно было воспользоваться в дальнейшем, присвоим ссылку на нее скалярной переменной:

```
$f = GetDay(2000,1,1);
```

Во время обращения к GetDay о было сформировано вычислительное окружение анонимной функции, на которую сейчас указывает переменная $f. Вычислительное окружение включает, в том числе, и переменную $previous_years_days с ее значением. Обратите внимание, что внутри анонимной функции значение этой переменной не вычисляется. В дальнейшем для заполнения календаря мы будем вызывать анонимную функцию через ссылку $f.

Массив массивов

Сформируем массив @caiendar, используя результаты предыдущего раздела.

```
for $i (1,3..12) { . '
for $j (1..30) {
$calendar[$i-l][$j-l] =&$f($i, $j);
}
}; • . for $i (1,3,5,7,8,10,12) {
$calendar[$i-l][30] = &$f($i, 31); }; for $j (1..28) {
$calendar[l][$j-l] = &$f(2, $j); };
# Если год високосный, то добавляется еще один элемент массива $calendar[l][28] = &$f(2,29);
```

Массив @caiendar состоит из 12 элементов по числу месяцев в году. Каждый элемент массива является ссылкой на другой массив, имеющий столько элементов, сколько дней в соответствующем месяце. Значениями элементов вложенных массивов являются английские названия соответствующих дней недели: "Monday", "Tuesday" и т. д.

Обращаем внимание на то, что при формировании массива ^calendar осуществляется неявное создание ссылок $caiendar [$i] и применяется компактная запись $calendar[$i] [$j] для обозначения индивидуального элемента двумерного массива, обсуждавшаяся в разделе 9.3.3.

Содержимое массива @calendar можно вывести для просмотра при помощи следующих операторов:

```
for $i (0..11) {
for $j (0..$#{$calendar[$i]}) {
print $j+l,".",$i+l," is $calendar[$i][$j]\n";
} };
```

Напомним, что запись $#array обозначает верхнее значение индекса массива @array. В результате выполнения данного цикла будет выведена длинная последовательность строк вида

```
1.1 is Saturday 2.1 is Sunday
```

Другие структуры данных

На основе массива Ocaiendar, содержащего календарь на 2000 год, покажем, как можно строить более сложные структуры данных. Структура двумерного массива не очень удобна для представления содержащихся в ней данных в привычном виде настенного календаря. Перегруппируем данные, объединяя их в группы по дням недели. Для этого построим новую структуру, которую для краткости назовем "массив хешей массивов", отдавая себе отчет в том, что такое словосочетание не только далеко не изящно, но и по существу неточно.

Новая структура представляет собой массив @months, состоящий из 12 элементов по числу месяцев в году. Каждый элемент содержит ссылку на анонимный хеш-массив. Каждый вложенный хеш-массив содержит набор ключей, имеющих имена, совпадающие с английскими названиями дней недели: "Monday", "Tuesday" и т. д. Каждому ключу соответствует значение, являющееся, в свою очередь, ссылкой на анонимный массив, содержащий все числа данного месяца, приходящиеся на день недели, соответствующий ключу: все понедельники, все вторники и т. д. Структура массива @months представлена на рис. 9.2.

Рис 9.2. Структура массива @months

```
for $i (0..11) { . for $j (0..$#{$calendar[$i.]}> {
push @{$months[$i]{$calendar[$i][$j]}}, $j+l; } };
```

Замечание

Функция push @array, list помещает список list в конец массива garray.

Первым аргументом встроенной функции push является массив, в который попадают все дни (i+l)-го месяца, приходящиеся на один и тот же день недели: все понедельники, все вторники и т. д. На этот массив указывает ссылка $months[$i] {"key"}, где ключ "key" принимает значения "Monday", "Tuesday" и т. д. Для обращения к самому массиву ссылку следует разымено-вать, заключив в фигурные скобки: @{$months[$i] ("key"}}. Если вместо ключа "key" подставить нужное значение из $caiendar[$i] [$j], то получим аргумент функции push.

Вновь сформированную структуру удобно использовать для вывода календаря в традиционном виде. Последовательность операторов

```
for $i (0..11) {
print "month # ", $i+l, "\n";
for $DayName (keys %{$months[$i]}) {
print " ${DayName}: @{$months[$i]{$DayName}}\n";
} };
```

распечатает календарь в виде

```
month #1
Monday 3 10 17 24 31
Thursday 6 13 20 27
Wednesday 5 12 19 26
Sunday 2 9 16 23 30
Saturday 1 8 15 22 29
Friday 7 14 21 28
Tuesday 4 11 18 25
```

Встроенная функция keys %hash возвращает список всех ключей ассоциативного массива %hash. Вывод ключей осуществляется функцией keys в случайном порядке, поэтому дни недели расположены не в естественной последовательности, а случайным образом.

Для вывода ключей в порядке следования дней недели воспользуемся встроенной функцией сортировки **sort** SUBNAME LIST

Замечание

> Функция sort () сортирует список LIST и возвращает отсортированный список значений. По умолчанию используется обычный лексикографический (словарный) порядок сортировки. Его можно изменить при помощи аргумента SUBNAME, представляющего собой имя подпрограммы. Подпрограмма SUBNAME возвращает целое число, определяющее порядок следования элементов списка. Любая процедура сортировки состоит из последовательности сравнений двух величин. Для того чтобы правильно задать порядок сортировки, надо представить себе SUBNAME как функцию двух аргументов. В данном случае аргументы в подпрограмму SUBNAME передаются не общим для Perl способом - через массив @_, а через переменные *$a* и $b, обозначающие внутри подпрограммы соответственно первый и второй аргумент. Подпрограмму SUBNAME надо составить таким образом, чтобы она возвращала положительное целое, нуль, отрицательное целое, когда при сравнении аргумент $a назначается меньшим аргумента $b, равным аргументу $b, большим аргумента $b соответственно. Для этого внутри подпрограммы удобно использовать операции числового (<=>) и строкового (cmp) сравнения, возвращающие значения -1,0, 1, если первый аргумент соответственно меньше второго, равен второму, больше второго.

Вместо имени подпрограммы в качестве аргумента SUBNAME может использоваться блок, определяющий пррядок сортировки.

Зададим функцию weekOrder, определяющую порядок сортировки

```
sub WeekOrder {
my %week=("Monday"=>0,
"Tuesday"=>1,
"Wednesday"=>2,
"Thursday"=>3,
"Friday"=>4,
"Saturday"=>5,
"Sunday"=>6) ; $week{$a}<=>$week{$b} };
```

Используя функцию sort () с заданным порядком сортировки

```
for $i (0..11) {
print "month # ", $1+1, "\n";
for $DayName (sort WeekOrder keys %{$months[$i]}) { print " $DayName @{$months[$i]
{$DayName}}\n";
} . ' ' };
```

получим структурированный вывод календаря в виде, упорядоченном по месяцам и дням недели:

```
month f 1
Monday 3 10 17 24 31
```

```
Tuesday 4 11 18 25
Wednesday 5 12 19 26
Thursday 6 13 20 27
Friday 7 14 21 28
Saturday 1 8 15 22 29
Sunday 2 9 16 23 30
```

В качестве следующего примера построим на основе массива gmonths новую структуру, которую можно было бы назвать "хеш-массив хеш-массивов массивов", если бы такое название имело право на существование. В действительности, все просто. Речь идет о том, чтобы заменить в массиве @months числовые индексы ключами, совпадающими с названиями месяцев, и таким образом получить ассоциативный массив %months со сложной внутренней структурой (см. рис. 9.3).

Рис 9.3. Ассоциативный массив %months со сложной внутренней структурой

При построении хеш-массива %months воспользуемся вспомогательным хеш-массивом %OrderedMonths, который будем использовать для задания порядка сортировки:

```
# вспомогательный массив %OrderedMonths %OrderedMonths =( "January"=>0,
"February"=>l,
"March"=>2,
"April"=>3,
"May"=>4, "June"=>5, "July"=>6, "August"=>7, "September"=>8, "October"=>9, "November"=>10,
"December"=>ll ); # формирование структуры for $month (sort {$OrderedMonths{$a}
<=>$OrderedMonths{$b}}
keys %OrderedMonths) { $i = $OrderedMonths{$month}; $months{$month}=$months[$ i];' };
# Вывод элементов хеш-массива %months for $month (sort {$OrderedMonths{$a}
<=>$OrderedMonths{$b}}
keys %OrderedMonths) { print "$month\n"; $i = $OrderedMonths{$month); for $DayName (sort
WeekOrder keys %{$months{$month}}) {
print " $DayName @{$months[$i]{$DayName}}\n"; } };
```

В результате выполнения примера 9.3 будет распечатан календарь на 2000 год в виде:

```
January
Monday 3 10 17 24 31
Tuesday 4 11 18 25
Wednesday 5 12 19.26
```

```
Thursday 6 13 20 27
Friday 7 14 21 28
Saturday 1 8 15 22 29
Sunday 2 9 16 23 30
```

Рассмотренные примеры иллюстрируют подход, используемый в Perl для построения сложных структур данных. Читатель может сравнить возможности, предоставляемые языком Perl, с возможностями распространенных языков программирования, таких как Pascal или C. Любая сложная структура в Perl на "верхнем" уровне представляет собой массив или ассоциативный массив, в который вложены ссылки на массивы или хеш-массивы следующего уровня и т. д. В этой иерархии ссылки на массивы и хеш-массивы могут чередоваться в произвольном порядке. При помощи такого подхода средствами Perl можно представить любую структуру C или запись языка Pascal. Perl позволяет с легкостью создавать структуры, которые в других языках создать трудно или невозможно, например, структуру, эквивалентную массиву, состоящему из элементов разных типов:

```
@array = (1, 2 ,3, ("one"=>1, "two"=>2}, \sfunc, 4, 5};
```

Читатель может поупражняться в построении таких структур и открыть для себя новые нюансы применения этого гибкого и мощного подхода.

В заключение несколько слов о *фрагментах* массивов. Для доступа к элементам массива мы имеем специальную нотацию, состоящую из префикса $, имени массива и индекса элемента в квадратных скобках, например, $array[7]. Если здесь вместо индекса поместить список индексов, а префикс $ заменить префиксом @, то такая запись будет обозначать фрагмент массива, состоящий из элементов с индексами из заданного списка. Подобную нотацию можно использовать в выражениях, например,

```
Ssubarrayl = @array[7..12]; @subarray2 = @array[3,5,7];
```

Массив @subarrayi является фрагментом массива garray, состоящим из элементов со значениями индекса от 7 до 12. Массив @subarray2 является фрагментом массива @array, состоящим из элементов со значениями индекса 3, 5 и 7. В первом случае список индексов задан при помощи операции "диапазон", во втором случае - перечислением.

Для многомерного массива понятие "фрагмент" обобщается и означает подмножество элементов, получающееся, если для некоторых индексов из диапазона их изменения выделить список допустимых значений. Для выделения одномерных фрагментов можно воспользоваться приведенной выше нотацией. Например, для выделения из массива @caiendar фрагмента, содержащего календарь на первую неделю апреля, можно использовать запись

```
@april_first_week = @{'$calendar [3] } [0. . 6];
```

Если выделяемый фрагмент является многомерным, то для его обозначения специальной нотации не существует. В этом случае следует сформировать новый массив, являющийся фрагментом исходного

массива. Например, для выделения из массива @caiendar календаря на первый квартал можно воспользоваться циклом

```
for $i (0..2) { .
for $j (0..$#{$calendar[$i]}) {
$quarter.l[$i] [$j] = $ calendar [$i] [$j ] ; } ' };
```

Вопросы для самоконтроля

1. Что такое ссылка?
2. Объясните разницу между жесткой и символической ссылкой.
3. Все ли корректно в следующем фрагменте

   $href = \%hash; $$href[0] = 17;
4. Каким будет значение переменной $b после выполнения следующих операторов:

   $a = 1;

   $b = ref $a;
5. Что обозначает каждое из выражений:

   $$a[0]; ${$a[0]}; $a->[0]; '$a[0];
6. Приведите пример неявного создания ссылки.
7. $arr_ref — ссылка на анонимный массив. Как с ее помощью обратиться к третьему элементу этого массива? Напишите выражение.
8. Что такое "замыкание"?

Упражнения

1. Добавьте текст, содержащий последовательность операций, которые надо применить к переменной $b, чтобы получить значение переменной

```
$a = 7;
$b = \\\\$a;
```

В упражнениях 2-4 используйте результаты, полученные в примерах 9.1-9.3.

2. Вывести на экран все дни 2000 года, приходящиеся на воскресенья. Вывод должен содержать строку-заголовок, например, "All 2000[1] Sundays are: **", И ПО ОДНОЙ строке на каждый месяц года В виде:** <название месяца^ <дни месяцах

3. Вывести на экран календарь на второй квартал года в виде

```
<название месяца> <дни месяца> <дни месяца>
```

4. Вывести на экран календарь на первую неделю любого месяца. Вывод должен содержать строку-заголовок и по одной строке на каждый день недели в виде

```
<название месяца> <день месяца> <название дня недели>
```

5. Треугольником Паскаля называется следующая бесконечная таблица чисел:

Рис 9.4. Треугольник Паскаля

Каждое число в этой таблице равно сумме двух чисел, стоящих над ним слева и справа. Предложите структуру данных для хранения первых строк треугольника Паскаля. Напишите программу, заполняющую первые 32 строки и выводящую их на печать.