ССЫЛКИ

Ссылки являются одним из скалярных типов данных языка Perl, наряду с числами и строками.

- ***Ссылка*** (reference) — это информация о том, где при выполнении программы располагается в памяти объект определенного типа.

Эту информацию можно использовать для доступа к объекту ссылки (referent). **Ссылка — это возможность обратиться к какой-то информации не по имени, которое известно при компиляции, а по ее расположению в памяти при выполнении программы**.

В Perl не нужно заботиться о явном удалении из памяти объектов ссылок, поскольку

- занимаемая память автоматически освобождается встроенным сборщиком мусора, когда объекты ссылок перестают использоваться (то есть когда на них больше не указывает ни одна ссылка).

Для создания ссылки на существующий программный объект предусмотрена операция взятия ссылки, обозначаемая обратной косой чертой (backslash), которая ставится перед объектом ссылки.

Пример. Как можно получить ссылку на скалярную переменную и сохранить ее в другой переменной:

```
$scalar = 'Скаляр'; # объект ссылки
$ref2scalar = \$scalar; # ссылка на скаляр
```

При помощи операции **'\' можно брать ссылку на любые объекты, например, на литерал или на любое другое выражение**, только при этом значение объекта ссылки нельзя будет изменить. В этом случае *при выполнении программы выражение вычисляется, а результат сохраняется в анонимной области памяти, ссылка на которую и возвращается*.

Пример:

```
$ref2literal = \'Литерал'; # ссылка на литерал

$ref2expression = \($n1*$n2); # ссылка на выражение
```

- ***Анонимные данные*** — это значения, не связанные ни с одной переменной, доступ к которым происходит только по ссылке.

Ссылка всегда указывает на значение конкретного типа: скаляр, массив, хэш, подпрограмму (или элемент таблицы символов, о которых речи пока не было).

- На **какой именно тип объекта** указывает ссылка, можно узнать с помощью функции ***ref***(), которая возвращает строку, описывающую тип значения объекта ссылки.

Пример:

```
print ref($ref2scalar); # выведет: 'SCALAR'
```

Что получится, **если вывести значение самой ссылки**? Ее значение будет преобразовано в строку, содержащую тип объекта ссылки и его адрес в виде шестнадцатиричного числа, например:

print $ref2scalar; # выведет, например: 'SCALAR(0x335b04)'

- Обратно **преобразовать в ссылку это строковое представление адреса не удастся.**

РАЗЫМЕНОВАНИЕ ССЫЛОК

Чтобы получить **доступ к значению, на которое указывает ссылка**, нужно выполнить разыменование ссылки (dereference). Для этого

- переменная, содержащая ссылку, заключается в фигурные скобки и перед ней ставится нужный разыменовывающий префикс: $ для скаляра, @ для массива, % для хэша, & для подпрограммы.

(Другими словами, после разыменовывающего префикса, определяющего тип хранящегося в переменной значения, вместо имени переменной записывается содержащая ссылку переменная или выражение, возвращающее ссылку.)

Если не возникает неоднозначности в интерпретации выражения, то переменную, хранящую ссылку, в фигурные скобки можно не заключать.

Примеры **разыменования ссылок на скалярные значения**:

```
print "${$ref2scalar} "; # или: $$ref2scalar

print "${$ref2literal} "; # или: $$ref2literal

print "${$ref2expression} "; # или: $$ref2expression
```

Значение скалярной переменной при доступе по ссылке, естественно, может изменяться, но попытка с помощью ссылки изменить литерал вызовет ошибку при выполнении программы ("Modification of a read-only value attempted"):

```
${$ref2scalar} = 'Новый скаляр'; # вполне законно

${$ref2literal} = 'Новый литерал'; # ОШИБКА!!!
```

Когда на какое-то значение ссылается несколько ссылочных переменных, то обращаться к этому значению можно с помощью любой из них. Значение доступно до тех пор, пока на него имеется хотя бы одна ссылка. Например:

```
$ref2scalar = \$scalar; # ссылка на скаляр

$one_more_ref = $ref2scalar; # копия ссылки на скаляр

# будет выведено одно и то же значение $scalar:

print "${$ref2scalar} ${$one_more_ref}";
```

Если переменная $ref2scalar перестанет ссылаться на $scalar (например, после undef $ref2scalar), то значение $scalar все еще будет доступно через переменную $one_more_ref.

Значением объекта ссылки также может быть ссылка (косвенная ссылка) на другой объект, возможно, тоже на ссылку. Ссылка даже может содержать ссылку на саму себя! Таким образом при необходимости можно построить цепочку ссылок любой длины. Например:

$value = 'Полезное значение';

$ref1 = \$value; # ссылка на значение

$ref2 = \$ref1; # ссылка на ссылку на значение

$ref3 = \$ref2; # ссылка на ссылку на ссылку на значение

Можно организовать многоуровневые косвенные ссылки без использования промежуточных переменных, несколько раз применяя операцию взятия ссылки на значение. Например, создадим такую цепочку ссылок:

$ref_chain = \\\$value; # цепочка из трех ссылок

Для доступа по такой цепочке ссылок к исходному значению правило разыменования применяется нужное число раз. Например, в цепочке из трех ссылок с помощью трех префиксов $ мы последовательно получаем доступ к ссылочным переменным, а еще один префикс нужен для доступа к полезному значению:

выведем исходное значение через $ref3:

print ${${${$ref3}}}; # или короче: print $$$$ref3;

или через $ref_chain:

print $$$$ref_chain;

Если применить функцию ref() к переменной, содержащей ссылку на другой объект, то она вернет строку 'REF'. Если преобразовать в строку значение ссылки на ссылку, то будут выведено обозначение ссылочного типа и адрес объекта ссылки, например:

print ref($ref_chain); # выведет: 'REF'

print $ref_chain; # выведет, например: 'REF(0x334e8c)'

ССЫЛКИ НА МАССИВЫ

Ссылка на переменную типа "массив" также создается с помощью операции взятия ссылки:

```
@array = ('Это', 'список', 'в', 'массиве');

$ref2array = \@array; # ссылка на массив
```

Если обращение к массиву будет происходить только по ссылке, то можно обойтись без переменной типа "массив", а **создать анонимный массив и присвоить ссылку на него в скалярную переменную**.

- Ссылка на анонимный массив создается с помощью квадратных скобок, в которые заключается список начальных значений безымянного массива:

```
$ref2anon = [ # ссылка на анонимный массив

'Это', 'анонимный', 'массив'

]; # конец присваивания ссылки

$ref2empty = []; # ссылка на пустой анонимный массив
```

- **Анонимные массивы удобно использовать для создания ссылки на копию массива**.

Для этого существующий массив помещается в квадратные скобки, и его значение будет скопировано в созданный анонимный массив:

```
$ref2copy = [@array]; # ссылка на копию массива
```

- **Разыменование ссылки на массив** производится аналогично разыменованию ссылки на скалярную переменную, только с использованием префикса массива @:

```
# будет выведено одно и то же значение @array:

print "@{$ref2array} @$ref2array\n";
```

Естественно, что, **обращаясь к массиву по ссылке, можно выполнять с ним любые действия**, как и с именованным массивом, например:

```
@array_copy = @{$ref2array}; # копия массива

@{$ref2array}[0,1] = ('Новый', 'список'); # срез массива
```

Разыменование ссылки на элемент массива оформляется так:

перед ссылочной переменной, которая может заключаться в фигурные скобки, указывается префикс скалярного значения $, а после ссылочной переменной указывается индекс элемента в квадратных скобках. Другими словами, для обращения к элементу массива по ссылке имя массива заменяется ссылочной переменной:

```
print ${$ref2array}[0]; # или: $$ref2array[0]
```

- **Обращение по ссылке к элементу массива более наглядно записывается с помощью инфиксной операции ->**, слева от которой записывается имя переменной, содержащей ссылку на массив, а справа — индекс элемента массива в квадратных скобках.

Операция "стрелка" наглядно представляет ссылку, символы -> в ней записываются без пробела между ними. Вот пример:

```
# доступ по ссылке к значению элемента массива:

$element_value = $ref2array->[0];

# изменение значения элемента массива:

$ref2array->[0] = $new_value;
```

- Как к обычным скалярным значениям можно обращаться по ссылке к отдельным элементам массива, например:

```
$ref2element = \$array[0]; # ссылка на элемент массива

${$ref2element} = $new_value; # изменение элемента массива
```

МНОГОМЕРНЫЕ МАССИВЫ

В элементах массива можно хранить ссылки на другие массивы: это позволяет создавать в Perl многомерные массивы или "массивы массивов", как это делается в языке Java. В этом случае доступ к элементам многомерного массива также обычно записывается с использованием операции "стрелка", которая употребляется нужное количество раз:

```
@{$AoANxM->[$n]} # вложенный массив

$AoANxM->[$n]->[$m] # скалярный элемент двумерного массива

$AoANxMxP->[$n]->[$m]->[$p] # элемент 3-мерного массива
```

- Для удобства чтения программы **допускается не записывать операцию "стрелка" между парами индексов массива в квадратных скобках:**

```
$AoANxM->[$n][$m] # так гораздо симпатичнее!
```

```
$AoANxMxP->[$n][$m][$p] # а тем более так...
```

Пример. **Программа создания двумерного массива из трех строк по пять элементов в каждой строке**:

```
my $AoARxC = []; # ссылка на анонимный массив массивов

for (my $row = 0; $row < 3; $row++) { # цикл по строкам

      $AoARxC->[$row] = []; # строка: вложенный массив

            for (my $col = 0; $col < 5; $col++) { # по колонкам

                  $AoARxC->[$row]->[$col] = ($row+1).'.'.($col+1);

            }

      }
```

- **Небольшие многомерные массивы удобно создавать, используя вложенные анонимные массивы**.

  Это присваивание создаст такой же массив, что и в предыдущем примере:

```
$AoARxC = [ # ссылка на двумерный анонимный массив

      [1.1, 1.2, 1.3, 1.4, 1.5], # 1-я "строка"

      [2.1, 2.2, 2.3, 2.4, 2.5], # 2-я "строка"

      [3.1, 3.2, 3.3, 3.4, 3.5] # 3-я "строка"

]; # конец присваивания ссылки
```

- Для **вывода значений многомерного массива** обычно используется нужное число вложенных циклов `for` или других циклических конструкций:

```
# цикл по строкам (элементам массива верхнего уровня)

for (my $row = 0; $row < @{$AoARxC}; $row++) {

      # цикл по столбцам (элементам вложенных массивов)

      for (my $col = 0; $col < @{$AoARxC->[$row]}; $col++) {

            print "$AoARxC->[$row]->[$col] ";

      }

      print "\n";

}
```

В результате выполнения этой программы построчно будет выведено значение всех элементов из массива массивов:

```
1.1 1.2 1.3 1.4 1.5

2.1 2.2 2.3 2.4 2.5
```

```
3.1 3.2 3.3 3.4 3.5
```

В любой массив можно поместить список ссылок на другие программные объекты, например, таким образом:

```
@reference_list = (\$scalar, \@array, \%hash);
```

Можно записать то же самое более простым способом, поставив операцию взятия ссылки перед списком объектов в круглых скобках:

```
@reference_list = \($scalar, @array, %hash);
```

Списки ссылок на объекты могут, например, передаваться в подпрограмму для изменения перечисленных в списке объектов.

АВТОСОЗДАНИЕ ОБЪЕКТА ССЫЛКИ

Если попытаться разыменовать ссылку на несуществующий объект, то он автоматически будет создан. В этом случае работает механизм, называемый ***автосозданием объекта ссылки*** (буквально: "автооживление" — `autovivification`).

Пример. Во время обращения по ссылке к элементу массива автоматически создается массив из пяти элементов, ссылка на него присваивается в переменную $array_ref, а пятый элемент получает начальное значение:

```
$array_ref->[4] = '5-й элемент'; # присваивание значения

print ref($array_ref); # вызывает к жизни массив

print "\n";

print scalar(@{$array_ref}); #  из 5 элементов! печатаем длину

print "\n";

print $$array_ref[4]; # печатаем значение
```

Вывод:

```
ARRAY

5

5-й элемент
```

- Подобным образом применяя автосоздание объектов, можно создать цепочку ссылок, указывающих на некоторое значение:

```
$$$$ref = 25; # при попытке присвоить значение

# создаются 2 ссылочных переменных и 1 скалярная

print "$ref $$ref $$$ref $$$$ref\n";

# выведет: REF(0x334dd8) REF(0x334e8c) SCALAR(0x334e98) 25
```

ССЫЛКИ НА ХЭШИ

Все, что говорилось о ссылках на массивы, применимо к ссылкам на хэши. Ссылка на переменную типа "хэш" получается с помощью операции взятия ссылки:

```
%hash = ('Хэш' => 'ассоциативный массив');

$ref2hash = \%hash; # ссылка на весь хэш

print ref($ref2hash); # вернет: HASH
```

- Ссылка **на анонимный хэш** создается с помощью фигурных скобок, в которых записываются начальные значения хэша:

```
$ref2anon = { # ссылка на анонимный хэш

    'language' => 'Perl',

    'author' => 'Larry Wall',

    'version' => 5.8

}; # конец присваивания ссылки
```

- **При помощи анонимного хэша удобно создавать копию существующего хэша**, чтобы затем работать с ним через ссылку:

```
$ref2copy = {%hash}; # ссылка на копию хэша
```

- **Разыменование ссылки на хэш** записывается так же, как разыменование ссылки на массив, но с префиксом хэша %. При разыменовании ссылки на хэш переменную, содержащую ссылку, для наглядности можно обрамлять фигурными скобками:

```
# будет выведено одно и то же значение %hash:

print %{$ref2hash}, %$ref2hash;
```

- При помощи ссылок с хэшами можно выполнять любые действия, обращая внимание на правильное разыменование ссылки:

```
%hash_copy = %{$ref2hash}; # копия хэша

@hash_slice= @{$ref2hash}{$key1, $key2}; # срез хэша (массив)

@hash_keys = keys %{$ref2hash}; # ключи хэша (массив)
```

- **Разыменование ссылки на элемент хэша** также записывается уже знакомым нам способом, когда перед ссылочной переменной ставится префикс скаляра $, а после нее — ключ элемента хэша в фигурных скобках.

Ссылочная переменная может заключаться в фигурные скобки:

```
${$ref2hash}{'ключ'} = 'значение'; # изменение значения

print $$ref2hash{'ключ'}; # доступ к значению элемента хэша
```

В другой форме разыменования ссылки к переменной, содержащей ссылку на хэш, применяется операция "стрелка", после которой в фигурных скобках указывается ключ элемента хэша. Вот так:

```
$ref2hash->{'термин'} = 'определение'; # добавим элемент

$value = $ref2hash->{'Хэш'}; # найдем значение по ключу
```

Если ссылка используется как ключ хэша, она, как и любой ключ хэша, автоматически преобразуется в строку. Такие строки невозможно применять для доступа к объектам ссылки, но они могут служить отличными уникальными ключами хэша, поскольку строковое значение ссылки содержит адрес объекта в памяти, например, 'SCALAR(0x335b04)' или 'ARRAY(0x334dd8)'. Если все-таки требуется использовать ссылки в качестве ключей хэша, то можно воспользоваться модулем Tie::RefHash.

ССЫЛОЧНЫЕ СТРУКТУРЫ ДАННЫХ

Аналогично созданию "массива массивов" создаются и другие разновидности ссылочных структур данных: массивы хэшей, хэши массивов и хэши хэшей. Ссылочные структуры применяются для структурированного представления взаимосвязанных данных.

- Для хранения в каждом элементе массива нескольких значений применяется массив хэшей (Array of Hashes, AoH).

Пример **массива, содержащий ссылки на анонимные хэши, в каждом из которых хранятся сведения о каком-либо объекте**:

```
$AoH = [ # студенты 1 курса

      {name=>'Петя', age=>'2003.03.12'},

      {name =>'Вася', age =>'2003.05.19'},

      {name =>'Митя', age =>'2003.06.02'},

      {name =>'Ваня', age =>'2003.04.23'},

      # и так далее...

];

# напечатать хэш, на который ссылается 4-й элемент массива

print "Студент $AoH->[3]->{name} $AoH->[3]->{age}";

# выведет: Студент Ваня 2003.04.23
```

- Для того чтобы ассоциировать с каждым ключом хэша список скалярных значений, применяется **хэш массивов (Hash of Arrays, HoA).**

Пример**. Хэш массивов, где в каждом элементе хэша хранится ссылка на анонимный список ассоциированных значений**:

```
$HoA = { # годы создания языков программирования

      1964 => ['SIMULA', 'BASIC', 'PL/1'],

      1970 => ['Forth', 'Pascal', 'Prolog'],

      1979 => ['Ada', 'Modula-2'],

      1987 => ['Perl', 'Haskell', 'Oberon'],

      1991 => ['Python', 'Visual Basic'] };
```

```
# напечатать список, ассоциированный с 1987 годом

foreach my $language (sort @{$HoA->{1987}}) {

     print "$language ";

} # выведет: Haskell Oberon Perl
```

- Элементы хэша также могут хранить ссылки на другие хэши, образуя хэш хэшей (Hash of Hashes, HoH).

Пример описания хэша хэшей, где с каждым поисковым ключом ассоциируется анонимный хэш с информацией об объекте:

```
my $HoH = { # авторы и годы создания языков программирования

     'Pascal' => {author=>'Niklaus Wirth', year=>1970},

     'Perl' => {year=>1987, author=>'Larry Wall'},

     'C' => {author=>'Dennis Ritchie', year=>1972}

     };

 # в каком году был создан Pascal?

print $HoH->{'Pascal'}->{'year'}; # выведет: 1970

# кто создал язык Си?

print $HoH->{'C'}->{'author'}; # выведет: Dennis Ritchie
```

Имеющиеся в других языках программирования записи (record) или структуры (struct), в Perl чаще всего представляются в виде хэшей, в которых ключи используются в качестве имен полей и применяются для доступа к значениям полей записи.

Пример. Создадим набор записей с информацией о людях.

- Каждая запись будет анонимным хэшем, а ссылки на записи будут храниться в массиве.
- В каждой записи дату рождения представим в виде анонимного массива, содержащего год, месяц и день. Вот таким образом:

```
my $family = [ # массив записей о семье

     {name => 'Михаил', birthday => [1958, 11, 12]},

     {name => 'Ирина', birthday => [1955, 03, 23]},

     {name => 'Маша', birthday => [1980, 07, 27]},

     {name => 'Миша', birthday => [1981, 11, 28]},

     {name => 'Лев', birthday => [1988, 06, 24]}

];

# напечатаем год рождения Маши:

print "$family->[2]->{birthday}->[0]";
```

```
# или проще:

print "$family->[2]{birthday}[0]"; # выведет: 1980
```

- Подобные структуры легко динамически модифицировать при выполнении программы.

Пример, добавим в каждую запись новое поле — `'address'`, в котором сохраним ссылку на запись о месте проживания человека. Адрес оформим в виде анонимного хэша из нескольких полей:

```
# адрес в виде анонимного хэша, в $address - ссылка на него:

$address = {country => 'Россия', index => 641870}; # и т.д.

# добавить поле адреса и поместить туда $address:

foreach my $person (@{$family}) { # пропишем всех


    address = {country => 'Россия', index => 641870}

    $person->{address} = $address; # по одному адресу

} # выведем почтовый индекс для Ирины

print "$family->[1]->{address}->{index}\n"; # 641870
```

С помощью ссылок создаются и другие динамические структуры данных: связные списки, деревья и графы.

Таблица. Синтаксические конструкции для работы со ссылками на данные

	Скаляр	Массив	Хэш
Взятие ссылки на объект	$sref = \$scalar;	$aref = \@array;	$href = \%hash;
Создание ссылки на анонимный объект	$sref = \'Литерал';	$aref = [$a, $b];	$href = {$a => $b};
Доступ к значению объекта ссылки	${$sref} $$sref	@{$aref} @$aref	%{$href} %$href
Доступ к значению элемента объекта ссылки		$aref-> [$index] ${$aref}[$index]	$href->{$key} ${$href}{$key}
Доступ к срезу объекта ссылки		@{$aref}[$i1, $i2]	@{$href}{$k1, $k2}
Значение функции ref($ref) для объекта ссылки	SCALAR	ARRAY	HASH

Далее в примерах обращается внимание на различие в возможных формах создания сложных структур данных.

Например, можно так (обычный массив, элементы -- анонимные массивы, т.е. ссылки):

```
@m=(["q","w","e","r"],[1,2,3,4],["a","s","d","f"]);
print $m[0][0];
```

И можно так (анонимный массив, элементы -- анонимные массивы):

```
$m=[["q","w","e","r"],[1,2,3,4],["a","s","d","f"]];
print $m->[0][0];
```

Пример. Массив массивов

Читаем строки из файла и строим ***массив ссылок*** на массивы слов по строкам

```
while(<FILE>)
{
        @temp=split;
        Push @ss, [@temp];
}
$p=$ss[0]; # массив слов первой строки файла -- ссылка
Print “@$p”;
```

Пример. То же самое, но если формируем ***ссылку на массивы слов*** по строкам

```
while(<FILE>)
{
        @temp=split;
        Push @$ss, [@temp];
}
$p=$$ss[0]; # массив слов первой строки файла -- ссылка
Print “@$p”;
```

Пример. Массив хэшей

Здесь предполагается, что в файле text.txt данные записаны в виде

```
first=Programming second=PHP third=PHP

first=MMKG second=SRV third=Perl fourth=OSO

first=SPPO second=SPPO
```

Формируем массив хешей:

```
@mh=( {"1"=>"Programming","2"=>"PHP","3"=>"PHP",
"4"=>"KompGraph"},
{"1"=>"MMKG","2"=>"SRV","3"=>"Perl"}, {"1"=>"SPPO"
,"2"=>"Perl"} );

print $mh[1]{"2"};
```

Добавляем новый хеш:

```
push @mh, {"1"=>"SPPO","2"=>"SPPO"};

print "\n$mh[3]{'1'}";
```

Рассмотрим заполнение массива хешей из файла.

```
open(FILE,"<text.txt");

while(<FILE>)
{
   $rec={};

   for $field (split)
   {
     ($key,$value)=split "=", $field;

     $rec->{$key}=$value;

    }

push @mh, $rec;

}

print $mh[1]{second};
```

Пример. Хэш хэшей

Сгенерируем теперь хеш из файла. Пусть в файле text.txt имеются записи вида

```
P0:x=10.2 y=2.3 z=4.4
```

Тогда формируем хеш, например, так:

```
open(FILE,"<text.txt");

while($line=<FILE>)

{

@point=split(":",$line); @coor=split(" ",$point[1]);
@x=split("=",$coor[0]); @y=split("=",$coor[1]);
@z=split("=",$coor[2]); $hh{$point[0]}{$x[0]}=$x[1];
$hh{$point[0]}{$y[0]}=$y[1]; $hh{$point[0]}{$z[0]}=$z[1]; }

$p=$hh{"P1"};
for $key (keys %$p)
{

    print "\n".$key." = ".$$p{$key};

}
```

Или тело внутреннего цикла можно сделать короче:

```
@point=split(":",$line);

@coor=split(" ",$point[1]);

for $str (@coor)
{
   @v=split("=",$str);
   $hh{$point[0]}{$v[0]}=$v[1];

}
```

Теперь добавим запись в хеш:

```
$hh{"P4"}{'x'}=3.2;
```

Наконец, выведем весь хеш хешей:

```
for $k (keys %hh)
{
   $p=$hh{$k}; print "\n\n$k :";
   for $key (keys %$p)
   {
      print "\n".$key." = ".$$p{$key};

   }
}
```

Дополнительно к теме

Интерполяция выражений с помощью ссылок

Иногда требуется включить в строку результат вычисления какого-либо выражения. С помощью ссылок можно интерполировать любое выражение, например, вызов функции.

- Чтобы включить ***в строку значение скалярного выражения***, его надо заключить в круглые скобки и взять на него ссылку операцией \, а затем разыменовать ссылочное выражение как скаляр с помощью префикса $.

Вот таким образом:

```
$s = "localtime() ${\($x=localtime)} в скалярном ко\нтексте";
```

```
print "$s";

# Вывод -- значение выражения, например:

localtime() Mon Oct 11 07:19:55 2021 в скалярном контексте
```

- Чтобы включить ***в строку значение выражения, возвращающего список***, его надо заключить в квадратные скобки, организуя ссылку на анонимный массив, а потом разыменовать ссылочное выражение как массив с помощью префикса @.

```
$a = "localtime() @{[localtime()]} в списочном контексте";

# значение выражения, например:

localtime() 26 24 7 11 9 121 1 283 0 в списочном контексте
```

Символические ссылки

Ссылки, о которых до этого шла речь, называются жесткими ссылками.

Жесткая ссылка (hard reference) — это программный объект, хранящий в памяти адрес референта и тип его значения.

В Perl имеется еще одна разновидность ссылок, называемых символическими ссылками.

Символическая ссылка (symbolic reference) — это строковое значение, которое хранится в скалярной переменной и представляет из себя имя глобальной переменной:

```
$referent1 = 'Референт'; # объект ссылки

$symlink = 'referent1'; # символическая ссылка

# доступ по символической ссылке к значению объекта

print ${$symlink}; # будет выведено: 'Референт'
```

Символические ссылки используются значительно реже и бывают нужны, когда

- требуется во время выполнения программы программно создавать имена переменных в виде строк, чтобы затем получать доступ к их значениям.

Использование символических ссылок может приводить к трудно обнаруживаемым ошибкам, поэтому лучше запрещать использование в программе символических ссылок с помощью прагмы use strict 'refs'.