Списки и хэши

- Третий тип данных — хэши (или ассоциативные массивы) — неупорядоченный список скаляров, к которым обращение происходит не по индексу, как в массиве, а по названию в виде символьной строки. Такой строковый индекс называют ключом, по которому осуществляется доступ к значению.
- Ключом хэша может быть любая скалярная величина: строка, ссылка, целое или дробное число, автоматически преобразуемое в строку.
- Причем значения всех ключей в хэше уникальны, поскольку внутренняя организация хэша не допускает ключей с одинаковыми значениями.
- Ассоциированное с ключом значение может быть любой скалярной величиной.
- Размер хэша в Perl ограничен только доступной программе памятью, поэтому хэши позволяют эффективно обрабатывать большие объемы данных.
- переменная, имеющая тип хэша, записывается с разыменовывающим префиксом % перед именем
- Переменная хэша:

```
%hash # переменная-хэш
```

Непосредственные величины ключей и значений хэша могут быть представлены в виде списочного литерала.
Каждый элемент в литерале состоит из двух частей: поискового ключа и связанного с ним значения, разделенных символами =>, например:

```
('версия' => 5.8, 'язык' => 'Perl') # ключ - строка
(3.14 => 'число Пи')                  # ключ - дробь
(1 => 'one', 2 => 'two', 3 => 'three') # ключ - целое
($key1 => $value1, $key2 => $value2) # ключ в переменной
```

- Операция => эквивалентна запятой, за исключением того, что она создает строковый контекст, так что ее левый операнд автоматически преобразуется к строке.

Именно поэтому числа в этом примере записаны без кавычек. Литеральные списки, содержащие ассоциативные пары, обычно применяются для присваивания хэшам начальных значений:

```
 %quarter1 = (1 => 'январь', 2 => 'февраль', 3 => 'март');
%dns = ($site => $ip, 'www.perl.com' => '208.201.239.36');
%empty = (); # пустой список удаляет все элементы хэша
```

- Если в качестве ключа хэша используется переменная с неопределенным значением, то оно преобразуется в пустую строку, которая и станет поисковым ключом.
- Значения ключей в хэше уникальны, поэтому хэш часто используется для моделирования множества или простой базы данных с уникальным поисковым индексом.
- При добавлении нескольких элементов с одинаковыми ключами в хэше остается только последний добавленный:

```
%num2word = (10 => 'десять', 5 => 'пять', 10 => 'ten');
# в %num2word останется только (5 => 'пять', 10 => 'ten')
```

- Ситуация, когда с поисковым ключом хэша ассоциируется неопределенное значение, считается нормальной. Это чаще всего означает, что связанное с ключом значение будет добавлено позднее.
- Начальные значения элементов хэша могут браться из любого списка, при этом значения нечетных элементов списка становятся в хэше ключами, а четных - ассоциированными с этими ключами значениями.

Так что два следующих присваивания эквивалентны:

```
 %dictionary = ('я' => 'I', 'он' => 'he', 'она' => 'she');
%dictionary = ('я', 'I', 'он', 'he', 'она', 'she');
```

- Для заполнения хэша элементами вместо списочного литерала можно использовать массив, содержащий пары "ключ - значение":

```
%dictionary = @list_of_key_value_pairs; # массив пар
```

В повседневной работе хэш заполняется данными из списка, который считывается из файла или генерируется при помощи пользовательской функции.

- Следует иметь в виду, что, в отличие от массивов, ***элементы в хэше не упорядочены***, и порядок следования элементов при добавлении элементов в хэш и при выборке их из хэша обычно не совпадает.
- Все значения, хранящиеся в хэше, можно преобразовать в список, если употребить переменную-хэш в списочном контексте в правой части операции присваивания:

```
@key_value_list = %hash; # список ключей и значений
```

- При этом в список будут помещены все ассоциативные пары из хэша, и **ключи станут нечетными элементами списка, а значения - четными. Порядок копирования в массив ассоциативных пар заранее не известен.**

При **обращении к элементу хэша** в фигурных скобках после имени переменной указывается значение поискового ключа. Поскольку значение элемента хэша — это скалярная величина, при обращении к элементу хэша перед именем переменной ставится префикс $, как у прочих скалярных значений.

```
$hash{$key} = $value; # добавление значения в хэш по ключу
$value = $hash{$key}; # извлечение значения из хэша по ключу
```

Примеры использования элементов хэша:

```
$month = 'January';
$days_in_month{$month}= 31; # со строкой связано число
$ru{$month}= 'январе'; # со строкой связана строка
print "В $ru{$month} $days_in_month{'January'} день";
```

В некоторых программах можно встретить при записи элементов хэша строковые ключи, не заключенные в кавычки: это допускается, если ключ — одно слово, записанное по правилам написания идентификаторов, так называемое "голое слово" ("bare word").
Имена хэшей компилятор располагает в другой таблице имен, чем имена массивов или скаляров, поэтому три приведенные ниже переменные абсолютно разные:

```
$variable # скалярная переменная
@variable # переменная-массив
%variable # переменная-хэш
```

Пример

Типичным применением хэша можно считать **составление частотного словаря**, в котором со значением каждого слова ассоциируется счетчик его появления в тексте. Для простоты предположим, что слова в файле, содержащем текст, разделены только пробелами:

```
while (my $line = <>) { # считать строку из входного потока
        chomp($line); # удалить из строки символ '\n'
        @words = split(' ', $line); # разбить строку на слова
        foreach my $word (@words) { # для каждого найденного слова
                $hash{$word}++; # увеличить счетчик
        }
} # теперь в %hash содержатся счетчики слов
```

Замечание. Позднее, в теме, посвященной регулярным выражениям, будет сказано, как выделять из строки слова не только по пробелам.

Как это было сделано в последнем примере, программисты часто пользуются уникальностью ключей в хэше, чтобы исключить дублирование данных.

- Для удаления из данных повторений достаточно поместить их в хэш в качестве ключей. При этом даже не обязательно ассоциировать с ключами какие-либо значения. В результате набор ключей хэша будет гарантированно содержать только неповторяющиеся значения из обработанного набора данных.

ФУНКЦИИ РАБОТЫ С ХЭШАМИ

При обработке данных в хэше часто возникает необходимость проверить **наличие в нем элемента с определенным ключом.**

- Функция ***exists*** проверяет, содержится ли указанный ключ в хэше.

Если ключ найден, она возвращает истинное значение ('1'), и ложное значение (пустую строку), если такого ключа в хэше нет. При этом ассоциированное с ключом значение не проверяется и может быть любым, в том числе и неопределенным.
Так можно проверить наличие ключа в хэше:

```
print "ключ $key найден" if exists $hash{$key};
```

- При помощи функции ***defined***(), возвращающей истинное или ложное значение, можно проверить, **было ли задано значение в элементе хэша, ассоциированное с указанным ключом, или оно осталось неопределенным**.

```
print "с ключом $key связано значение" if defined $hash{$key};
```

Проверка с помощью функции **defined**($hash{$key}) отличается от проверки значения элемента на истинность значения $hash{$key}, так как значение элемента может быть определено, но равно нулю или пустой строке, что тоже воспринимается как ложь.

- Воспользовавшись функцией ***undef***(), **можно удалить из хэша только значение элемента, не удаляя его ключа,** то есть сделать его неопределенным:

```
undef $hash{$key }; # сделать значение неопределенным
```

После того как значение элемента было удалено функцией undef(), **проверки наличия в хэше ключа и значения** указанного элемента хэша дадут следующие результаты:

```
$hash{$key} # неопределенное значение – это ложь
defined $hash{$key} # ложь, ибо значение не определено
exists $hash{$key} # истина, ибо ключ есть
```

Неопределенное значение, хранимое в элементе хэша, означает, что необходимый поисковый ключ присутствует, но с ним не ассоциировано никакого значения.

- **Добавление элементов в хэш выполняется *операцией присваивания*, а удаление — функцией *delete*.**

Эта функция по указанному элементу удаляет из хэша соответствующую пару "ключ - значение" и возвращает только что удаленное значение. Это делается так:

```
$deleted_value = delete $hash{$key}; # удалить элемент
```

- Если аргументом функции delete будет несуществующий элемент массива, то она просто вернет неопределенное значение, не вызвав ошибки при выполнении программы.

- ***Функция keys*** возвращает **список всех ключей хэша**. Полученный список можно сохранить в массиве для дальнейшей обработки:

```
@hash_keys = keys %hash; # поместить список ключей в массив
```

Возможно также использовать список ключей для доступа в цикле ко всем значениям хэша. Так можно напечатать частотный словарь из предыдущего примера:

```
foreach my $word (keys %hash) { # для каждого ключа хэша
     print "$word встретилось $hash{$word} раз\n"; }
```

Элементы хэша, как и другие скалярные величины, помещенные в обрамленную двойными кавычками строку, заменяются своими значениями.

Пример

Перепишем последний пример, добавив сортировку ключей для вывода слов в алфавитном порядке. А для организации цикла можно применить модификатор foreach, совмещающий очередной элемент списка с переменной по умолчанию $_:

```
print "$_ встретилось $hash{$_} раз\n" foreach (sort keys %hash);
```

Следует помнить, что **если размер хэша велик**, то и полученный с помощью функции keys массив **ключей тоже будет занимать большой объем памяти**.

- В скалярном контексте функция keys возвращает количество ключей в хэше, поэтому с ее помощью можно легко проверить, не пустой ли хэш:

```
if (keys %hash) { #т.е. если scalar(keys(%hash)) != 0
     # обработать элементы хэша, если он не пуст }
```

Пустой хэш в скалярном контексте возвращает ложное значение (строку '0'), а непустой - истинное. Поэтому проверить, пуст ли хэш, **можно еще проще — употребив имя хэша в скалярном контексте**, что часто используется в конструкциях, проверяющих условие:

```
while (%hash) { # или scalar(%hash) != 0 (не пуст ли хэш?)
     # обработать элементы хэша }
```

- Встроенная функция ***values*****, дополняющая функцию keys**, возвращает список всех значений элементов хэша в том же порядке, в каком функция keys возвращает ключи.

С полученным списком можно поступать обычным образом: например, сохранить в массиве или обработать в цикле:

```
@hash_values = values %hash; # сохранить все значения хэша
print "$_\n" foreach (values %hash); # вывести значения
```

- В скалярном контексте функция **values возвращает количество значений в хэше**, так что ее можно использовать для того, чтобы узнать размер хэша.

```
$hash_size = values %hash; # число значений в хэше
```

- Функция ***each*** является встроенным **итератором** — программной конструкцией, контролирующей последовательную обработку элементов какой-либо коллекции данных.

Она предоставляет возможность **последовательно обработать все ассоциативные пары в хэше**, организуя перебор всех его элементов. При каждом вызове она возвращает двухэлементный список, состоящий из очередного ключа и значения из хэша. Пары элементов возвращаются в неизвестном заранее порядке.

```
($key, $value) = each %hash; # взять очередную пару элементов
```

- После того как будет возвращена последняя пара элементов хэша, функция each возвращает пустой список. После этого следующий вызов each начнет перебор элементов хэша сначала.

Пример

С помощью `each` удобно организовать обработку всех элементов хэша в цикле, который закончится, когда each вернет пустой список, означающий "ложь":

```
while (my ($key, $value) = each %hash) { # пока есть пары
    # обработать очередные ключ и значение хэша
    print "с ключом $key связано значение $value\n"; }
```

- Иногда требуется **искать ключи хэша по их значениям**. Для этого нужно создать обратный ассоциативный массив (или инвертированный хэш), поменяв местами значения и ключи хэша.

Это можно сделать так:

```
while (my ($key, $value) = each %hash_by_key) { # ключи хэша
    $hash_by_value{$value} = $key; # становятся значениями }
```

Этого же результата можно достичь с помощью функции ***reverse***, воспользовавшись тем, что она воспринимает хэш как список, в котором за каждым ключом идет значение, и меняет порядок всех элементов этого списка на обратный.

- Функция ***reverse*** возвращает список, в котором в каждой паре элементов за значением следует ключ, и этот список присваивается новому хэшу:

```
%hash_by_value = reverse %hash_by_key; # переворот списка
$key = $hash_by_value{$value}; # поиск по бывшему значению
```

Нечетные элементы инвертированного списка становятся ключами, а четные — значениями хэша `%hash_by_value`.

----------НЕКОТОРЫЕ ПРИЕМЫ РАБОТЫ С ХЭШЕМ

1.

Так как весь хэш, его ключи или значения можно легко преобразовать **в список**, то для обработки хэшей можно применять **любые функции, работающие со списками**. Именно поэтому в предыдущем примере была применена функция ***reverse***. Например, вывести ключи и значения хэша на печать можно так:

```
{ # организовать блок, где объявить временный массив
    my @temp = %hash; # сохранить в нем хэш
    print "@temp"; # и передать его функции print
} # по выходе из блока временный массив будет уничтожен
```

2.

Можно напечатать хэш по-другому, построчно и в более облагороженном виде, при помощи функции ***map***, которая также выполняет роль итератора:

```
print map {"Ключ: $_ значение: $hash{$_}\n" } keys %hash;
```

В этом примере на основании списка ключей, возвращенного функцией keys, функция map формирует список нужных строк, вставляя из хэша в каждую из них ключ и значение. Она возвращает сформированный список функции print, которая выводит его в выходной поток.

Это типичный для Perl прием — обрабатывать данные при помощи цепочки функций, когда результат работы одной функции передается на обработку другой, как это принято делать с помощью конвейеров команд в операционных системах семейства Unix.

3.

В приведенных выше примерах при необходимости обработки ключей хэша в алфавитном порядке они **сортировались с помощью функции** ***sort***. Вот пример обработки хэша в порядке возрастания не его ключей, а его значений:

```
foreach $key ( # каждый элемент списка,
    sort # отсортированный по порядку
    {$hash{$a} cmp $hash{$b}}# значений, ассоциированных
    keys %hash) { # с ключами хэша
    print "значение:$hash{$key} ключ:$key\n"; # обработать } # в цикле
```

- Здесь в блоке сравнения функции sort сопоставляется значения хэша, ассоциированные с очередными двумя ключами из списка, который предоставлен функцией keys.

СРЕЗЫ ХЭШЕЙ

Срез хэша (hash slice) — **это список значений хэша, заданный перечнем соответствующих ключей**.

- записывается в виде имени хэша с префиксом @ (так как срез — это список), за которым в фигурных скобках перечисляются ключи.

Список ключей в срезе хэша можно задать перечислением скалярных значений, переменной-списком или списком, возвращенным функцией. Например, так:

```
@hash{$key3, $key7, $key1} # срез хэша задан списком ключей
@hash{@key_values} # срез хэша задан массивом
@hash{keys %hash} # то же, что values(%hash)
```

- Если в срезе хэша список ключей состоит из единственного ключа, срез все равно является списком, хотя и из одного значения. Сравните:

```
@hash{$key} # срез хэша, заданный списком из одного ключа $hash{$key}
# значение элемента хэша, заданное ключом
```

- !!! Поскольку переменная-хэш в составе строки не интерполируется, для вставки в строку всех значений хэша можно воспользоваться срезом хэша:

```
%hash = ('0' => 'false', '1' => 'true');
"@hash{keys %hash}"; # будет "false true" или "true false"
```

- Срез хэша, как и любой другой список, может стоять в левой части операции присваивания.
- При этом списку ключей среза должен соответствовать список присваиваемых значений в правой части присваивания.
- !!!Воспользовавшись срезом, можно добавить в хэш сразу несколько пар или объединить два хэша, добавив к одному другой.

Пример:

```
@hash{$k1, $k2, $k3}= ($v1, $v2, $v3); # добавить список
@old{keys %new}= values %new; # добавить хэш %new к %old
```

- С помощью среза хэша и функций keys и values можно **поменять в хэше местами ключи и значения**, то есть сделать значения ключами, а ключи — значениями.

```
@hash_keys = keys %hash; # сохранить ключи в массиве
@hash_values = values %hash; # сохранить список значений
%hash = (); # очистить хэш
@hash{@hash_values}=@hash_keys; # срезу хэша присвоить список
```

Специальные ассоциативные массивы

Исполняющая система Perl предоставляет программисту доступ к специальным ассоциативным массивам, в которых хранится полезная служебная информация. Вот некоторые из специальных хэшей:

- %***ENV*** перечень системных переменных окружения (например, PATH)
- %***INC*** перечень внешних программ, подключаемых по require или do
- %***SIG*** используется для установки обработчиков сигналов от процессов

Пример. При выполнении программы можно использовать значения переменных окружения: перечислить все их значения или выбрать нужные.

```
foreach my $name (keys %ENV) {
     print "$name=$ENV{$name}\n";
}
($who, $home) = @ENV{"USER", "HOME"}; # под Unix
($who, $home) = @ENV{"USERNAME", "HOMEPATH"}; # и Windows XP
```

Контекст и хэш

- Хэш — не добавляет нового типа контекста: ключи и значения отдельных элементов хэша — это скалярные величины, а перечень всех элементов хэша, срезы хэша, выборки всех его ключей и всех его значений — это списки.
- Хотя переменная-хэш хранит особое значение — ассоциативный массив, но когда она применяется в левой части операции присваивания, она создает списочный контекст.

На этом основаны ***приемы инициализации хэшей значениями списков***. Поэтому же, например,

- при присваивании хэшу скалярной величины она рассматривается как список, состоящий из одного элемента, и этот элемент становится единственным ключом в хэше, с которым ассоциируется неопределенное (не присвоенное) значение:

```
%hash = $scalar; # то же, что %hash = ($scalar)
# defined($hash{$scalar})  будет ложно: значения не было
# exists($hash{$scalar}) будет истинно: ключ есть
```