Глава 3. Типы данных

Учебник по Perl

Содержание [скрыть]

Приступая к знакомству с любым языком программирования, мы прежде всего выясняем, а какие типы данных он позволяет обрабатывать, предоставляет ли язык механизм создания новых типов из уже существующих. Ведь язык программирования, как знаковая система обработки информации, как раз и предназначен для обработки информации, представленной данными и алгоритмами. Данные определяются своим *типом* – множеством значений и набором допустимых операций. Одни языки программирования предлагают большое число разнообразных типов данных, как, например, универсальный язык C, в котором даже тип данных, используемый для хранения символов, может быть со знаком или без знака; другие могут обходиться всего лишь двумя типами данных, как, например, VBScript, в котором для хранения и обработки любых скалярных данных (числовых, строковых и булевых) используется единственный вариантный тип данных и существует возможность создания массивов скалярных данных. В конечном счете, язык должен иметь такое разнообразие типов данных, чтобы программист мог с их помощью решать задачи, для которых предназначен язык программирования.

Язык Perl для решения своих задач предлагает всего три типа данных: скаляры, массивы скаляров и ассоциативные массивы скаляров, или хеши. В соответствии с допустимыми типами данных существует и три типа переменных, в которых можно хранить данные перечисленных типов. В этой главе определяются все допустимые в языке типы данных, вводятся числовые и строковые литералы, конструкторы массивов и ассоциативных массивов, а также обсуждаются переменные и их использование, но начнем мы наше изучение с вопроса, без которого невозможно вхождение в любой язык программирования – набора допустимых символов, или алфавита языка.

Алфавит языка

С чего начинается изучение любого языка? Конечно, с алфавита. Помните, как в первом классе мы усердно выводили в прописях все буквы русского алфавита, а в старших классах, приступая к

изучению иностранного языка, мы сначала учили буквы его алфавита и какие звуки они обозначают, а потом из букв складывали слова, которые уже можно было использовать для составления предложений, несущих определенную информативность: "Мама мыла раму".

Аналогично происходит и при изучении языков программирования. Сначала мы должны выяснить, какие можно использовать символы для составления лексем (слов языка), из которых можно конструировать операторы (предложения языка).

В языке Perl можно использовать все буквы латинского алфавита (прописные и строчные), арабские цифры и знак подчеркивания "_". Perl относится к языкам, чувствительным к регистру. Это означает, что символы прописной и строчной буквы считаются *различными*. Поэтому, например, два идентификатора One и one, используемые для задания имени переменной, – это два различных идентификатора и следовательно, определяемые ими переменные одного и того же типа также являются различными.

Замечание

Буквы национальных алфавитов, в частности русского, можно использовать только в строковых данных. Идентификаторы переменных могут содержать буквы только латинского алфавита. Кроме букв, цифр и символа подчеркивания, которые называются *алфавитно-цифровыми* символами, используется набор специальных символов, представленный в табл. 3.1.

Таблица 3.1. *Специальные символы языка Perl*

Символ	Название	Символ	Название
`	Обратный апостроф	~	Тильда
!	Восклицательный знак	@	Коммерческое АТ
№	Номер	$	Знак доллара
%	Процент	^	"Крышка"
&	Амперсанд	*	Звездочка
-	Минус	+	Плюс
=	Равенство	|	Вертикальная черта
'	Прямой апостроф	"	Кавычки
<	Меньше	>	Больше
/	Косая черта	\	Обратная косая черта

,	Запятая	.	Точка
:	Двоеточие	;	Точка с запятой
[	Квадратная левая скобка	]	Квадратная правая скобка
{	Фигурная левая скобка	}	Фигурная правая скобка
(	Круглая левая скобка	)	Круглая правая скобка
?	Вопросительный знак		Пробел

Анализ специальных символов показывает, что в языке Perl используются *все* символы, которые можно ввести с клавиатуры. Если буквенно-цифровые символы используются в составе идентификаторов, то специальные символы служат для определения знаков операций, уточнения синтаксиса выражений, а также именования специальных встроенных переменных языка Perl. Если читателю какой-то из перечисленных объектов сейчас и не совсем ясен, то при последующем изложении все встанет на свои места. Алфавит языка используется для создания "правильных" (распознаваемых интерпретатором языка) лексем. Среди всего множества таких лексем существует подмножество предопределенных лексем, называемых *ключевыми словами* и используемых для создания правильных конструкций языка.

Набор ключевых слов языка Perl не велик и представлен ниже:

```
if, elseif, else, unless, while, until, foreach, for, next, continue, last, do, eval, goto, sub, my, return
```

Кроме перечисленных ключевых слов, определяющих синтаксические языковые конструкции, в языке Perl сложился набор стандартных функций, который реализован в любом интерпретаторе Perl. Имена этих функций можно тоже считать зарезервированными словами языка и не использовать в качестве имен пользовательских функций или меток в программе. Мы не будем здесь перечислять имена всех стандартных функций, так как их количество достаточно велико, да и мало прока будет от такого простого перечисления, а отошлем читателя к приложению А, где он может увидеть имена всех стандартных функций.

Скалярный тип данных

Скалярный тип данных в Perl предназначен для представления и обработки числовых данных (чисел) и последовательности символов, называемых строками. Для задания в программе перечисленных данных используются буквальные константы, или литералы: числовые и строковые.

Числовые литералы используются для представления обычных чисел, необходимых для реализации какого-либо алгоритма в программе Perl. Обычно используются числа с основанием десять, или десятичные числа, но язык позволяет использовать и восьмеричные (с основанием восемь), и

шестнадцатеричные (с основанием шестнадцать) числа, которые полезны при работе с содержимым памяти компьютера в процессе решения некоторых системных задач.

Десятичные числа могут быть целыми или вещественными с дробной частью, которые в программировании часто называют числами с плавающей точкой из-за способа их представления и хранения в памяти компьютера. Соответствующие им литералы ничем не отличаются от записи подобных чисел в математике: последовательность цифр без пробелов для целых чисел и последовательность цифр, в которой точка отделяет целую часть от дробной, для вещественных чисел (пример 3.1).

Пример 3.1. Числовые литералы.

```
123                 # Целое десятичное число.

 234.89           # Вещественное число.

0.6780             # Вещественное с нулевой целой частью

678                  # Незначащие нули можно не задавать

1_000_000.67 # Для отделения разрядов в целой части числа

                     # можно использовать символ подчеркивания.
```

Для вещественных чисел с плавающей точкой можно использовать и экспоненциальную форму записи:

```
[цифры] . [цифры] [E | e] [+1 -] [цифры]
```

Эта форма записи означает, что для получения значения числа следует его мантиссу, заданную в форме действительного числа с точкой ([цифры]. [цифры]), умножить на десять в степени числа со знаком, заданного в его экспоненциальной части после символа E или e (пример 3.2).

Пример 3.2. Экспоненциальная форма записи вещественных чисел

```
10.67E56 # Знак "+" в экспоненте можно опускать.

10.67e+06 # Так экспонента лучше читаема.

1e-203 # Число близко к машинному нулю.

1e+308 # Число близко к бесконечно большому числу.
```

Замечание

Интерпретатор языка Perl представляет все числа (и целые, и вещественные) в формате чисел с плавающей точкой удвоенной точности. Это означает, что реально нельзя задать больше шестнадцати значащих цифр мантиссы, а экспонента ограничена диапазоном от -323 до +308. Интерпретатор не сгенерирует ошибки, если мантисса будет превосходить 16 цифр, а экспонента 3 цифры, но при отображении таких чисел мантисса будет приведена к шестнадцати значащим цифрам. Если экспонента меньше нижнего предела, то будет выводиться нуль, а если больше верхнего предела, то используется специальный символ 1.#INF, обозначающий бесконечно большое число. Подобный алгоритм представления очень больших и очень маленьких чисел не приводит к возникновению, соответственно, ошибок переполнения и исчезновения порядка, свойственной многим языкам программирования. Если задать целое число с числом значащих цифр больше 15, то при выводе оно будет отображаться как вещественное в экспоненциальной форме.

Некоторые системные установки или анализ некоторых системных параметров легче осуществлять с использованием чисел, представленных в восьмеричной или шестнадцатеричной системах счисления. Форма записи таких чисел аналогична их синтаксису в языке C: любое целое число, начинающееся с нуля "0", трактуется интерпретатором как восьмеричное целое число, а символы, непосредственно следующие за комбинацией "0x", рассматриваются как шестнадцатеричные цифры. При использовании восьмеричных чисел следует помнить, что в них не может быть цифры больше, чем 7, а в шестнадцатеричных числах кроме десяти цифр от 0 до 9 используются буквы A или a, B или b, C или c, D или d, E или e, F или f для обозначения недостающих цифр числа (пример 3.3).

Пример 3.3. Восьмиричные и шестнадцатиричные числа

```
010 # Восьмеричное 10, равное десятичному 8.

0x10 # Шестнадцатеричное 10, равное десятичному 16.

0239 # Вызовет ошибку интерпретации: нельзя использовать цифру 9.

0xAIFf # Соответствует 41477 десятичному.

0xGA # Вызовет ошибку интерпретации: нельзя использовать букву G.
```

Замечание

Задание шестнадцатеричных цифр — это единственный случай в Perl, когда прописные и строчные буквы идентичны. В других случаях их употребления, например в идентификаторах, они различны.

Внимание

Нельзя вместо последовательности символов "0x", идентифицирующей шестнадцатеричные числа, использовать последовательность "ox".

Строковые литералы, или просто *строки,* представляют последовательность символов, заключенную в одинарные ('), двойные (") или обратные (') ка вычки, которая рассматривается как единое целое. Использование одинарных и двойных кавычек для задания строк аналогично их применению для этих же целей в системе UNIX.

В строке, ограниченной одинарными кавычками, нельзя использовать ESC-, или управляющие последовательности, а также в нее нельзя подставить значение переменной. Единственное исключение составляют две управляющие последовательности: (V) и (\\). Первая используется для отображения одинарной кавычки в самой строке, так как иначе интерпретатор рассматривал бы первую, встретившуюся ему одинарную кавычку как признак завершения строки, что не соответствовало бы ее включению в строку. Вторая последовательность используется для отображения самой обратной косой черты. Примеры задания строковых литералов, ограниченных одинарными кавычками, можно найти в табл. 3.2.

Таблица 3.2. *Символьные литералы, ограниченные одинарными кавычками*

Строка	Отображение	Комментарий
'Простая строка #1'	Простая строка #1	Строка без управляющих последовательностей
'VPerl.exeV '	'Perl.exe'	Строка с одинарными кавычками
'D: \\Perl.exe'	D: \Perl . exe	Строка с обратной дробной чертой
'Последовательность \n'	Последовательность \n	Управляющая последовательность \n не влияет на отображение строки
'Завтрак Бутерброд с ветчиной Чашка кофе '	Завтрак Бутерброд с ветчиной Чашка кофе	Многострочный символьный литерал отображается в нескольких строках

Замечание

Esc-последовательности, состоящие из обратной, косой черты (\), за которой следует буква или комбинация цифр. В них символ обратной косой черты рассматривается как символ, изменяющий значение буквы. Они вместе являются одним целым и выполняют определенное действие при выводе на устройство отображения, например, переход на новую строку (\n). Комбинация цифр трактуется как ASCII-код отображаемого символа. Название таких

последовательностей происходит от английского слова "escape", означающего изменять смысл. Их еще называют *управляющие последовательности.*

Строковый литерал может распространяться на несколько строк программы (см. последний литерал табл. 3.2). Для этого при его вводе с клавиатуры следует использовать клавишу <Enter> для перехода на новую строку.

Многострочные литералы отображаются на стольких строках, на скольких они заданы. Это означает, что символ перехода на новую строку, введенный с клавиатуры, сохраняется в символьном литерале, ограниченном одинарными кавычками. Следует заметить, что это справедливо и для строковых литералов, ограниченных двойными кавычками.

Строки в двойных кавычках позволяют вставлять и интерпретировать управляющие последовательности, а также осуществлять подстановку значений переменных, содержащих скаляры или списки. Управляющие последовательности (табл. 3.3) при выводе строк могут интерпретироваться как символы новой строки, табуляции и т. п., а могут изменять регистр следующих за ними букв.

Таблица 3.3. *Управляющие последовательности*

Управляющая последовательность	Значение
\a	Звонок
\b	Возврат на шаг
\e	Символ ESC
\f	Перевод формата
\n	Переход на новую строку
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\$	Знак доллара
\@	Амперсанд или АТ коммерческое
\0nnn	Восьмеричный код символа
\xnn	Шестнадцатеричный код символа

\cn	Эмулирует нажатие комбинации клавиш +. Например, \cC соответствует +
\l	Переводит следующий символ в нижний регистр
\u	Переводит следующий символ в верхний регистр
\L	Переводит следующую за ней последовательность символов, ограниченную управляющей последовательностью \E , в нижний регистр
\Q	В следующей за ней последовательности символов, ограниченной управляющей последовательностью \E , перед каждым не алфавитно-цифровым символом вставляет обратную дробную черту
\U	Переводит следующую за ней последовательность символов, ограниченную управляющей последовательностью \E , в верхний регистр
\E	Ограничивает действие управляющих последовательностей \L, \Q И \U
\\	Символ обратной дробной черты
\"	Двойные кавычки
\'	Одинарные кавычки

Замечание

Если после обратной косой черты в строковом литерале, ограниченном двойными кавычками, следует символ, который не составляет с ней управляющую последовательность, то обратная косая черта не отображается при выводе строки на устройство отображения.

Строковые литералы в двойных кавычках удобно использовать для структурированного вывода текстовой информации, заданной одной строкой. Примеры строк в двойных кавычках представлены в табл. 3.4.

Таблица 3.4. *Символьные литералы, ограниченные двойными кавычками*

Строка	Отображение	Комментарий
"'\Uline\E #1"	LINE #1	Управляющие последовательности перевода регистра \l, \u, \L и \и действуют только на буквы латинского

		алфавита и не применимы к буквам русского алфавита
"Конец страницы\f"	Конец страницы	При выводе на экран монитора или в файл в конце строки отображается символ перехода на новую страницу; при выводе на принтер печать начинается с новой страницы после вывода этой строки
" \tЗавтрак\nБутерброд с ветчиной\nЧашка кофе\n"	Завтрак Бутерброд с ветчиной Чашка кофе	Символьный литерал задан одной строкой с управляющими символами

Последней разновидностью строковых литералов являются строки в обратных кавычках, которые, по существу, не являются строками данных в том смысле, что содержащиеся в них символы не обрабатываются при выводе интерпретатором языка Perl как некоторый поток отображаемых символов. Встретив строку в обратных кавычках, интерпретатор передает ее на обработку операционной системе, под управлением которой он функционирует: Windows, UNIX или какая-либо другая, которая выполняет переданную ей команду и возвращает в программу Perl результаты ее выполнения в виде строки, которую в дальнейшем можно использовать для организации вычислений.

Таким образом, строки в обратных кавычках должны содержать значимую для операционной системы последовательность символов: команду операционной системы, строку загрузки приложения и т. п. Например, при выводе строки ~dir~ оператором print отобразится не слово "dir", а результат выполнения команды dir операционной системы. В системе Windows эта команда отобразит содержимое текущей папки (пример 3.4).

```
Том в устройстве D не имеет метки Серийный номер тома: 1F66-19F2 Содержимое каталога
D:\PerlOurBook
<КАТАЛОГ> 09.01.00 16:01 .
<КАТАЛОГ> 09.01.00 16:01 ..
EXAMPLE PL 32 23.01.00 11:56 exarrple.pl
01 <КАТАЛОГ> 11.01.00 14:12 01
02 <КАТАЛОГ> 11.01.00 14:12 02
03 <КАТАЛОГ> 11.01.00 14:12 03
PerlINF TXT 1 781 12.01.00 11:39 Perlinf.txt
EXAMPLE1 PL 347 18.01.00 18:02 examplel.pl
3 файл(а,ов) 2 160 байт
5 каталог(а,ов) 78 086 144 байт свободно
```

В Perl, как и в UNIX, строки в обратных кавычках используются для "ввода" в программу результатов выполнения не только системных команд, но и выводимых на экран монитора результатов выполнения другой программы, так как всегда можно передать на выполнение командной оболочке имя загружаемого модуля программы.

Замечание

Некоторые символы (их еще называют метасимволы) имеют специальное значение для командной оболочки. К ним относятся *,- <, >, ?, | и &. В системе UNIX, чтобы изменить интерпретацию метасимвола как символа со специальным значением, ставят перед ним обратную косую черту, которая изменяет (escape) его специальное назначение. Теперь он рассматривается командной оболочкой просто как символ, представляющий самого себя. Если во вводимой строке много таких специальных символов, то пользователю необходимо перед каждым поставить обратную косую черту, что приводит к плохой читаемости всей строки. Во избежание подобных "затруднений" в UNIX используются строки в одинарных кавычках, в которых все символы интерпретируются так, как они есть. Подобную же функцию одинарные кавычки выполняют и в языке Perl.

При работе в UNIX широко используется подстановка значений переменных в строку команды, которая передается на обработку оболочке shell. При задании команды в строке ввода используются строки в двойных кавычках, которые так же, как и строки в одинарных кавычках, отменяют специальные значения метасимволов, за исключением символа $, который используется для подстановки значения переменной. Обратная косая черта перед ним изменяет его специальное значение. Именно этот механизм строк в двойных кавычках послужил прототипом для аналогичных конструкций в языке Perl.

Строка в обратных кавычках используется в UNIX для подстановки стандартного вывода команды, т. е. содержимое строки в обратных кавычках интерпретируется командной оболочкой как команда системы, которая должна быть выполнена, а результат ее выполнения подставлен вместо строки в обратных кавычках. В Perl эта конструкция перенесена без всяких изменений.

Все обрабатываемые программой данные хранятся в некоторой области памяти компьютера, определяемой своим адресом. Для удобства программирования доступа к данным языки высокого уровня, и Perl здесь не исключение, используют переменные, с помощью которых программист может ссылаться на данные, расположенные в памяти, или изменять их содержание. Переменная задается своим именем, которое используется программой для обращения к области памяти и извлечения хранящихся в ней данных или, наоборот, записи данных в область памяти. Обычно говорят, что в переменных хранятся данные, хотя, как мы видим, это не совсем так. Правильнее говорить, что переменная определяет именованную область памяти, в которой хранятся некоторые данные.

Более того, переменная определяет тип данных, хранимых в области памяти, на которую она ссылается. В большинстве языков программирования переменные до их использования в программе

объявляются как переменные определенного типа, информируя транслятор, что в них можно хранить данные соответствующего типа. Как мы помним, в Perl нет операторов объявления переменных определенного типа; они автоматически объявляются при первом их использовании в конструкциях языка, например в операторе присваивания значения переменной. Любая переменная определяется заданием своего имени, которое представляет собой правильный идентификатор языка. В Perl имя любой переменной состоит из специального символа (префикса), определяющего тип переменной, за которым следует идентификатор. Для переменных скалярного типа (пример 3.5), или просто скалярных переменных, таким определяющим символом является знак доллара "$".

```
# Правильные имена скалярных переменных. $Name; $name_surname; $name_l;
# Неправильные имена скалярных переменных.
$l_name; # Идентификатор не может начинаться с цифры.
$Name@Surname; # Недопустимый символ @
```

Скалярная переменная может хранить только одно скалярное данное: числовое или строковое, причем не существует никакого способа определить, каг кой именно тип скалярных данных в ней содержится. Дело в том, что при использовании этих переменных в различных операциях хранимые в них данные автоматически преобразуются из одного типа в другой, т. е. в арифметических операциях строка преобразуется в число, а в строковых операциях числовое значение преобразуется в строковое. Преобразование строки в числовое значение осуществляется, если она содержит последовательность символов, которая интерпретируется как число, иначе интерпретатор генерирует ошибку. Шестнадцатеричные числа с префиксом "ох" и десятичные числа с символом подчеркивания для отделения триад в целой части числа, заданные как строки, не преобразуются в числа, а последовательность цифр,' начинающаяся с о, не интерпретируется как восьмеричное число.

Для преобразования строк, содержащих представления шестнадцатеричных и восьмеричных чисел, в числовые значения следует пользоваться функцией oct. Как отмечалось выше, строки в двойных кавычках не только интерпретируют управляющие символы, но и позволяют подставлять значения скалярных переменных. Это означает, что в строке можно задать имя переменной, ко-торое при вычислениях заменяется значением, содержащимся в переменной на момент вычислений. (Подобную процедуру подстановки значения переменной в символьную строку в дальнейшем для краткости мы будем называть просто подстановкой переменной.) Например, следующая последовательность операторов

```
$s = "\$10";
$n = "Книга стоит $s долларов.";
print $n;
```

отобразит на экране монитора строку

```
Книга стоит $10 долларов.
```

Замечание

Можно подставлять значения не только скалярных переменных, но и массивов скаляров, элементов массивов и хешей, а также сечений массивов и хешей. Об этом мы расскажем в следующих параграфах этой главы, определяя соответствующие типы данных и их переменные.

При подстановке переменной ее имя должно быть отделено разделителями от остальных символов строки, причем это правило не обязательно для первого символа имени переменной, так как интерпретатор, встретив символ "$" в строке, ограниченной двойными кавычками, начинает выделять правильный идентификатор.

Разделителями могут быть пробелы или управляющие последовательности. Можно и явно указать идентификатор переменной, задав его в фигурных скобках. Подобная техника демонстрируется следующим фрагментом программы:

```
$day = 'пятницу';
$number = 5;
$html = "HTML";
$s = "${html}-документ отослан B\n$day\t$number февраля.";
print $s;
```

Результатом выполнения этого фрагмента будет отображение двух строк на экране монитора:

```
HTML-документ отослан в пятницу 5 февраля.
```

Переменная $html подставляется с явным указанием ее идентификатора, для выделения идентификаторов остальных переменных используются разделители.

Подставлять можно скалярные переменные, значения которых определены с помощью числовых и любых типов строковых литералов, причем строка в обратных кавычках интерпретируется как команда операционной системы.

Замечание

Рассмотренные в этом параграфе различные типы кавычек для задания строковых литералов на самом деле являются всего лишь удобной формой записи операций языка Perl: q//, qq/7, qx/7. *(Эти операции подробно будут рассмотрены в главе 4).*

Синтаксический анализатор языка Perl при анализе текста программы выделяет слова (не заключенная в кавычки последовательность алфавитно-цифровых символов) и определяет их принадлежность набору ключевых слов. Если слово не является ключевым, то интерпретатор рассматривает его как строку символов, заключенную в кавычки. Это позволяет задавать строковые литералы, не заключая их в кавычки:

```
$day = Friday; # Тождественно оператору $day = 'Friday';
```

Такие слова без кавычек в тексте программы иногда еще называют простыми словами (barewords).

Задание строковых литералов без кавычек возможно только для литералов, содержащих буквы латинского алфавита. Попытка применить подобную технику для литералов, содержащих буквы русского алфавита, приведет к ошибке компиляции.

Завершая разговор о литералах, следует упомянуть о специальных литералах языка Perl: _LINE_, _FILE_, _END_ и _DATA_. Они являются самостоятельными лексемами, а не переменными, поэтому их нельзя вставлять в строки. Литерал _LINE_ представляет номер текущей строки текста программы, а _FILE_– имя файла программы. Литерал _END_ используется для указания логического конца программы. Информация, расположенная в файле программы после этого литерала, не обрабатывается интерпретатором, но может быть прочитана через файл с дескриптором DATA. Последний литерал _DATA_ аналогичен литералу _END_, только дополнительно он открывает файл с дескриптором DATA для чтения информации, находящейся в файле программы после него. Программа примера 3.6 демонстрирует использование специальных литералов.

```
#! /per!520/bin/Perl -w
$file = _FILE_;
$prog = _LINE_;
print "Находимся в строке: $prog\n",
"Файл: $file";; _END_ print "Текст после лексемы _END_";
```

Результат работы этой программы будет следующим, если файл программы хранится в файле D:\PerlEx\examplel.exe:

```
Находимся в строке: 3
Файл: D:\PerlEX\EXAMPLE1.PL
```

Вывода последнего в программе оператора печати не наблюдается, так как он расположен после лексемы _END_.

Массивы скаляров

Массив, в отличие от скалярного типа данных, представляющего единственное значение, – это тип данных, предназначенный для хранения и обработки нескольких скалярных данных, ссылаться на которые можно с помощью индекса. Все массивы Perl одномерны и их можно мыслить как некий линейный список скаляров. Для извлечения какого-либо значения, хранимого в массиве, достаточно одного индекса. В программе массив задается с помощью специальной синтаксической конструкции языка Perl, называемой конструктором массива. Он представляет собой список скалярных значений, заключенный в круглые скобки. Элементы списка отделяются друг от друга запятыми и представляют элементы массива:

```
(скаляр_1, скаляр_2, ... , скаляр_n)
```

Как и в других языках программирования, массив Perl представляет набор однотипных данных – скаляров, но скалярные данные могут быть как числовыми, так и строковыми, поэтому, с точки зрения других языков программирования, в массивах Perl хранятся смешанные данные – числа и строки. В качестве скаляра в конструкторе массива может использоваться числовой или строковый литерал или скалярная переменная, чье значение и будет являться элементом массива:

```
(5, "умножить на", 4) # Используются только литералы.
($first, 'равно', $second) # Используются значения скалярных переменных.
```

Массив может состоять из неограниченного числа элементов, а может не иметь ни одного. В таком случае его называют пустым массивом. Конструктор пустого массива – круглые скобки без содержащегося в них списка ().

Как отмечалось в начале параграфа, массив характеризуется тем, что к любому его элементу можно обратиться при помощи индекса (целого литерала или скалярной переменной, принимающей целое значение), который задается в квадратных скобках непосредственно после конструктора массива. Индексы массивов в языке Perl начинаются с 0, а отрицательные индексы позволяют обратиться к элементам в обратном их заданию порядке:

```
(5, "умножить на", 4)[0] # Первый элемент массива: 5.
(5, "умножить на", 4)[2] # Последний элемент массива: 4.
(5, "умножить на", 4)[-1] f Последний элемент массива: 4.
(5, "умножить на", 4) [-3] # Первый элемент массива: 5.
```

При использовании отрицательных индексов индекс -1 соответствует последнему элементу массива, а индекс -n, где n – количество элементов массива, первому элементу массива.

Нельзя использовать индекс для извлечения элементов списка, возвращаемого некоторыми функциями Perl. Индекс можно применять только к массивам или их конструкторам, а для этого достаточно заключить список в круглые скобки. Например, конструкция stat(@m) [0] вызовет ошибку компиляции, если возвращаемым значением функции stat является список, тогда как конструкция (stat (@m)) [0] приведет к желаемому эффекту.

В качестве элемента списка в конструкторе массива можно использовать конструктор массива, но мы не получим в этом случае, как ожидает читатель, знакомый с другими языками программирования, массив массивов, т. е. многомерный массив, к элементам которого можно обратиться с помощью нескольких индексов. Дело в том, что если в качестве элемента списка в конструкторе массива используется конструктор массива, то его элементы добавляются к элементам массива, определяемого внешним конструктором, т. е. каждый его элемент становится элементом внешнего

массива, увеличивая количество его элементов на количество элементов внутреннего массива. Например, массив

```
(1, (2, 3), 4}
```

эквивалентен массиву

```
(1, 2, 3, 4)
```

(О реализации многомерных массивов при помощи ссылок см. в части 9.)

В программе массивы хранятся в специальных переменных, имена которых начинаются с символа "@". Объявление переменной массива, или просто массива, чаще всего осуществляется в операторе присваивания (=), в правой части которого обычно задается конструктор массива:

```
Sarray = ($ml, '+', $m2, '=', $ml+$m2);
```

В этом примере также продемонстрировано, что в конструкторе массива можно использовать выражения, о которых речь пойдет в следующей главе.

Задать или получить значения элементов массива, хранящегося в переменной, можно и с помощью индекса. Однако "операцию" индексирования нельзя применять непосредственно к имени переменной массива, ее следует применять к переменной, в которой первый символ заменен на символ скалярной переменной $. Подобное "неудобство" связано с последовательным проведением в Perl использования первого символа переменной для указания ее типа: ведь элемент массива является ничем иным как скаляром. Примеры использования индекса с переменными массива представлены ниже:

```
$ml[0] = "Первый"; # Задает первый элемент массива @ml.
$ml[l] = "Второй"; # Задает второй элемент массива @ml.
@m2 = (1, 2, 3, 4, 5); # Задание массива @т2.
print @ml, "\n", $m2[0]; # Печать массива toil и
# первого элемента массива @m2.
```

Массивы в Perl являются динамическими. Добавление нового элемента в массив автоматически увеличивает его размер. С помощью индекса можно добавить элемент, отстоящий от последнего определенного элемента массива на любое число элементов, и это действие не приведет к ошибке. Просто все промежуточные элементы будут не определены (их значение будет равно пустой строке ""). При такой реализации массивов программисту не надо заботиться, как в других языках программирования, что индекс выйдет за размеры массива. В случае обращения к не существующему элементу массива Perl возвращает значение, равное нулевой строке.

Замечание

> Если для интерпретатора Perl включен режим отображения предупреждающих сообщений во время выполнения, то в случае обращения к не определенному или не существующему элементу массива будет отображаться сообщение, что значение элемента не определено.

В любой момент можно определить число элементов массива. Для этого следует воспользоваться синтаксической конструкцией

```
$ #имя_массива
```

которая возвращает максимальный индекс массива или -1, если массив не определен. Чтобы получить количество элементов массива, следует к возвращаемому значению этой конструкции добавить 1, так как индексы массивов в Perl начинаются со.

При работе с массивами самой "утомительной" процедурой может оказаться процедура присваивания значений элементам массива. Хорошо, если все необходимые данные для заполнения массива большой размерности уже существуют в каком-либо текстовом файле. Тогда можно присвоить значения элементам массива, воспользовавшись операцией чтения из внешнего файла, но об этом в следующей главе. А если необходимо в программе создать массив натуральных чисел до 1000 включительно? Неужели надо писать 1000 операторов присваивания, или организовывать цикл, в котором осуществлять соответствующие присваивания, или создавать текстовый файл, содержащий эти числа? К счастью, нет! В Perl для подобных ситуаций предусмотрена операция диапазон, которую можно использовать в конструкторе массива. Например, чтобы создать массив натуральных чисел до 1000, достаточно одного оператора:

```
@naturalNumbers = (1..1000);
```

Общий синтаксис операции диапазон следующий:

```
первое_число..последнее_число
```

Она определяет набор чисел (целых или вещественных), начинающихся с первое_число и не превосходящих последнее_число, в котором последующее больше предыдущего на единицу. Эта операция действительна не только для чисел, но и для алфавитно-цифровых строк, но в этом случае увеличение на единицу означает увеличение на единицу ASCII-кода символа строки. Ниже показаны примеры использования операции диапазон со строковыми данными:

```
"a".."c" i Соответствует: "a", "b", "c"
"BCY".."BDB" # Соответствует: "BCY", "BCZ", "BDA", "BOB"
```

Эта операция особенно удобна для задания целых чисел, начинающихся с нуля, например, для представления дня месяца в дате вида "01.02.2000":

```
@ day_ofjnonth = ("01".."31");
```

(Подробнее операция диапазон рассматривается в части 4.)

Переменные массива и элемент массива можно подставлять в строки, ограниченные двойными кавычками. Если интерпретатор встречает в строке переменную массива, то он вставляет в нее значения элементов массива, отделенные друг от друга пробелами. Результатом выполнения следующих операторов

```
@т = (1. .3);
print "Числа@{т}являются целыми";
```

будет строка

```
Числа! 2 являются целыми
```

Все правила определения идентификатора скалярной переменной при ее подстановке в строку переносятся и на переменную массива.

В связи с возможностью подстановки переменной массива можно указать быстрое решение проблемы распечатки содержимого массива. Дело в том, что если в операторе печати просто указать переменную массива, то его элементы выводятся сплошной строкой без пробелов между ними, что является неудобным при работе с числовыми данными. Если вставить переменную массива в строку, то при ее печати между элементами массива будут вставлены пробелы. Следующие два оператора печати

```
print @m, "\n"; print "@m\n";
```

отобразят массив @m предыдущего примера по-разному:

```
123
123
```

Подстановка в строку элемента массива с помощью индексного выражения ничем не отличается от подстановки скалярной переменной: элемент массива, полученный с помощью индекса, является скалярной величиной. Однако здесь может возникнуть одна интересная проблема, если в программе определена скалярная переменная с таким же идентификатором, что и у массива. Значение этой переменной или значение соответствующего элемента массива будет вставлено в строку? Например, если в программе определена скалярная переменная $var и массив @var (о том, почему это возможно, мы расскажем в разделе 3.5 данной главы), то значение переменной или элемента массива будет вставлено в строку "$var [0]".

Правильный ответ – значение элемента, так как при синтаксическом анализе интерпретатор будет рассматривать встретившуюся последовательность символов как лексему $var[0], а не как имя

переменной $var с последующим символом "[". Если необходимо использовать значение скалярной переменной $var в строке, то можно предложить три способа решения этой проблемы:

```
"${var}[0.]" # Фигурные скобки ограничивают символы, рассматриваемые
# интерпретатором как единое целое с символом $. "$var\[0]"
# Обратная дробная черта ограничивает идентификатор
# переменной. "$var" .· "[0]" # Конкатенация строк (операция ".") позволяет однозначно
# интерпретировать переменную в первой строке.
```

Иногда для работы необходимо выделить некоторое подмножество элементов массива, которое мы будем называть фрагментом массива. Можно ее выполнить просто, но не эффективно: присвоить элементам некоторого нового массива значения соответствующих элементов старого, можно воспользоваться специальной конструкцией Perl для выделения фрагмента массива. Если после имени переменной массива в квадратных скобках задать список индексов некоторых элементов массива, то такая конструкция и будет определять фрагмент массива, причем индексы не обязательно должны идти в каком-то определенном порядке – их можно задавать произвольно. Для выделения фрагмента, состоящего из последовательно идущих элементов массива, можно использовать знакомую нам операцию диапазон. Фрагмент массива сам является массивом, и поэтому его можно использовать в правой части оператора присваивания. Несколько примеров создания фрагментов массива приведено ниже:

```
@т = (10..19); # Исходный массив:
# (10, 11, 12, 13, 14, 15, 16, 17, 18, 19). @т[0, 2, 4, 6, 8];
# Фрагмент 1: (10, 12, 14, 16, 18). @т[6, 4, 5, 8, 6];
# Фрагмент 2: (16, 14, 15, 18, 16). @т[2..4];
# Фрагмент 3: (12, 13, 14). @т[8, 2..4, 0];
# Фрагмент 4: (18, 12, 13, 14, 10).
```

При выделении фрагмента массива используется имя переменной массива, начинающейся с символа "@", тогда как при ссылке на элемент массива префикс имени переменной заменяется на символ "$". Здесь опять прослеживается последовательное использование префикса для задания типа переменной. Фрагмент массива является массивом, а потому следует использовать символ "@".

Ассоциативные массивы

Ассоциативные массивы, называемые также хеш-массивами или просто хешами, – это то, чем гордятся программисты на языке Perl. Они позволяют легко создавать динамические структуры данных – списки и деревья разных видов, с помощью которых можно реализовать функциональность простой системы управления базой данных. Подобной конструкции не найти ни в одном современном языке программирования.

Ассоциативные массивы отличаются от массивов скаляров тем, что в них для ссылки на элементы используются строки, а не числовые индексы, т. е. концептуально они представляют собой список не просто значений элементов массива, а последовательность ассоциированных пар ключ/значение. Ключ является строковым литералом, и именно он и используется для доступа к ассоциированному с ним значению массива.

В программе хеши задаются аналогично массивам скаляров с помощью конструктора, представляющего собой список, заключенный в круглые скобки, в котором пары ключ/значение следуют друг за другом:

```
(ключ_1, значение_1, ключ_2, значение_2, ... , ключ_n, значение_n)
```

Для хранения ассоциативных массивов, как и для других элементов данных, используются переменные, первым символом которых является символ процента "%". Переменные, в которых хранятся ассоциативные массивы, часто называют хеш-переменными. Ассоциативный массив создается во время операции присвоения такой переменной списка значений:

```
%т = ("Имя", "Ларри", "Фамилия", "Уолл");
```

Замечание

Для краткости мы будем иногда говорить, что при создании массива его переменной присваивается список, подразумевая при этом, что в правой части операции присваивания задан конструктор массива.

В ассоциативном массиве %т ключами являются строки "имя" и "Фамилия", а ассоциированными с ними значениями, соответственно, "ларри" и "УОЛЛ". Теперь, чтобы получить значение, соответствующее какому-либо ключу, следует воспользоваться конструкцией:

```
$т{"ключ"}
```

Обратите внимание, что при работе с элементом ассоциативного массива, символ хеш-переменной "%" заменяется на символ скалярной переменной "$". Аналогично мы поступали и при ссылке на элемент массива скаляров. Единственное отличие – ключ задается в фигурных скобках. Итак, чтобы, например, присвоить некоторой скалярной переменной значение элемента хеш-массива %т, следует воспользоваться следующим оператором:

```
$surname = $m{"Фамилия"};
```

Теперь скалярная переменная $ surname имеет в качестве своего значения строку Уолл.

Замечание

> Интерпретация списка как последовательности пар ключ/значение происходит *только* при операции его присвоения хеш-переменной. Если список, присваивается переменной массива скаляров, то он интерпретируется как простая последовательность значений элементов массива.

Инициализация хеш-массива с помощью списка, элементы которого отделяются друг от друга символом запятая ",", не очень удобно, так как в длинном списке трудно выделять соответствующие пары ключ/значение. Для улучшения наглядности пару ключ/значение можно соединить последовательностью символов "=>", заменив ей разделяющую запятую в списке. По правде говоря, и запятая ",", и символы "=>" представляют собой знаки операций в Perl, причем операция "=>" эквивалентна операции "запятая" с той лишь разницей, что ее левый операнд всегда интерпретируется как строковый литерал, даже если он не заключен в кавычки.

> **Замечание**
>
> Интерпретация левого операнда операции "=>" как строкового литерала справедлива для последовательности символов, в которой используются буквы латинского алфавита. Буквы русского алфавита вызовут ошибку интерпретации, так как последовательность символов не будет распознана как слово языка Perl.

Рассмотренный нами ранее хеш-массив %т можно инициировать и таким способом:

```
%т = (
"Имя" => "Ларри",
"Фамилия" => "Уолл" );
```

Если бы мы хотели в качестве индексов имени и фамилии использовать английские слова name и surname, то этот же ассоциированный массив можно было бы задать следующим оператором:

```
%т = (
Name => "Ларри",
Surname => "Уолл" );
```

Добавить новый элемент ассоциативного массива или изменить значение существующего очень легко. Достаточно присвоить его элементу, определяемому заданным ключом, значение в операторе присваивания:

```
$т{"Имя"} = "Гарри"; # Изменили значение существующего элемента. $т{"Телефон"} = "345-56-78";
# Добавили новый элемент.
```

Если при использовании подобной конструкции ассоциативный массив еще не существовал, то при выполнении операции присваивания сначала будет создан сам массив, а потом присвоится значение

его элементу. Ассоциативные массивы, как и массивы скаляров, являются динамическими: все добавляемые элементы автоматически увеличивают их^размерность.

Удалить элемент ассоциативного массива можно только с помощью встроенной функции delete:

```
delete($m{"Телефон"}); # Удалили элемент с ключом "Телефон".
```

Совет

При изменении значения элемента ассоциативного массива с помощью ключа следует проверять правильность его задания (отсутствие дополнительных пробелов, регистр букв), так как в случае несоответствия заданного ключа элемента с существующими в хеш-массив просто добавится новый элемент с заданным ключом.

При работе с ассоциативным массивом часто требуется организовать перебор по множеству всех его ключей или значений. В языке существуют две встроенные функции – keys и values, которые представляют в виде списка, соответственно, ключи и значения ассоциативного массива, Следующий фрагмент программы

```
print keys(%m), "\n"; # Печать ключей. print values(%m), "\n";
# Печать значений.
```

отобразит на экране монитора строку ключей и строку значений массива %т

```
Фамилия Имя Телефон УоллЛарри 345-56-11
```

Обратите внимание, что они отображаются не в том порядке, как задавались с помощью конструктора массива.

Замечание

При создании элементов ассоциативного массива они сохраняются в памяти в порядке, удобном для их .последующего извлечения. Поэтому при его печати последовательность элементов не соответствует порядку их задания. Для упорядочивания значений хеш-массивов следует воспользоваться встроенной функцией сортировки.

Итак, хеш-массивы позволяют обращаться к своим элементам не с помощью числового индекса, а с помощью индекса, представленного строкой. Но что же здесь такого замечательного, какие возможности предоставляет подобная конструкция? Огромные. И первое, что приходит на ум, – это использовать ключи как аналог ключей реляционных таблиц. Правда, хеш-массивы не позволяют непосредственно хранить запись, а только один элемент, но и этого уже достаточно, чтобы создать достаточно сложные структуры данных (пример 3.7).

```
#! Perl -w %friend = (
"0001", "Александр Иванов",
"0002", "Александр Петров",
"0003", "Александр Сидоров" ); %city = (
"0001", "Санкт-Петербург",
"0002", "Рязань", '"0003", "Кострома" ) ; %job = ( ·
"0001", "учитель",
"0002", "программист",
"0003", "управляющий"
) ; . ' $person = "0001";
print "Мой знакомый $friend{$person}\n"; print "живет в городе $city{$person}\n"; print "и
имеет профессию $job{$person}.\n";
```

В этом примере создана простейшая база данных знакомых, в которой хранятся их имена, места жительства и профессии. Перечисленная информация содержится в разных хеш-массивах с одинаковым набором ключей, которые и связывают информацию по каждому человеку. Выполнение программы примера 3.6 приведет к отображению на экране монитора следующего текста:

```
Мой знакомый Александр Иванов живет в городе Санкт Петербург и имеет профессию учитель.
```

Если изменить значение переменной $person на другой ключ, то отобразится связанная информация о другом человеке.

Это только простейший пример, который может навести читателя на более плодотворные идеи применения хеш-массивов, одну из которых нам хотелось бы сейчас наметить: создание связанного списка.

Связанный список – это простейшая динамическая структура данных, расположенных в определенном порядке. Каждый элемент связанного списка состоит из некоторого значения, ассоциированного с данным элементом, и ссылки на следующий элемент списка. Последний элемент списка не имеет ссылки на следующий, что обычно реализуется в виде пустой ссылки. Для окончательного задания связанного списка следует объявить переменную, указывающую на первый элемент списка, которую называют заголовком. Для связанного списка определяются операции выбора, удаления и добавления элемента списка относительно заданного. Графически связанный список можно представить так, как показано на рис. 3.1, где указатель на следующий элемент обозначен серым цветом.

Рис 3.1. Графическое представление связанного списка

С помощью хеш-массивов связанный список реализуется просто. Для этого следует значение элемента задать в качестве ключа для следующего за ним элемента списка, определив таким образом указатель на следующий элемент. Значением последнего элемента в хеш-массиве (с ключом, равным

значению последнего элемента связанного списка) будет пустая строка "". Переменная-заголовок должна иметь значение, равное ключу первого элемента списка. В примере 3.8 показана реализация связанного списка, а также добавление нового элемента.

```
%linked_list = ( "начало" => "первый", "первый" => "третий",
"третий" => ""
);
$header = "начало";
# Добавление элемента со значением "второй"
# после элемента со значением "первый".
$temp = $linked_lis.t{ "первый"};
# Запомнили старый, указатель ., $linked_list{"второй"} = $temp;
# Добавили новый элемент. $linked_list{"первый"} = "второй";
# Указатель на новый элемент.
$item = $header;
# Печать нового связанного.списка.
while ($linked_list{$item}) { # Пока не дойдем до пустой строки ""
print $linked_list{$item}, "\n"; # будем печатать значения элементов.
$item = $linked_list{$item};
}
```

Результатом выполнения программы примера 3.8 будет печать значений элементов нового связанного списка в следующем порядке:

```
первый второй третий
```

Этот пример только демонстрационный, чтобы показать легкость реализации подобной динамической структуры. При действительной реализации связанного списка следует все возможные действия оформить в виде функций, которые в дальнейшем использовать для работы со связанным списком.

Переменные

Итак, мы познакомились с тремя основными типами данных и соответственно тремя типами переменных языка Perl, используемых для их обработки и хранения в программе. В этом последнем параграфе данной главы мы суммируем наши знания о переменных и дополним некоторыми аспектами, связанными с их реализацией в интерпретаторе и использованием в простейшей операции присваивания.

Напомним, что переменную можно рассматривать как именованную область памяти, которая не только предоставляет возможность обращаться к содержимому этой области памяти по имени переменной, но и задает тип хранимых в ней данных, определяя, таким образом, формат хранения данных и набор применимых к ним операций.

Первый символ имени переменной языка Perl определяет ее тип. В языке Perl можно определить переменные трех типов: скаляр, массив и хеш-массив. Для каждого типа переменных интерпретатор создает собственное пространство имен, храня в нем идентификаторы объявленных в программе переменных заданного типа. Это позволяет создавать переменные разных типов с одинаковыми идентификаторами. В программе Perl бесконфликтно могут сосуществовать и скалярная переменная $var, и массив скаляров @var, и хеш-массив %var.

Так как имя переменной всегда начинается с одного из символов "$", "@" или "%", то использование в качестве идентификатора "предопределенных" слов не приводит к конфликтам в процессе интерпретации программы.

Следует отметить, что в языке Perl существует достаточно большой набор специальных встроенных переменных, которые можно использовать для получения или изменения разнообразной системной информации. Большинство из них являются скалярными переменными с "идентификатором", состоящим из одного специального символа, например, $~, $__ и т. д.

(Подробно специальные переменные рассматриваются в части 14.)

Немного забегая вперед, скажем, что переменные используются в выражениях, которые, в свою очередь, могут являться операндами определенных в языке операций. Интерпретация операций и значений их операндов в Perl зависит от контекста, в котором они вычисляются. Существует два основных контекста: скалярный и списковый. Например, если левым операндом операции присваивания является скалярная переменная, то правый операнд вычисляется в скалярном контексте, т. е. его значением должен быть скаляр; если левый операнд массив или хеш (или фрагмент массива или хеша), то правый операнд вычисляется в списковом контексте, т. е. должен предоставить значение, являющееся списком.

Замечание

Использование конструктора массива с элементами, являющимися скалярными переменными, в качестве левого операнда операции присваивания предписывает вычислять правый операнд в списковом контексте.

Переменные массивов и хешей, а также их конструкторы, используемые в качестве операндов операции присваивания (=), будут иметь разные значения в зависимости от используемого контекста.

В списковом контексте конструктор массива будет иметь значение, представляющее собой все значения списка в заданном порядке, тогда как в скалярном контексте он будет иметь значение, равное значению последнего элемента списка:

```
@array = (0, 2, 4); # Массив скаляров @array
# содержит три элемента: 0, 2, 4. $last = (0, 2, 4);
```

```
# Значение скалярной переменной $last равно 4.
```

Переменная массива скаляров в списковом контексте возвращает список всех элементов массива, а употребленная в скалярном контексте, в отличие от своего конструктора, будет иметь значение, равное числу элементов массива:

```
@new_array = Sarray; # Массив @new_array
# содержит все элементы массива @array. $number = Sarray;
# Скалярная переменная $number
# равна 3 - числу элементов массива Sarray.
```

В качестве левого операнда операции присваивания можно использовать заключенный в круглые скобки список, элементами которого являются скалярные переменные, а правым операндом может быть конструктор массива или переменная массива. В этом случае каждой переменной левого списка присваивается значение соответствующего элемента правого списка. Если количество элементов правого списка меньше числа скалярных переменных левого списка, то переменные, которым не хватило значений, будут не определены. Несколько примеров приводится ниже:

```
($a, $b = (1, 2, 3); # $a = 1, $ b = 2.
($a, $b $c) = (1, 2); tt $a= 1, $ b =2, $c = "".
```

В языке Perl каждая операция имеет вычисляемое значение. Значением операции присваивания со скаляром в качестве правого операнда является значение этого скаляра в любом контексте. Если правым операндом является конструктор массива или массив, то в списковом контексте значением операции присваивания будет список элементов массива, а в скалярном контексте – число элементов массива.

```
$x = ( @a = (1, 2) ); # $x = 2.
```

Хеш-переменные так же, как и массивы скаляров, ведут себя по-разному в разных контекстах. Употребленные в списковом контексте, они вычисляются в виде обычного списка, состоящего из элементов всех пар ключ/значение. Однако порядок элементов в списке может не соответствовать порядку задания пар в хеше. Это связано с тем, что внутренняя реализация хеш-массивов основана на использовании хеш-таблиц для ускорения поиска соответствия ключ/значение, поэтому при вычислении хеша в виде списка порядок следования "пар" ключ/значение и не соответствует порядку их задания, но после ключа всегда следует его значение.

```
%hash = ( I => I, 2 =>2, 3 => 3, 4=> 4 );
@list = %hash; # @list = (blue, 3, green, 2, red, 1, black, 4)
```

В скалярном контексте вычисляемым значением хеш-массива является строка, состоящая из числа использованных участков записей (bucket) и числа выделенных участков записей, разделенных

символом "/". Если присвоить скалярной переменной хеш-массив %hash предыдущего примера, то ее значением будет строка "4/8", которая говорит, что 4 из 8 выделенных участков записей хеш-таблицы используется.

Любая переменная в Perl может находиться в состоянии: определена или не определена. Если ей присвоено какое-либо значение, то такая переменная считается определенной. Если ей не было присвоено значение до ее использования в каком-либо выражении, то такая переменная считается не определенной. Чтобы узнать, определена или нет переменная, можно воспользоваться встроенной функцией defined, которая возвращает г (Истина) в случае определенности переменной и пустую строку "", когда переменная не определена:

```
@т = (1,2);
defined @т; '# Возвращает 1, массив скаляров @т определен.
defined $т; # Возвращает "", переменная $т не определена.
```

Определенную переменную можно в любой момент сделать неопределенной, выполнив функцию undef с параметром, равным имени этой переменной:

```
@т = (1,2);
defined @т; # Возвращает 1, массив скаляров @т определен.
undef @m;
defined @m; tt Возвращает "", массив скаляров @т не определен.
```

Если переменная сделана не определенной с помощью функции undef, ТО она, естественно, теряет присвоенное ранее ей значение.

И последнее, о чем нам здесь хотелось бы упомянуть в связи с обсуждением переменных Perl, — это области видимости переменных, т. е. области доступности переменных. Во всех приведенных примерах все переменные являются глобальными, они доступны из любой точки программы. Язык Perl, однако, позволяет создавать переменные с областью видимости, ограниченной блоком или телом подпрограммы. Это так называемые локальные, или синтаксические переменные, имена которых могут совпадать с именами глобальных переменных, и которые существуют, только пока выполняются операторы некоторого блока или подпрограммы. После завершения выполнения операторов блока, эти переменные уничтожаются.

(Более подробно локальные переменные описаны в части 11.)

Вопросы для самоконтроля

1. Перечислите три встроенных типа данных языка Perl.
2. В чем отличие числового литерала от строкового.
3. Объясните различие между строкой, ограниченной одинарными кавычками, и строкой, ограниченной двойными кавычками.

4. Каким образом можно выполнить системную команду из программы Perl?
5. Что такое массив скаляров и ассоциативный массив?
6. Как задаются в программе массивы и хеш-массивы?
7. Как объявляются в программе переменные для хранения скалярных данных, массивов скаляров и хеш-массивов?
8. Что такое интерполяция переменной?
9. Можно ли интерполировать массивы скаляров и хеш-массивы?
10. Какие два контекста для операции присваивания вы знаете, и как ведут себя массивы скаляров и хеш-массивы в них?

Упражнения

1. Найдите ошибки в следующем фрагменте кода Perl:
   $m.= 'Исходные данные:\n'; @data = (1, 2, 3, 4} ; print $m, 'Запись: Sdata';
2. Что напечатают следующие операторы и почему:
   $т = "Скаляр \$m\n";
   @т = (1, 2, 3);
   print "Значение равно $m[0]\n";
   print "Значение равно $m [0]";
3. Предположим, что есть группа слушателей курса по языку Perl, состоящая из 10 человек. В середине курса слушатели сдают промежуточный экзамен, а в конце — выпускную работу. За экзамен и за выпускную работу выставляется оценка по пятибалльной системе. По окончании курса каждый слушатель получает удостоверение, в котором указано, естественно, его имя, а также оценки за экзамен и выпускную работу. Разработайте базу данных слушателей курса, которую можно использовать для автоматизации подготовки удостоверений об успешном окончании курса. (Указание: воспользуйтесь хеш-массивами.)
4. Дополните программу примера 3.8 удалением первого и последнего элемента связанного списка. (Указание: воспользуйтесь функцией delete ().)
5. После выполнения упражнения 4 в связанном списке останется один элемент. Удалите его, распечатайте, а затем снова добавьте два элемента в список и распечатайте.