Глава 10. Работа со строками

Учебник по Perl

Содержание [скрыть]

Язык, созданный первоначально с главной целью – облегчить обработку большого количества отчетов, просто обязан располагать развитыми средствами для работы с текстом. Напомним, что в среде UNIX, из которой вышел язык Perl, средства для обработки текстовых строк имеются в различных утилитах: sed, awk, grep, cut. Командный интерпретатор shell, также обладающий некоторыми средствами для обработки строк, позволяет организовать совместную работу этих утилит, передавая выход одной программы на вход другой через механизм, называемый конвейером. Такой подход требует написания достаточно изощренных скриптов на языке shell в сочетании с обращением к внутренним командам утилит обработки текста sed или awk. Язык Perl, являясь средством создания программ-сценариев, в то же время один обладает всеми возможностями перечисленных утилит и даже их превосходит.

Типичная задача, возникающая при обработке текстового файла, заключается в том, чтобы найти в нем фрагмент, удовлетворяющий заданным условиям, и выполнить над ним некоторую операцию: удалить, заменить на другой фрагмент, извлечь для дальнейшего использования и т. д. Условия поиска можно достаточно просто выразить словами. Например: найти строку, содержащую слово Perl. Или: найти все фрагменты, находящиеся в конце строки и содержащие две цифры, за которыми следует произвольное количество прописных букв. Для формализованной записи подобных условий

используются *регулярные выражения,* позволяющие описать *образец* или *шаблон* поиска при помощи специальных правил. Манипуляции с регулярными выражениями осуществляются при помощи соответствующих операций, которые мы также рассмотрим в этой главе.

Регулярные выражения

Регулярное выражение, по сути, представляет собой набор правил для описания текстовых строк. Сами правила записываются в виде последовательности обычных символов и метасимволов, которая затем в качестве образца используется в операциях поиска и замены текста. *Метасимволы* – это символы, имеющие в регулярном выражении специальное значение. Пользователи DOS/Windows хорошо знают метасимвол *, используемый для порождения имен файлов и обозначающий любую допустимую последовательность. Регулярные выражения применяются многими программами UNIX, в том числе интерпретатором shell. Каждая из них использует свое множество метасимволов. В большинстве случаев они совпадают.

Метасимволы

В языке Perl к Метасимволам относятся следующие символы:

"\", ".", "^", "$", "|", "[", "(", ")", "*", "+", "?", "}".

Различные метасимволы выполняют в регулярном выражении различные функции, в частности, используются для обозначения одиночного символа или их группы, обозначают привязку к определенному месту строки, число возможных повторений отдельных элементов, возможность выбора из нескольких вариантов и т. д.

Регулярное выражение, подобно арифметическому выражению, строится с соблюдением определенных правил. В нем можно выделить операнды (элементы) и операции.

Простейшее регулярное выражение состоит из одного обычного символа. Обычный символ в регулярном выражении представляет самого себя. Соответственно, последовательность обычных символов представляет саму себя и не нуждается в дополнительной интерпретации. Для использования в операциях в качестве образца регулярное выражение заключается между двумя одинаковыми символами-ограничителями. Часто в качестве ограничителя используется символ "/" (косая черта). Например, образцу /Perl/ будут соответствовать все строки, содержащие слово Perl.

Если, в регулярном выражении какой-либо метасимвол требуется использовать в буквальном, а не специальном значении, его нужно *экранировать* или *маскировать* при помощи другого метасимвола – "\". Например, образцу /*/ соответствует фрагмент текста *. Здесь первый метасимвол "\" экранирует второй метасимвол "\", а третий метасимвол "\" экранирует метасимвол "*".

Метасимвол "." представляет любой одиночный символ, кроме символа новой строки. Так, образцу /./ будет соответствовать любая непустая строка. Если в операциях сопоставления с образцом установлен флаг s, то метасимволу "." соответствует также и символ новой строки.

Метасимвол "[" используется в конструкции [...] для представления любого одиночного символа из числа заключенных в скобки, т. е. он представляет *класс* символов. Два символа, соединенные знаком минус, задают диапазон значений, например, [A-za-z] задает все прописные и строчные буквы английского алфавита. Если первым символом в скобках является символ "^", вся конструкция обозначает любой символ, не перечисленный в скобках. Например, [^0-9] обозначает все нецифровые символы. Ниже мы рассмотрим и другие способы представления классов символов.

Метасимволы "^" и "$" используются для задания привязки к определенному месту строки.

Метасимвол "^" в качестве первого символа регулярного выражения обозначает начало строки.

Метасимвол "$" в качестве последнего символа регулярного выражения обозначает конец строки.

Например, следующим образцам соответствуют:

- /^$/ – пустая строка (начало и конец, между которыми пусто);
- /^рег1/ – слово Perl в начале строки;
- /Рег 1$/ –слово Perl в конце строки.

Метасимвол "|" можно рассматривать как символ операции, задающей выбор из нескольких вариантов (подобно логической операции ИЛИ). Например, образцу /a|b|c/ соответствует фрагмент текста, содержащий любой из символов a, b, c, Если вариантами выбора являются одиночные символы, как в данном примере, то лучше использовать конструкцию, определяющую класс символов, в данном случае [abc]. Но в отличие от конструкции [...] операция "|" применима и тогда, когда вариантами выбора являются последовательности символов. Например, образцу /Word|Excel!windows/ соответствует фрагмент^текста^содержащий любое из слов Word, Excel, Windows.

Следующая группа метасимволов служит в качестве коэффициентов или *множителей,* определяющих количество возможных повторений отдельных *атомарных* элементов регулярного выражения.

- r* – нуль и более повторений
- r+– одно и более повторений
- r? – нуль или одно повторение
- r{n}– ровно n повторений
- r{n}, – n и более повторений
- r{n,m} – минимум n, максимум m повторений r

Атомарные элементы, или *атомы* – это простейшие элементы, из которых строится регулярное выражение. Это не обязательно одиночный символ.

Образец Соответствие

- /.*/ любая строка;
- /.+/ любая непустая строка;

- / [0-9] {3} / любая последовательность из трех цифр;
- /\ [+/ последовательность, состоящая из любого числа символов [.

В первых двух примерах атомом является метасимвол ".". В третьем образце в качестве атома выступает конструкция [0-9], определяющая класс цифровых символов. В четвертом образце атом – это пара символов "\[", включающая метасимвол "\", отменяющий специальное значение следующего за ним метасимвола "[". Полный список атомов мы приведем после изучения всех необходимых синтаксических конструкций.

Алгоритм, применяемый в операциях поиска и замены (см. ниже) для обработки регулярных выражений, содержащих множители, является "жадным": он пытается найти для образца, снабженного множителем, максимальный сопоставимый фрагмент текста. Рассмотрим, например, что происходит при поиске в строке

"Скроен колпак не по-колпаковски, надо колпак переколпаковать" фрагмента, удовлетворяющего образцу /. * колпак/.

Алгоритм найдет максимальный фрагмент, удовлетворяющий выражению . * (вся строка без завершающего символа новой строки), затем начнет двигаться назад, отбрасывая в найденном фрагменте по одному символу, до тех пор, пока не будет достигнуто соответствие с образцом. Найденный фрагмент будет иметь вид "Скроен колпак не по-колпаковски, надо колпак переколпак".

Можно заставить алгоритм работать иначе, снабдив множитель "*" модификатором "?". В этом случае алгоритм из "жадного" превращается в "ленивый" и будет для образца, снабженного множителем, искать минимальный соответствующий фрагмент. "Ленивый" алгоритм для множителя "*?" начнет поиск в строке с пустого фрагмента "", добавляя к нему по одному символу из строки до тех пор, пока не достигнет соответствия с образцом. В этом случае найденный фрагмент будет иметь вид "Скроен колпак". Все сказанное справедливо и для других множителей. Например, в строке "1234567" будет найден:

- для образца /\d*/ – максимальный фрагмент "1234567";
- для образца /\d+/ – максимальный фрагмент "1234567";
- для образца /\d?/ – максимальный фрагмент "1";
- для образца /\d{2,5}/ – максимальный фрагмент "12345";
- для образца /\d*?/ – минимальный фрагмент "";
- для образца /\d+?/ – минимальный фрагмент "1";
- для образца /\d??/ – минимальный фрагмент "";
- для образца /\d{2,5}?/ – минимальный фрагмент "12".

Метапоследовательности

Символ "\", непосредственно предшествующий одному из метасимволов, отменяет специальное значение последнего. Если же "\" непосредственно предшествует обычному символу, то, напротив, такая последовательность во многих случаях приобретает специальное значение. Подобного рода последовательности будем называть *метапоследовательностями.* Метапоследова-тельности в регулярном выражении служат, в основном, для представления отдельных символов, их классов или определенного места в строке, дополняя и иногда дублируя/функции метасимволов. Рассмотрим существующие метапосл едовател ьности.

- \nnrt – представляет\символ, восьмеричный код которого равен лил. Например, последовательность \120\145\162\154 представляет слово Perl (\120 – восьмеричный код буквы P, \145 – буквы e, \162 – буквы r, \154- буквы!);
- \xпп – представляет символ, шестнадцатеричный код которого равен *пп.* Слово Perl, например, представляется последовательностью \x50\x65\x72\x6c;
- \cп – представляет управляющий символ, который генерируется при нажатии комбинации клавиш <Ctrl>+<N>, где N – символ, например, \co соответствует <Ctrl>+<D>;
- \$ – символ "$"; G \@ – символ "@"; O \% – символ *"%";*
- \a – представляет символ с десятичным ASCII-кодом 7 (звонок). При выводе производит звуковой сигнал;
- \e – символ Esc, десятичный ASCII-код 27;
- \f – символ перевода страницы, десятичный ASCII-код 12;
- \п – символ новой строки, десятичный ASCII-код 10;
- \r – символ "возврат каретки", десятичный ASCII-код 13;
- \t – символ горизонтальной табуляции, десятичный ASCII-код 9;
- \v – символ вертикальной табуляции, десятичный ASCII-код 11;
- *\з* – представляет класс *пробельных* символов. К пробельным символам относятся: пробел, имвол табуляции, возврат каретки, символ новой строки и символ перевода страницы. То же самое, что и [\t,\r,\n,\fj;
- \s – представляет класс непробельных символов, то же самое, что и класс [л \t,\r,\n,\f];
- \d – класс цифровых символов, то же, что и [0-9]; П \o – класс нецифровых символов, то же, что и [лo-9];
- \w – представляет класс буквенно-цифровых символов, состоящий из букв, цифр и символа подчеркивания "_". То же самое, что и [a-zA-z_ o-9j. Обратите внимание, что в этот класс входят только буквы английского алфавита;
- \w – представляет класс небуквенно-цифровых символов. То же, что и Pa-zA-Z_0-9];
- \д – обозначает начало строки; П \z – обозначает конец строки;

Замечание

> Последовательность \A эквивалентна метасимволу ^ в начале регулярного выражения, а последовательность \z– метасимволу $ в конце регулярного выражения, за исключением одного случая. Назовем строку, содержащую внутри себя символы новой строки (ASCII 10), *мульти-строкой.* Фактически муль-ти-строка состоит из отдельных строк, разделенных ограничителями – символами новой строки. При выводе мульти-строка отображается в виде нескольких строк. Если к мульти-строке применяется операция поиска или замены с опцией /т (см. ниже), то последовательности \A и \z обозначают соответственно начало и конец всей мульти-строки, а метасимволам ^ и $ соответствуют еще и границы внутренних строк, образующих мульти-строку.

- \ь – обозначает границы слова. Под словом понимается последовательность символов из класса \w. Граница слова определяется как точка между символами из класса \w и символами из класса \w;
- \в – обозначает не-границы слова, т. е. класс символов [^\ь];
- \l – обозначает, что следующий символ регулярного выражения преобразуется к нижнему регистру. Например, запись /\ip/ означает, что символ P будет преобразован к нижнему регистру, после чего новый образец /p/ может быть использован в соответствующей операции поиска или замены;
- \и – обозначает, что следующий символ регулярного выражения преобразуется к верхнему регистру;
- \ь.. .\E – обозначает, что все символы в регулярном выражении между \ь и \E преобразуются к нижнему регистру;
- \и.. .\E – обозначает, что все символы в регулярном выражении между \U и \E преобразуются к верхнему регистру;
- \Q...\E– обозначает, что все метасимволы в регулярном выражении между \Q и \E экранируются при помощи символа "\". Например, запись /\0^*?+\E/ эквивалентна записи /\^*\?\+у;
- \с – обозначает точку, в которой закончился предыдущий поиск m//g (см. описание операции поиска т//).

Атомы

Из всех метасимволов, перечисленных в начале раздела, нам осталось рассмотреть "(" и "). Эти метасимволы служат для группирования ряда элементов, входящих в состав образца, в один элемент. Например, образцу /<abc)+/ соответствует строка, состоящая из одного или более повторений последовательности abc, в то время, как образцу /abc+/ – строка, состоящая из начальных символов аь, за которыми следует один или более символов с.

Теперь мы можем перечислить "атомы", из которых строится регулярное выражение.

- Регулярное выражение в скобках, представляющее несколько элементов, сгруппированных в один.
- Любой обычный символ (не метасимвол).
- Символ ".", представляющий любой одиночный символ, кроме символа новой строки.
- Конструкция [...], представляющая класс символов, перечисленных в квадратных скобках.

- Метапоследовательность, представляющая символ или класс символов *(см. раздел 10.1.2):* \&, \n, \r, \t, \f, \e, \d, \D, \w, \w, \s, \s.
- Метапоследовательность вида *ппп,* определяющая символ при помощи его восьмеричного ASCII-кода *ппп (см. раздел 1&Ь2)*
- Метапоследовательность вида \хлп, определяющая (символ при помощи его шестнадцатеричного ASCII-кода пп *(см. раздел 10.1.2).*
- Метапоследовательность вида \сп, представляющая управляющий символ ctri-n *(см. раздел 10.1.2).*
- Конструкция вида \number, представляющая обратную ссылку *(см раздел 10.1.4).*
- Любая конструкция вида *character,* не имеющая специального значения, а представляющая собственно символ *character,* например: *, \$_{y}$, \ь.
- Напомним, что в регулярном выражении множители *, +, ?, {n,m} применяются именно к атому, расположенному непосредственно слева.

Обратные ссылки

Ранее мы установили, что группу элементов регулярного выражения можно заключить в скобки и рассматривать как один элемент. Заключение группы элементов в скобки имеет дополнительный и весьма полезный эффект. Если в результате поиска будет найден фрагмент текста, соответствующий образцу, заключенному в скобки, то этот фрагмент сохранится в специальной переменной. Внутри регулярного выражения к нему можно будет обратиться, используя запись *number,* где number—номер конструкции () в исходном регулярном выражении. Запись \number, указывающую на найденный по образцу фрагмент текста, будем называть *обратной ссылкой.* Можно задать любое количество конструкций вида () и ссылаться на соответствующие найденные фрагменты текста, как на \i, \2 и т. д. Например, образцу /(. +) -\1/ соответствуют слова "ха-ха", "хи-хи", "ку-ку" и т. п., а образцу /{.)(.). ?\2\1/ — все палиндромы из четырех или пяти букв. (Палиндром — слово или предложение, которое одинаково читается слева направо и справа налево.)

Внутри образца конструкция \n (n=i,...,9) всегда обозначает обратную ссылку. Запись вида \пп также интерпретируется как обратная ссылка, но только в том случае, если в исходном выражении задано не менее, чем пп скобочных конструкций вида (). Иначе запись \пп обозначает символ с восьмеричным кодом пп.

Для ссылки на найденный фрагмент текста за пределами регулярного выражения, например, при задании замещающего текста в операции замены, вместо записи \number используется запись $пшпЬег. Например, операция замены *(см. раздел 10.2)*

```
$str=~s/(\S+)\s+(\S+)/$2 $1/
```

меняет местами первые два слова в строке $str.

Область действия переменных $1, $2 и т. д, распространяется до наступления одного из следующих событий: конец текущего блока; конец строки, являющейся аргументом функции evai; следующее

совпадение с образцом. Аналогичным образом определяется область действия и для следующих предопределенных переменных, используемых в операциях сопоставления с образцом.

- $& – часть строки, найденная при последней операции сопоставления с образцом.
- $' – часть строки, стоящая перед совпавшей частью при последней успешной операции сопоставления с образцом.
- $' – часть строки, стоящая после совпавшей части при последней успешной операции сопоставления с образцом.

Например, в результате выполнения операции поиска (см. раздел 10.2)

```
$str=~m/two/
```

в строке $str="one two three" образца /two/ переменным будут присвоены следующие значения:

```
$& - "two"; $* - "one"; $' - "three".
```

Эти значения будут сохраняться до наступления одного из перечисленных выше событий, и их можно использовать, например, для формирования строки с обратным порядком следования слов: $rstr=$'.$&.$". Строка $rstr будет иметь ВИД "three two one".

Следует отметить, что, если обращение к одной из переменных $&, $', $· встречается *где-либо* в программе, то интерпретатор peli будет вычислять и запоминать их для *каждой* операции сопоставления с образцом, что, в свою очередь, замедляет выполнение всей программы. Поэтому не следует использовать данные переменные без особой необходимости.

Расширенный синтаксис регулярных выражений

Выше мы использовали скобки для группирования нескольких элементов регулярного выражения в один элемент. Побочным эффектом данной операции является запоминание найденного фрагмента текста, соответствующего образцу, заключенному в скобки, в специальной переменной. Если скобки используются только для группирования элементов регулярного выражения, то найденный фрагмент текста можно не запоминать. Для этого после открывающей скобки "(" следует поместить конструкцию "?:", например, в случае задания альтернативы

```
/ (?: Perl i perl) /.
?:pattern
```

Конструкция относится к классу конструкций общего вида (?...), добавляющих новые возможности для задания образцов за счет расширения синтаксиса регулярного выражения, а не за счет введения новых метасимволов или метапоследовательностей. Символ, следующий за символом "?", опреде- , ляет функцию, выполняемую данной синтаксической конструкцией. В на^ стоящее время определены около десяти расширенных конструкций регулярного выражения, большая часть которых рассмотрена в

данном разделе. Оставшиеся конструкции, на наш взгляд, не являются необходимыми для первоначального знакомства с языком.

```
(?#text)
```

Текст после символа tt и до закрывающей скобки), комментарий, игнорируется интерпретатором и используется для добавления непосредственно в регулярное выражение.

```
(?:pattern)
(?imsx-imsx:pattern)
```

Использовать скобки только для группирования элементов без создания обратных ссылок. Символы imsx-imsx между вопросительным знаком и двоеточием интерпретируются как флаги, модифицирующие функцию данного выражения (см. ниже).

```
(?=pattern)
```

Следующий фрагмент в тексте должен соответствовать образцу pattern. Обычно образец для операций поиска или замены задается при помощи регулярного выражения. Результатом операции поиска является фрагмент, соответствующий образцу, который сохраняется в специальной переменной $&. Конструкция (?=pattern) в составе регулярного выражения позволяет задать условие поиска, не включая найденный фрагмент, соответствующий образцу pattern, в результат, сохраняемый в переменной $&. Конструкция (?=pattern) в регулярном выражении задает условие, что *следующий* фрагмент текста должен удовлетворять образцу pattern. Обращаем внимание на слово *следующий.* Данная конструкция неприменима для задания условия, что *предыдущий* фрагмент текста должен соответствовать заданному образцу. Например, образцу /ь+(?=с+)/ соответствует часть строки, состоящая из одной или более литер ь, за которыми следуют одна или более литер с, причем найденный фрагмент текста будет содержать только последовательность литер ь без последовательности литер с.

Рассмотрим строку

```
$str = "aaabbbcccddd";
```

В результате операции поиска

```
$str =~ m/b+(?=c+)/;
```

будут сохранены следующие значения в специальных переменных:

```
$Ч – aaa $& - bbb $' – cccddd
```

Если в операции поиска указать образец /ь+с+/, то значения специальных переменных будут следующими:

```
$" – aaa $& – bbbccc $' - ddd
```

В свою очередь, операция поиска по образцу / (?=b+) c+/ в нашем примере не даст результата. Данный образец задает условие, что следующий фрагмент текста должен содержать непустую последовательность литер b. В нашей строке такой -фрагмент будет найден (это фрагмент bbb>, но не будет включен в результат поиска. Следующий фрагмент, в соответствии с образцом, должен представлять непустую последовательность литер c, но в нашем случае этого соответствия не будет, так как мы остановились перед фрагментом bbb, не включив его в результат, и поэтому следующим фрагментом будет bbb, а не ccc.

Конструкцию (?=pattern) будем называть *регулярным выражением с положительным постусловием.*

```
(?!pattern)
```

Конструкция (? [pattern) в регулярном выражении задает условие, что *следующий* фрагмент текста *не* должен удовлетворять образцу pattern. Найденный фрагмент не запоминается в переменной $&. Например, результат операции поиска

```
$str =~ m/b+(?!c+)/;
```

в рассмотренной выше строке $str будет зафиксирован в следующих значениях специальных переменных:

```
$' – aaa
$& - bb
$' – bcccddd
```

Найденная подстрока соответствует образцу: она состоит из двух литер bb, за которыми не следует последовательность литер c.

По аналогии с предыдущей данную конструкцию назовем *регулярным выражением с отрицательным постусловием.*

```
(?<=pattern)
```

Конструкция (?<=pattern) в регулярном выражении задает условие, что *предыдущий* фрагмент текста должен удовлетворять образцу pattern. Найденный фрагмент не запоминается в переменной $&. Образец pattern должен иметь фиксированную длину, т. е. не содержать множителей.

В нашем примере в результате операции поиска

```
$str =~ m/(?<=b)b+/;
```

значения специальных переменных будут распределены следующим образом:

```
$' – aaab
$& - bb
$' - cccddd I
```

Данную конструкцию *назовем регулярным выражением с положительным предусловием.*

```
(?<!pattern)
```

Конструкция (?<! pattern) в регулярном выражении задает условие, что *предыдущий* фрагмент текста *не* должен удовлетворять образцу pattern. Найденный фрагмент не запоминается в переменной $&. Образец pattern должен иметь фиксированную длину, т. е. не содержать множителей.

В нашем примере в результате операции поиска

```
$str =~ m/(?<!b)c+/;
```

специальные переменные получат следующие значения:

```
$' – aaabbbc $& – ee $' - ddd
```

Данную конструкцию будем называть *регулярным выражением с отрицательным предусловием.*

```
(?imsx-imsx)
```

Задание флагов операции сопоставления с образцом осуществляется в самом образце. Флаги модифицируют выполнение операции и обычно являются частью синтаксиса самой операции. Расширенная конструкция (?imsx-imsx) позволяет задать флаги операции внутри самого образца. Эта возможность может быть полезной, например, в таблицах, когда разные элементы таблицы требуется по-разному сопоставлять с заданным образцом, например, некоторые элементы – с учетом регистра, другие – без учета. Допустимыми являются следующие флаги.

- i – поиск без учета регистра;
- dm – строка трактуется как мульти-строка, состоящая из нескольких строк, разделенных символом новой строки;
- s – строка трактуется как одна строка, в этом случае метасимволу "." соответствует любой одиночный символ, включая символ новой строки;
- x – разрешается использовать в образцах пробелы и комментарии. При использовании флага x пробелы в образцах игнорируются. Признаком комментария является символ #, как и в основном тексте Perl-программы. Пробелы позволяют сделать образец более читаемым.

Одна из литер i, m, s, x после знака "-" обозначает отмену соответствующего флага.

При помощи данной расширенной конструкции можно задать, например, следующий образец

```
/(?ix) perl # игнорирование регистра при поиске/
```

Флаг i предписывает не учитывать регистр в операциях сопоставления с образцом, так что образцу будет соответствовать и слово perl, и слово Perl. Флаг x позволяет выделить слово "perl" пробелами и использовать в образце комментарий. И пробелы, и комментарий не будут учитываться в операции сопоставления с образцом.

Сводка результатов

Изложенное в данном разделе можно суммировать в виде набора правил, которыми следует руководствоваться при работе с регулярными выражениями.

- Любой одиночный символ, не являющийся метасимволом, представляет самого себя.
- Специальное значение метасимвола можно отменить, поместив перед ним специальный экранирующий метасимвол "\".
- Можно определить класс символов, заключив их в квадратные скобки. Если первым после открывающей скобки "[" является символ "Л", то вся конструкция обозначает символы, не перечисленные в скобках. Внутри скобок два символа, соединенные знаком "-", определяют диапазон. Чтобы включить в состав класса символ "-", его следует поместить в начале или в конце списка, или экранировать при помощи символа "\".
- Символы можно задавать при помощи метапоследовательностей, состоящих из символа "\", за оторым следует обычный символ или последовательность символов.
- Альтернативный выбор задается перечислением альтернативных вариантов, разделенных символом "|". Обычно вся конструкция при этом заключается в круглые скобки.
- Внутри регулярного выражения можно выделить подобразец, заключив его в круглые скобки. На и-ю конструкцию в скобках можно затем сослаться, используя нотацию \n внутри и $n – вне регулярного выражения.

В заключение раздела приведем в табл. 10.1 и 10.2 сводку метасимволов и метапоследовательностей, рассмотренных в данной главе.

Таблица 10.1. *Символы, имеющие специальное значение у в регулярных выражениях Perl*

Метасимвол	Интерпретация
\	Отменяет (экранирует) специальное значение следующего за ним метасимвола
-	Любой одиночный символ, кроме символа новой строки
	Любой одиночный символ, включая символ новой строки, если в операции сопоставления с образцом задан флаг s

л	Обозначает начало строки, если является первым символом образца
$	Обозначает конец строки, если является последним символом образца
I	Разделяет альтернативные варианты
[...]	Любой одиночный символ из тех, которые перечислены в квадратных скобках. Пара символов, разделенных знаком минус, задает диапазон символов. Например, [A-Za-z] задает все прописные и строчные буквы английского алфавита. Если первым символом в скобках является символ "л", то вся конструкция обозначает любой символ, не перечисленный в скобках. Внутри скобок символы ".", "*", "[" и "\" теряют свое специальное значение
(...)	Группирование элементов образца в один элемент
*	Нуль и более повторений регулярного выражения, стоящего непосредственно перед *
+	Одно или более повторений регулярного выражения, стоящего непосредственно перед +
9	Одно или ни одного повторения регулярного выражения, стоящего непосредственно перед ?
{ n, m }	Минимальное n и максимальное m число повторений регулярного выражения, стоящего перед {п, т). Конструкция {п} означает ровно п повторений, (п, } – минимум п повторений
\0пп	Символ, восьмеричный код которого равен пп
\a	При выводе производит звуковой сигнал
\A	Обозначает начало строки
\b	Обозначает границы слова.
	Под словом понимается последовательность символов из класса \w (см. ниже). Граница слова определяется как точка между символами из класса \w

	и символами из класса \w (см. ниже)
\B	Обозначает не-границы слова
\cn	Управляющий символ, который генерируется при нажатии комбинации клавиш <Ctrl>+<N>
\d	Любой цифровой символ, то же, что и [0-9]
\D	Любой нецифровой символ, то же, что и [^0-9]
\e	Символ Esc, ASCII 27
\E	Ограничитель последовательностей \L, \u, \Q
\f	Символ перевода страницы, ASCI1 1 2
\G	Обозначает точку, в которой закончился предыдущий поиск m/ /g
\1	Преобразует следующий символ регулярного выражения к нижнему регистру

Таблица 10.2. *Метапоследовательности в регулярных выражениях Perl*

Метапоследовательность	Значение
\L	Преобразует все последующие символы в регулярном выражении к нижнему регистру до тех пор, пока не встретится последовательность \E
\n	Символ новой строки, ASCII 10
\Q	Эквивалентно экранированию всех последующих метасимволов в регулярном выражении при помощи символа "\" до тех пор, пока не встретится последовательность \E
\r	Символ "возврат каретки", ASCII 1 3

\s	Класс пробельных символов: пробел (space), символ табуляции (tab), возврат каретки (carriage return), символ перевода строки (line feed) и символ перевода страницы {form feed); эквивалентное \t,\r,\n, \f]
\S	Класс непробельных символов
\t	Символ табуляции, ASCII 9
\u	Преобразует следующий символ к верхнему регистру
\U	Преобразует все последующие символы в регулярном выражении к верхнему регистру до тех пор, пока не встретится последовательность \E
\v	Символ вертикальной табуляции, ASCII i 1
\w	Любая буква, цифра или символ подчеркивания
\W	Любой символ, не являющийся буквой, цифрой или символом подчеркивания
\xnn	Символ, шестнадцатеричный код которого равен пп
\z	Обозначает конец строки

Операции с регулярными выражениями

В данной главе неоднократно упоминались операции с регулярными выражениями, такие как поиск по образцу. Основными среди них являются: операция поиска т//, операция замены s/// и операция транслитерации tr///.

Операция поиска

```
m/PATTERN/cgimosx
```

Операция поиска HI/PATTERN/ осуществляет поиск образца PATTERN. Результатом операции является значение 1 (ИСТИНА) или пустая строка"

(ЛОЖЬ). По умолчанию поиск осуществляется в строке, содержащейся в специальной переменной $_. Можно назначить другую строку для поиска в ней заданного образца при помощи операций связывания

=~ или ! ~:

```
$var =~ m/PATTERN/cgimosx
```

В результате последней операции поиск образца PATTERN будет осуществляться в строке, задаваемой переменной $var. Если в правой части операции связывания стоит операция поиска т//, то в левой части может находиться не обязательно переменная, а любое строковое выражение.

Операция ! ~ отличается от операции =~ тем, что возвращает противоположное логическое значение. Например, в результате поиска в строке "aaabbbccc" образца /b+/Г

```
$s="aaabbbccc" =~ m/b+/; $s="aaabbbccc" !~ m/b+/;,
```

в обоих случаях будет найден фрагмент ььь. Но в первом случае возвращаемое значение, сохраненное в переменной $s, будет равно 1 (ИСТИНА), а во втором случае – пустой строке " (ЛОЖЬ).

Образец PATTERN может содержать переменные, значения которых подставляются при каждом выполнении поиска по данному образцу.

Флаги cgimosx модифицируют выполнение операции поиска. Флаги imsx имеют тот же смысл, что и в рассмотренном выше случае конструкции расширенного регулярного выражения (?imsx-imsx) .

- i – поиск без учета регистра;
- m – трактуется как мульти-строка, состоящая из нескольких строк, разделенных символом новой;
- s – строка трактуется как одна строка, в этом случае метасимволу "." со-, ответствует любой одиночный символ, включая символ новой строки;
- x – разрешается использовать в образцах пробелы и комментарии.
- стальные флаги выполняют следующие функции.
- g – Задает глобальный поиск образца в заданной строке. Это означает, что будут найдены все фрагменты текста, удовлетворяющие образцу, а не только первый из них, как имеет место по умолчанию. Возвращаемое значение зависит от контекста. Если в составе образца имеются подобразцы, заключенные в скобки (), то в контексте массива для каждого заключенного в скобки подобразца возвращается список всех найденных фрагментов. Если в составе образца нет подобразцов, заключенных в скобки, то в контексте массива возвращается список всех найденных фрагментов, удовлетворяющих образцу. В скалярном контексте каждая операция m//g осуществляет поиск следующего фрагмента, удовлетворяющего образцу, возвращая значение 1 (ИСТИНА), если он найден, и пустую строку ", если не найден. Позиция строки, в которой завершился последний поиск образца при установленном флаге g, может быть получена при помощи встроенной функции роз о (см. ниже). Обычно при неудачном поиске начальная позиция поиска восстанавливается в начало строки. Флаг с отменяет восстановление начальной позиции поиска при неудачном поиске образца.

Рассмотрим следующий скрипт.

```
$str="abaabbaaabbbaaaabbbb";
tt контекст массива, нет подобразцов в скобках
@result=$str =~m/a+b+/g;
print "контекст массива, нет конструкций в скобках:\n";
print "\@result=@result\n";
# контекст массива, есть конструкции в скобках, задающие обратные ссылки
9result=$Str =~m/(a+)(b+)/g;
print "контекст массива, есть конструкции в скобках:\n";
print "\@result=@result\n";
# скалярный контекст
print "скалярный контекст:\n";
while ($result=$str =~m/(a+)(b+)/g) {
print "result=$result, current match is $&, position=",pos($str),"\n"; }
```

Результатом его выполнения будет вывод:

```
контекст массива, нет конструкций в скобках:
@result=ab aabb aaabbb aaaabbbb
контекст массива, есть конструкции в скобках:
@result=a b aa bb aaa bbb aaaa bbbb
скалярный контекст:
result=l, current match is ab, position=2
result=l, current match is aabb, position=6
result=l, current match is 'aaabbb, position=12
result=l, current match is aaaabbbb, position=20
```

Используется совместно с флагом g. Отменяет восстановление начальной позиции поиска при неудачном поиске образца.

Рассмотрим скрипт

```
$str="abaabbaaabbbaaaabbbb";
while ($result=$str =~m/(a+)(b+)/g) {
print "result=$result, current match is $&, position=",pos($str),"\n";
} · print "last position=", pos($str), "\n";
```

Здесь поиск образца /a+ь+/ в строке $str осуществляется в цикле до первой неудачи. При последнем (неудачном) поиске начальная позиция поиска по умолчанию устанавливается в начало строки, в этом случае вывод имеет вид:

```
result=l, current match is ab, position=2 result=l, current match is aabb, position=6
result=l, current match is aaabbb, position=12 result=l, current match is aaaabbbb,
position=20 last position=
```

Если глобальный поиск осуществлять при установленном флаге c:

```
while ($result=$str =~m/ (a+) (b+)/gc) {
```

то при последнем неудачном поиске начальная позиция поиска не переустанавливается. Вывод имеет вид:

```
result=l, current match is ab, position=2 result=l, current match is aabb, position=6 result=l,
current match is aaabbb, position=12 result=l, current match is aaaabbbb, position=20 last
position=20
```

При задании образца для глобального поиска m//g можно использовать ме-тапоследовательность \c, представляющую точку, в которой закончился последний поиск m//g. Например, в результате выполнения скрипта .

```
^^X
$str="l) abc 2) aabbcc 3) aaabbbccc 4) aaaabbbbcccc";
$str=~m/3\)\s+/g; \
! $str=~m/\Ga+/; ,'
```

сначала по образцу будет найден фрагмент "3)", а затем фрагмент, удовлетворяющий образцу /a+/ и расположенный сразу за точкой, в которой завершился последний поиск. Этим фрагментом является "aaa".

По– Значения переменных, входящих в состав образца PATTERN, подставляются только один раз, а не при каждом поиске по данному образцу. Рассмотрим, например, следующий скрипт:

```
@pattnlist=("a+", "Ы-", "c+", "d+") ; foreach $pattn (@pattnlist) (
$line = <STDIN>; $line =~ m/$pattn/o;
print "pattn=$pattn \$&= $&\n"; }
```

Массив gpattniist содержит список образцов "a+", "ь+", "c+" и "d+". В цикле по элементам этого списка в переменную $iine считывается очередная строка из стандартного ввода. В ней осуществляется поиск по образцу, совпадающему с текущим элементом списка. Поскольку использован флаг o, подстановка значений в образце /$pattn/ будет осуществлена один раз за время жизни данной Perl-программы, т. е. в качестве образца на каждом шаге цикла будет использовано выражение "a+". Если операцию поиска осуществлять без флага o:

```
$line =~ m/$pattn/,
```

то в качестве образца будут последовательно использованы все элементы списка "a+", "ь+", "c+" и "d+".

В качестве символов-ограничителей для выделения образца можно использовать любую пару символов, не являющихся цифрой, буквой или пробельным символом. Если в качестве ограничителя используется символ "/", то литеру m в обозначении операции можно опустить и использовать упрощенную форму /PATTERN/ ВМеСТО m/PATTERN/.

Если в качестве ограничителя используется одинарная кавычка ', то подстановка значений переменных внутри образца не производится.

Если в качестве ограничителя используется символ "?": ?PATTERN?, то при применении операции поиска находится только одно соответствие. Например, в результате выполнения скрипта

```
$str="abaabbaaabbbaaaabbbb";
while ($result = $str =~ m?a+b+?g) (
print "result=$result, current match is $&, position=", pos($str),"\n";
}
```

будет найдено только первое соответствие образцу:

```
result=l, current match is ab, position=2
```

Операция замены

```
s/PATTERN/REPLACEMENT/egimosx
```

Операция замены S/PATTERN/REPLACEMENT/ осуществляет поиск образца PATTERN и, в случае успеха, замену найденного фрагмента текстом REPLACEMENT. Возвращаемым значением является число сделанных замен или пустая строка (ЛОЖЬ), если замен не былоТПо умолчанию поиск и замена осуществляются в специальной переменной $_. Ее можно заменить другой строкой при помощи операций связывания =~ или ! ~:

```
$var =~ s/PATTERN/REPLACEMENT/egimosx
```

Флаг *g* задает глобальную замену всех фрагментов, удовлетворяющих образцу PATTERN,ТеКСТОМ REPLACEMENT.

Флаг e означает, что заменяющий текст REPLACEMENT следует рассматривать как Perl-выражение, которое надо предварительно вычислить. Например, в результате выполнения скрипта

```
$str="abaabbaaabbbaaaabbbb"; $result=$str =~s[ (a+b+)]<length($1)>ge; print "result=$result
new str=$str\n";
```

будет выведено число сделанных замен $ result и новое значение строки $str, в которой каждый найденный фрагмент, соответствующий образцу [a+ь+], заменен числом, равным его длине:

```
result=4 new str=2468
```

Флаги imosx имеют тот же смысл, что для операции поиска т//.

Так же, как и в операции замены, в качестве ограничителей для выделения образца можно использовать любую пару символов, не являющихся цифрой, буквой или пробельным символом. Можно использовать различные ограничители для выделения образца и замещающего текста, например,

```
s(pattern)<replacement>.
```

Операция транслитерации

```
tr/SEARCHLIST/REPLACEMENTLIST/cds
```

Преобразует каждый символ из списка поиска SEARCHLIST в соответствующий символ из списка замены REPLACEMENTLIST и возвращает число преобразованных символов. По умолчанию преобразования осуществляются в строке, задаваемой переменной $_. Как и в рассмотренных выше операциях поиска и замены, при помощи операций связывания =~ и ! ~ можно задать для преобразования строку, отличную от принятой по умолчанию

```
$str =~ tr/SEARCHLIST/REPLACEMENTLIST/cds
```

Списки SEARCHLIST и REPLACEMENTLIST задаются перечислением символов, могут содержать диапазоны – два символа, соединенных знаком "-", и иметь собственные символы-ограничители, например, tr(a-j) /0-9/. Операция tr/// имеет синонимичную форму, используемую в потоковом редакторе sed:

```
y/SEARCHLIST/REPLACEMENTLIST/cds
```

Флаги cds имеют следующий смысл.

- c – вместо списка поиска SEARCHLIST использовать его дополнение до основного множества символов (обычно расширенное множество ASCII).
- d – удалить все символы, входящие в список поиска SEARCHLIST, для которых нет соответствия в списке замены REPLACEMENTLIST. Если флаг d не установлен и список замены REPLACEMENTLIST короче, чем список поиска SEARCHLIST, то вместо недостающих символов в списке замены используется последний символ этого списка. Если список замены пуст, то символы из списка поиска SEARCHLIST преобразуются сами в себя, что удобно использовать для подсчета числа символов в некотором классе. П s – все последовательности символов, которые были преобразованы в один и тот же символ, заменяются одним экземпляром этого символа.

```
$str =~ tr/A-Z/a-z/;
# преобразование прописных букв в строчные $count=$str=~tr/\000//c;
# подсчет числа символов в строке $str $str =~ tr/\200-\377/ /cs;
# любая последовательность символов с ASCII-кодами от 128 до 255 преобразуется в единственный пробел
```

Следующий скрипт преобразует русский текст в DOS-кодировке 866, содержащийся в файле "866.txt", в русский текст в Windows-кодировке 1251, и записывает преобразованный текст в файл "1251.txt".

```
open(IN866, "866.txt"); open(OUT1251,">125I.txt"); while ($line=<IN866>) { .
$line=~tr/\200-\257\340-\361/\300-\377\250\270/; print OUT1251 $line;
>
close(IN866); close(OUT1251);
```

Операция заключения в кавычки *qr//*

```
qr/STRING/imosx
```

Операция qr// по своему синтаксису похожа^наГдругие операции заключения в кавычки, такие как q//, qq//, qx//, qw//. Она обсуждается в данном разделе, так как имеет непосредственное отношение к регулярным выражениям. Регулярное выражение, содержащее переменные, метасимволы, мета-последовательности, расширенный синтаксис, перед использованием должно быть обработано компилятором. Операция qr// осуществляет предварительную компиляцию регулярного выражения STRING, преобразуя его в некоторое внутреннее представление, с тем, чтобы сохранить скомпилированное регулярное выражение в переменной, которую затем можно использовать без повторной компиляции самостоятельно или в составе других регулярных выражений.

Преимущества от применения операции qr// проявляются, например, в следующей ситуации. Допустим, что мы собираемся многократно использовать в качестве образца достаточно сложное регулярное выражение, например, /Л([Л]*) *([Л]*)/. Его можно использовать непосредственно в операции сопоставления с образцом

```
if ($line =~ /Л(Г ]*) *([Л ]*)/) {...},
```

или сохранить в переменной $pattern=ll/4 ([Л]*) *([А]*) и обращаться к переменной:

```
if ($line =~ /$pattern/) (...},
```

В обоих случаях регулярное выражение при каждом обращении обрабатывается компилятором, что при многократном использовании увеличивает время выполнения. Если сохранить образец при помощи операции qr//:

```
$pattn = qr/~(r ]*) *<Г ]*)/,
```

то переменная $pattn будет содержать откомпилированное регулярное выражение, которое можно неоднократно использовать без дополнительной компиляции.

Флаги imosx имеют тот же смысл, что и в операции замены т//. Например, в следующем тексте операция qr// применяется с флагами ox:

```
$s="aA!Bb2cC3Dd45Ee";
@pattns=("\\d+ # последовательность цифр",
"[A-Z]+ t последовательность прописных букв", "[a-z]+ # последовательность строчных букв");
foreach $pattn Opattns) { my $pattern=qr/$pattn/ox; while ($s=~/$pattern/g) { $p=$p.$&; . } }
print "s=$s p=$p\n";
```

В данном примере определен массив @pattns, состоящий из регулярных выражений. В цикле по элементам массива проверяется наличие в заданной строке $з фрагмента, соответствующего текущему образцу. Найденный фрагмент добавляется в конец строки $p. Флаг x в операции qr// позволяет использовать образцы в том виде, в каком они хранятся в массиве – с пробелами и комментариями. Если в операции qr// флаг o не установлен, то в результате выполнения скрипта строка $p будет состоять из символов строки $s, расположенных в следующем порядке: сначала все цифры, затем все прописные буквы, затем все строчные буквы. Если, как в данном ^тексте, флаг o установлен, то в операции $pattern=qr/$pattn/ox подстановка переменной $pattn произойдет только один раз, и строка $s будет три раза проверяться на наличие фрагмента, удовлетворяющего первому образцу $pattns[i]. В результате строка $p будет состоять только из цифр, входящих в строку $s, повторенных три раза.

Функции для работы со строками

В данном разделе мы рассмотрим некоторые встроенные функции языка Perl, предназначенные для работы со строками текста. Часть из них использует рассмотренное выше понятие регулярного выражения.

Функция

```
chop () chop [list]
```

удаляет последний символ из всех элементов списка list, возвращает последний удаленный символ. Список может состоять из одной строки. Если аргумент отсутствует, операция удаления последнего символа применяется к встроенной переменной $_. Обычно применяется для удаления завершающего символа перевода строки, остающегося при считывании строки из входного файла.

Функция

```
length() length EXPR
```

возвращает длину скалярной величины EXPR в байтах.

```
#!/usr/bin/perl \^_____,.
$input = <STDIN>;
$Len = length($input);
print "Строка до удаления последнего символа: $input\n";
print "Длина строки до удаления последнего символа: $Len\n";
$Chopped = chop($input};
$Len = length($input);
print "Строка после удаления последнего символа: $input\n";
print "Длина строки после удаления последнего символа: $Len\n";
print "Удаленный символ: <$Chopped>\n";
```

Если после запуска данного скрипта ввести строку "qwerty", то вывод будет иметь вид:

```
qwerty
Строка до удаления последнего символа: qwerty
Длина строки до удаления последнего символа: 7
Строка после удаления последнего символа: qwerty
Длина строки после удаления последнего символа: 6
Удаленный символ: < >
```

Последним символом, удаленным функцией chop (), является символ новой строки, сохраненный в переменной $chopped. При выводе он вызывает переход на следующую строку, поэтому в данном выводе третья строка – пустая. В последней операции print вывод осуществляется в две строки, так как переменная $chopped содержит символ новой строки.

Функции

```
lc(), uc(), Icfirst(), ucfirstO
```

предназначены для преобразования строчных букв в прописные и наоборот.

Функция

```
1c EXPR
```

возвращает выражение, полученное из выражения EXPR преобразованием всех символов в строчные.

Функция

```
UC EXPR
```

возвращает выражение, полученное из выражения EXPR преобразованием всех символов в прописные.

Функция

```
Icfirst EXPR
```

возвращает выражение, полученное из выражения EXPR преобразованием первого символа в строчный.

Функция

```
ucfirst EXPR
```

возвращает выражение, полученное из выражения EXPR преобразованием первого символа в прописной.

```
#!/usr/bin/perl
print "\n Ф-ция uc() преобразует ",$s="upper case"," в ",uc $s;
print "\n Ф-ция ucfirst() преобразует ",$s="uPPER CASE"," в ",ucfirst $s;
print "\n Ф-ция 1c() преобразует ", $s="LOWER CASE"," в ",1c $s;
print "\n ф-ция IcfirstO преобразует ",$s="Lower case"," в ",lcfirst $s;
```

В результате выполнения данного скрипта будут напечатаны строки:

```
Ф-ция uc() преобразует upper case в UPPER CASE
Ф-ция ucfirst() преобразует UPPER CASE в UPPER CASE
Ф-ция 1c() преобразует LOWER CASE в lower case
Ф-ция IcfirstO преобразует Lower case в lower case
```

Функция

```
join() join EXPR, LIST
```

объединяет отдельные строки списка LIST в одну, используя в качестве разделителя строк значение выражения EXPR, и возвращает эту строку.

Функция

```
split ()
split [/PATTERN/[, EXPR[, LIMIT]]]
```

разбивает строку EXPR на отдельные строки, используя в качестве разделителя образец, задаваемый регулярным выражением PATTERN. В списковом контексте возвращает массив полученных строк, в скалярном контексте — их число. Если функция split о вызывается в скалярном контексте, выделяемые строки помещаются в предопределенный массив @_. Об этом не следует забывать, так как массив §_ обычно используется для передачи параметров в подпрограмму, и обращение к функции split о неявно в скалярном контексте эти параметры уничтожит.

Если присутствует параметр LIMIT, то он задает максимальное количество строк, на которое может быть разбита исходная строка. Отрицательное значение параметра LIMIT трактуется как произвольно большое положительное число.

Если параметр EXPR опущен, разбивается строка $_. Если отсутствует также и параметр PATTERN, то в качестве разделителей полей используются пробельные символы после пропуска всех начальных пробельных символов (что соответствует заданию образца в виде /\s+/). К пробельным символам относятся пробел (space), символ табуляции ЛаЬ), возврат каретки (carriage feturn), символ перевода строки (line feed) и^симврл перевода страницы (form feed). .

Замечание

Предопределенная глобальная переменная $_ служит для обозначения используемой по умолчанию области ввода и поиска по образцу. Обычно мы осуществляем ввод при помощи операции "0" ("ромб"). Внутри угловых скобок о может стоять дескриптор файла ввода, например, <STDIN>. Если дескриптор файла отсутствует, то в качестве файлов ввода используются файлы, переданные программе Perl *в* качестве аргументов командной строки. Пусть, например, программа содержится в файле script. pi.

```
#!/usr/bin/perl while (<>) {
print; };
```

Программа вызвана следующим образом

```
script.pl filel file2 file3
```

Тогда операция о будет считывать строки сначала из файла filel, затем из файла file2 и, наконец, из файла file3. Если в командной строке файлы не указаны, то в качестве файла ввода будет использован стандартный ввод.

Только *в* случае, когда условное выражение оператора while состоит из единственной операции "ромб", вводимое значение автоматически присваивается предопределенной переменной $_. Вот что означают слова о том, что переменная $_ применяется для обозначения используемой по умолчанию области ввода. Аналогично обстоит дело с поиском по образцу.

```
I!/usr/bin/perl while (<>} {
chop;
print "Число полей во входной строке '$_' равно ", $n=split;
print "ХпВходная строка разбита на строки:\n";
foreach $i (@_) {
print $i . "\n"; }
print "Объединение списка строк в одну строку через ' +':\n";
$joined = join "+", @_;
print "$joined\n"; }
```

В результате применения операции ввода 0 внутри условного выражения оператора while вводимая строка будет присвоена переменной $_. Функция chop о без параметров применяется к переменной $_. В операции print вторым операндом является выражение $n=spiit, в котором функция split вызывается в скалярном контексте и без параметров. Поэтому она применяется по умолчанию к переменной $_. В качестве разделителей полей по умолчанию используется множество пробельных символов, а результат помещается в масссив @_. Затем к массиву @_ применяется функция joint), объединяющая строки-элементы массива в одну строку.

Если ввести строку "one two three", то вывод будет иметь вид:

```
one two three
Число полей во входной строке 'one two three' равно 3
Входная строка разбита на строки:
one
two
three
Объединение списка строк в одну строку через '+':
one+two+three
```

Функция

```
index()
index STR, SUBSTR[, POSITION]
```

находит первое, начиная с позиции POSITION, вхождение подстроки SUBSTR в строку STR, и возвращает найденную позицию. Если параметр POSITION не задан, по умолчанию принимается значение POSITION = $[. Если подстрока SUBSTR не найдена, возвращается значение $ [- 1.

Замечание

Предопределенная переменная $ [содержит индекс первого элемента в массиве и первого элемента в строке. По умолчанию ее значение равно 0. В принципе его можно изменить, но делать это не рекомендуется. Таким образом, по умолчанию значение параметра POSITION полагается равным 0, а функция index возвращает-1, если не найдена подстрока SUBSTR.

Функция

```
rindex{}
rindex STR, SUBSTR, POSITION
```

находит последнее, ограниченное справа позицией POSITION, вхождение подстроки SUBSTR в строку STR, и возвращает найденную позицию. Если подстрока SUBSTR не найдена, возвращается значение $ [- i.

```
f!/bin/perl · ,
$STR = "Этот безумный, безумный, безумный, безумный мир!";
$ SUBSTR = "безумный"; ^~– --’’
$POS = 7;
print "Индекс первого символа строки по умолчанию равен $[\n";
print "Позиция первого вхождения подстроки '$SUBSTR'
в строку '$STR' = ",index($STR, $SUBSTR), "\n"; print "Позиция первого после позиции $POS
вхождения подстроки '$SUBSTR'
в строку '$STR' = ",index($STR, $SUBSTR, $POS), "\n"; print "Позиция последнего вхождения
подстроки '$SUBSTR'
в строку '$STR [1] = ",rindex($STR, $SUBSTR), "\n"; print "Позиция последнего перед позицией $POS
вхождения подстроки '$SUBSTR'
в строку '$STR' = ",rindex($STR, $SUBSTR, $POS), "\n";
$[=2;
print "ХпИндекс первого символа строки по умолчанию изменен на $[\n"; print "Позиция первого
вхождения подстроки '$SUBSTR'
в строку '$STR' = ",index($STR, $SUBSTR), "\n"; print "Позиция первого после позиции $POS
вхождения подстроки '$SUBSTR'
в строку '$STR' = ",index($STR, $SUBSTR, $POS), "\n";
print "Позиция последнего вхождения подстроки '$SUBSTR' в строку '$STR' = ",rindex($STR,
$SUBSTR), "\n";
print "Позиция последнего перед позицией $POS вхождения подстроки '$SUBSTR' в строку '$STR' =
",rindex($STR, $SUBSTR, $POS), "\n";
```

В результате выполнения скрипта будут выведены следующие строки:

```
Индекс первого символа строки по умолчанию равен 0 Позиция первого вхождения подстроки
'безумный'
в строку 'Этот безумный, безумный, безумный, безумный мир!' = 5 Позиция первого после позиции
7 вхождения подстроки 'безумный'
в строку 'Этот безумный, безумный, безумный, безумный мир!' = 15 Позиция последнего вхождения
подстроки 'безумный'
в строку 'Этот безумный, безумный, безумный, .безумный мир!' = 35 Позиция последнего перед
позицией 7 вхождения подстроки 'безумный'
в строку 'Этот безумный, безумный, безумный, безумный мир!' = 5
Индекс первого символа строки по умолчанию изменен на 2 Позиция первого вхождения подстроки
'безумный'
в строку 'Этот безумный, безумный, безумный, безумный мир!' = 7 Позиция первого после позиции
7 вхождения подстроки 'безумный'
```

```
в строку 'Этот безумный, безумный, безумный, безумный мир!' = 7 Позиция последнего вхождения подстроки 'безумный'
в строку 'Этот безумный, безумный, безумный, безумный мир!' = 37 Позиция последнего перед позицией 7 вхождения подстроки 'безумный'
в строку 'Этот безумный, безумный, безумный, безумный мир!' = 7
```

Функция

```
substr ()
substr EXPR, OFFSET [,LENGTH [,REPLACEMENT ]]
```

извлекает из выражения EXPR подстроку и возвращает ее. Возвращаемая подстрока состоит из LENGTH символов, расположенных справа от позиции OFFSET. Если параметр LENGTH опущен, возвращается вся оставшаяся часть выражения EXPR. Если параметр LENGTH отрицательный, его абсолютное значение задает количество символов от конца строки, не включаемых в результирующую подстроку. Если параметр OFFSET имеет отрицательное значение, смещение отсчитывается с конца строки. Функция substr о может стоять в левой части операции присваивания. Например, в результате выполнения операторов

```
$Str = "Язык Pascal"; substr($Str, 5,6) = "Perl";
```

переменная $str получит значение "язык Perl". Тот же результат будет достигнут, если указать параметр REPLACEMENT, значение которого будет подставлено в EXPR вместо выделенной подстроки. Сама подстрока в этом случае возвращается в качестве значения функции substr ().

```
#!/bin/perl
# Исходная строка
$Str = "Карл у Клары украл кораллы";
$0ffset = 13;
print "Исходная строка:'$Str'\n";
# Смещение 13, длина подстроки не задана
$Substr = substr $Str, $Offset;
print "Смещение $0ffset, длина подстроки не задана, результат:\n";
print "$Substr\n";
# Смещение 13, длина подстроки +5
$Substr = substr $Str, $0ffset, 5;
print "Смещение $0ffset, длина подстроки +5, результат:\n";
print "$Substr\n"; /
# Смещение 13, длина подстроки -1 \
$Substr = substr $Str, $0ffset, -1;
print "Смещение $0ffset, длина подстроки -1, результат:\n";
```

```
print "$Substr\n";
# Отрицательное смещение -7, длина подстроки +7
$0ffset = -7;
$Substr = substr $Str, $Offset, 7;
print "Отрицательное смещение $Offset, длина подстроки +7, результат:\n";
print "$Substr\n";
f Отрицательное смещение -7, длина подстроки -1
$Substr = substr $Str, $0ffset, -1;
print "Отрицательное смещение $Offset, длина подстроки -1, результат:\n";
print "$Substr\n";
t Замена подстроки
$Repl = "бокалы";
$Substr = substr $Str,$Offset,7,$Repl;
print "В строке '$Str' слово '$Repl' заменяет слово '$Substr'\n";
Вывод выглядит следующим образом:
Исходная строка:'Карл у Клары украл кораллы'
Смещение 13, длина подстроки не задана, результат:
украл кораллы
Смещение 13, длина подстроки +5, результат:
украл
Смещение 13, длина подстроки -1, результат:
украл коралл
Отрицательное смещение -7, длина подстроки +7, результат:
кораллы
Отрицательное смещение -7, длина подстроки -1, результат:
коралл
В строке 'Карл у Клары украл бокалы' слово 'бокалы' заменяет слово 'кораллы'
```

Функция

```
eval () eval EXPR
```

рассматривает параметр EXPR как текст Perl-программы, компилирует его и, если не обнаруживает ошибок, выполняет в текущем вычислительном окружении. Если параметр EXPR отсутствует, вместо него по умолчанию используется глобальная переменная $_. Компиляция программного кода EXPR осуществляется при каждом вызове функции eval () во время выполнения основной программы. Если выполнение мини-программы EXPR завершается успешно, функция eval о возвращает значение последнего выражения, вычисленного внутри EXPR. Если код EXPR содержит синтаксические ошибки, или обращение к функции die (}, или возникла ошибка во время выполнения EXPR, то в специальную

переменную $@ помещается сообщение об ошибке, а функция eval () возвращает *неопределенное значение.*

Замечание

Основным применением функции eval о является перехватывание исключений. *Исключением* мы называем ошибку, возникающую при выполнении программы, когда нормальная последовательность выполнения прерывается (например, при делении на нуль). Обычной реакцией на исключение является аварийное завершение программы. Язык Perl предоставляет возможность перехватывать исключения без аварийного выхода. Если исключение возникло в основной программе, то программа завершается. Если ошибка возникла внутри мини-программы функции eval (), то аварийно завершается только функция eval(), а основная программа продолжает выполняться и может реагировать на возникшую ошибку, сообщение о которой ей доступно через переменную $@.

В следующем примере функция eval о применяется для перехватывания ошибки, связанной с делением на 0 при вычислении функции ctg(x). Используются встроенные функции sin, cos и warn. Последняя функция осуществляет вывод сообщения, задаваемого ее аргументом, на стандартное устройство вывода ошибок STDERR.

```
#!/bin/perl
$fi = 0.314159265358979;
$f = '$ctg = cos($x)/sin($x)
for $i (0..10) {
$x = $i*$fi;
eval $f;
print "x= $x ctg(x) = $ctg\n" if defined $ctg;
warn "x= $x ", $@ if not defined $ctg; };
```

Вывод программы выглядит следующим образом

```
x= 0 Illegal division by zero at (eval 1) line 1. x= 0.314159265358979 ctg(x)
=3.07768353717526 x= 0.628318530717958 ctg(x) = 1.37638192047118
```

Замечание

Иногда бывает полезно искусственно вызвать исключительную ситуацию. Для этого можно воспользоваться функцией die () LIST. Назначение функции die () — генерировать исключения. Если функция die () вызывается в основной программе вне функции eval (), то она осуществляет аварийное завершение основной программы и выводит сообщение об ошибке LIST на стандарт* ное устройство вывода ошибок STDERR. Если она вызывается внутри функции eval (), то осуществляет аварийное завершение eval () и помещает сообщение об ошибке в специальную переменную $@.

Функция

```
pos()
```

```
pos [$SCALAR]
```

возвращает позицию, в которой завершился последний глобальный поиск $scALAR=~m/.../g, осуществленный в строке, задаваемой переменной $SCALAR. Возвращаемое значение равно числу length($') + length($&). Следующий глобальный поиск m/.../g в данной строке начнется именно с этой позиции.

Если аргумент $ SCALAR отсутствует, возвращается позиция завершения последнего глобального поиска, осуществленного в строке $_.

```
$words = "one two three four"; while ($words =~ m/\w+/g) {
print "pos=",pos($words)," length(\$~)=",length($'),
" length(\$s)=",length($s),"\n"; }
```

В результате выполнения данного скрипта будут выведены номера позиций, соответствующих окончаниям слов в строке $words:

```
pos=3 length($~}=0 length($&)=3
pos=7 length{$')=4 length($&)=3
pos=13 length($~)=8 length($&)=5
pos=18 length($')=14 length($&)=4
```

Функцию pos () можно использовать в левой части операции присваивания для изменения начальной позиции следующего поиска:

```
I изменение начальной позиции для последующего поиска
$words = "one two three four";
pos $words = 4;
while ($words =~ m/\w+/g) {
print pos $words, "\n"; }
```

В последнем случае поиск слов начнется со второго слова, и будут выведены номера позиций 7, 13 и 18.

Функция

```
quotemeta () quotemeta [EXPR]
```

возвращает строку EXPR, в которой все символы, кроме алфавитно-цифровых символов и символа подчеркивания "_", экранированы символом "\". Например, в результате выполнения

```
print quotemeta "*****", "\n";
```

будет выведена строка

```
\*\*\*\*\*
```

Если аргумент EXPR отсутствует, вместо него используется переменная $_.

Вопросы для самоконтроля

1. Что такое регулярное выражение?

2. Какие символы имеют в регулярном выражении Perl специальное значение?

3. Что такое метапоследовательность, как она образуется?

4. Что такое обратная ссылка?

5. Какая переменная используется в операции подстановки по умолчанию?

6. Какой смысл имеет символ "$" в следующих регулярных выражениях:

```
/abc*$/
/[abc*$]/
/$abc/
```

7. Какой смысл имеет символ "" в следующих регулярных выражениях: A

```
/ ^ abc/
/[abc]/
/abc ^ /
```

8. Объясните, какие множества строк соответствуют следующим образцам. Приведите пример.

```
/a.out/
/a\.out/
/\d{2,3}-\d{2}-\d{2}/
/(.)(.).\2\1/ /(.) (.).\02\01/
```

9. Напишите образец, задающий палиндром из шести букв.

10. Напишите команду замены, которая:

• заменяет все символы новой строки пробелами;

• выделяет из полного маршрутного имени файла имя файла;

• выделяет из полного маршрутного имени файла имя каталога.

11. Каково значение следующих выражений, если значение переменной

```
$var равно "123qwerty"? $var =~ /./ $var =- /[A-Z]*/ $var =~ /\w{4-9}/ $var =~ /(\d)2(\D/ $var
=~ /qwerty$/ $var =~ /123?/
```

12. Какое значение будет иметь переменная $var после следующих операций подстановки, если ее начальное значение равно "qwertyi23qwerty"?

```
$var =~ s/qwerty/XYZ/; $var =~ s/[a-zJ+/X/g; $var =~ s/B/W/i; $var =~ s/(.)\d.*\l/d/; $var =~ s/(\d+)/$l*2/e;
```

13. Начальное значение переменной $var равно "qwertyi23qwerty". Каким оно будет после выполнения операций транслитерации?

```
$var =~ tr/a-z/A-Z/; $var =~ tr/a-z/0-9/; $var =~ tr/a-z/O-9/d; $var =~ tr/231/564/; $var =~ tr/123/ /s; . $var =~ tr/123//cd;
```

14. Переменная $var имеет значение "qwertyqwerty". Каково значение, возвращаемое функцией?

```
substr ($var, 0, 3);
substr ($var, 4);
substr ($var, -2, 2);
substr ($var, 2, 0) ;
index ($var, "rt"); index ($var, "rtyu"); index ($var, "er", 1); index ($var, "er", 7); rindex ($var, "er");
```

Упражнения

1. Напишите программу, которая читает стандартный ввод, умножает каждое встретившееся число на 2 и выводит результирующую строку.

2. Напишите программу, которая читает стандартный ввод, удваивает каждую букву и выводит результирующую строку.

3. Напишите программу, подсчитывающую, сколько раз каждый алфавитно-цифровой символ встретился во вхбдном файле.

4. Напишите программу, которая считывает строку из стандартного файла ввода, меняет в ней порядок следования символов на обратный и выводит результат.

5. Напишите программу, которая выполняет преобразование русского текста из одной системы кодировки в другую:

```
(Dos 866, Windows 1251, UNIX KOI8} <=> (Dos 866, Windows 1251, UNIX, KOI8}
```

Для выполнения задания можно воспользоваться табл. 10.3, содержащей шестнадцатеричные коды символов русского алфавита.

Таблица 10.3. Таблицы кодов русского алфавита

Символ	866	1251	KOI8	Символ	866	1251	KOI8
А	80	C0	E1	а	A0	E0	C1
Б	81	C1	E2	б	A1	E1	C2
В	82	C2	F7	в	A2	E2	D7
Г	83	C3	E7	г	A3	E3	C7
Д	84	C4	E4	д	A4	E4	C4
Е	85	C5	E5	е	A5	E5	C5
Ё	F0	A8	B3	е	F1	B8	A3
Ж	86	C6	F6	ж	A6	E6	D6
З	87	C7	FA	з	A7	*E7*	DA
И	88	C8	E9	и	A8	E8	C9
И	89	C9	EA	й	A9	E9	CA

К	8A	CA	EB	к	AA	EA	CB
Л	8B	CB	EC	л	AB	EB	CC
М	8C	CC	ED	М	AC	EC	CD
Н	8D	CD	EE	Н	AD	ED	CE
О	8E	CE	EF	0	AE	EE	CF
П	8F	CF	FO	П	AF	EF	DO
Р	90	DO	F2	Р	EO	FO	D2
С	91	D1	F3	с	E1	F1	D3
Т	92	D2	F4	т	E2	F2	D4
У	93	D3	F5	У	E3	F3	D5
ф	94	D4	E6	ф	E4	F4	C6
Х	95	D5	E8	Х	E5	F5	C8

Ц	96	D6	E3	ц	E6	F6	C3
ч	97	D7	FE	‡	E7	F7	DE
ш	98	D8	FB	ш	E8	F8	DB
Щ	99	D9	FD	Щ	E9	F9	DD
ъ	9A	DA	FF	ъ	EA	FA	DF
ы	9B	DB	F9	ы	EB	FB	D9
ь	9C	DC	F8	ь	EC	FC	D8
э	9D	DD	FC	э	ED	FD	DC
ю	9E	DE	EO	ю	EE	FE	CO
я	9F	DF	F1	я	EF	FF	D1