МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

НАБЕРЕЖНОЧЕЛНИНСКИЙ ИНСТИТУТ

СИНТАКСИС ЯЗЫКА PERL

Учебно-методическое пособие
по дисциплине
«ВЕБ-ПРОГРАММИРОВАНИЕ»

Набережные Челны
2018

Галиуллин Л.А. Синтаксис языка Perl: учебно-методическое пособие по дисциплине «Веб-программирование» [Электронный ресурс] / Казанский федеральный университет, Электронный архив, 2018.

Рассматривается язык разработки web-приложений и язык системных сценариев Perl. Представлены синтаксис Perl, декларации, сложные предложения, основные операторы. Приведены контрольные вопросы. Для студентов направлений подготовки «Информатика и вычислительная техника», «Программная инженерия».

Введение

Прежде чем приступить к последовательному ознакомлению с новым для вас языком, должен оговориться и сказать, что все примеры да и сам язык, описание которого следует ниже - это Perl версии для операционной системы Linux Red Hat версии и ActivePerl для Windows 7/NT/2000. Существуют реализации этого языка для операционных систем OS/2 , MS-DOS, но они отстают по возможностям от оригинала, рожденного в Unix.

На пятнадцатый год своего существования *Практический Язык для Извлечения текстов и Генерации отчетов* (Practical Extraction and Reporting Language) по-прежнему популярен не только среди линуксоидов, но и среди Web-программистов.

Начнем с самого простого. Введите в файл test1.pl следующие строки:

```
#!/usr/local/bin/perl
# Содержимое файла test1.pl
print "Наше Вам с кисточкой!\n";
```

А теперь подробно разберем каждую строку.

```
#!/usr/local/bin/perl
```

Данная строка должна быть первой в любой Perl-программе. Она указывает системному интерпретатору, что данный файл - это Perl-программа.

```
# Содержимое файла test1.pl
```

Эта строка комментария. Она всегда начинается символом '#'.

```
print "Наше Вам с кисточкой!\n";
```

Самая последняя строка просто выводит на экран надпись "Наше Вам с кисточкой!".

Здесь слово print - это команда "вывести". Все что в кавычках - это символы, \n - перевод строки и ';' - признак конца команды. Он обязателен. В одной строке может быть несколько команд и все они должны завершаться символом ';'. После него может быть символ '#' - тогда остаток строки считается комментарием.

Чтобы этот пример заработал, вам надо иметь установленный Perl и набрать в командной строке: perl test1.pl (в Windows) или ./test.pl (в *nix).

Синтаксис Perl

Perl программа (скрипт) состоит из последовательности деклараций и предложений. Что должно быть обязательно декларировано, так это форматы отчетов и подпрограммы (функции). Все необъявленные переменные, массивы имеют значение 0 или null.

Декларации (объявления)

Perl имеет свободный формат. Комментарии начинаются с символа '#' и продолжаются до конца строки. Декларации могут использоваться в любом месте программы так же как и предложения (statements), но действуют они только в фазе компиляции программы. Обычно их помещают или в начале или в конце программы. Декларация подпрограмм позволяет использовать имя подпрограммы как списковый оператор, начиная с момента декларирования:

```
sub test;       # Декларация подпрограммы test
$var1 = test $0; # Использование как оператора списка.
```

Декларации подпрограмм могут быть загружены из отдельного файла предложением require или загружены и импортированы в текущую область имен предложением use.

Простое предложение

Простое предложение обязательно заканчивается символом ';', если только это не последнее предложение в блоке, где ';' можно опустить. Существуют операторы, такие как eval{} и do{}, которые выглядят как сложные предложения, но на самом деле это термы и требуют обязательного указания конца предложения.

Любое простое предложение может содержать модификатор перед ';'. Существуют следующие модификаторы:

if EXPR, unless EXPR, while EXPR, until EXPR

где EXPR - выражение, возвращающее логическое значение true или false. Модификаторы while и until вычисляются в начале

предложения, кроме do, который выполняется первым.

if EXPR- Модификатор "если". Предложение выполняется, если EXPR возвращает true.

```
$var = 1;
if $var > 0 $var2 = 3; # Результат: $var2 = 3
```

while EXPR - Модификатор "пока". Предложение выполняется пока EXPR = true

```
$var = 1;
print $var++ while $var < 5; # Результат: 1234
```

until EXPR- Модификатор "до ". Предложение выполняется до тех пор пока EXPR = false

```
$var = 1;
print $var++ until $var > 5; # Результат: 12345
```

unless EXPR - Модификатор "если не". Обратный к if. Выражение выполняется, если EXPR = false.

```
$var = 1;
print $var++ unless $var > 5; # Результат: 1
```

Сложные предложения

Последовательность простых предложений, ограниченная функциональными ограничителями, называется блоком. В Perl это может быть целый файл, последовательность предложений в операторе eval{} или чаще всего это множество простых предложений, ограниченных круглыми скобками '{}'.

Существуют следующие виды сложных предложений:

if (EXPR) BLOCK, if (EXPR) BLOCK else BLOCK, if (EXPR) BLOCK elsif (EXPR) BLOCK ... else BLOCK

LABEL while (EXPR) BLOCK, LABEL while (EXPR) BLOCK continue BLOCK

LABEL for (EXPR; EXPR; EXPR) BLOCK, LABEL foreach VAR (LIST) BLOCK

LABEL BLOCK continue BLOCK

Обратите внимание, что сложные предложения описаны в термах блоков, а не предложений, как в языке C. Поэтому необходимо использовать круглые скобки для обозначения блока.

if (EXPR) BLOCK - Вычисляется логическое выражение EXPR

и если true блок выполняется.

```
$var =1;
if ($var == 1)
{ print $var,"\n";
}
```

Результат: 1

if (EXPR) BLOCK else BLOCK2 - Если EXPR=true выполняется BLOCK иначе BLOCK2.

```
$var =2;
if ($var == 1)
{ print "\$var = 1\n";
}
else
{ print "\$var не равно 1\n";
}
```

Результат: $var не равно 1

if (EXPR1) BLOCK1 elsif (EXPR2) BLOCK2 ... else BLOCK - Если EXPR1=true выполняется BLOCK1 иначе если EXPR2=true выполняется BLOCK2 иначе ... иначе BLOCK.

```
$var = 1;
if ($var == 0)
{ print "\$var = 0\n";
}
elsif ($var == 1)
{ print "\$var = 1\n";
}
else
{ print "Не известное \$var\n";
}
```

Результат: $var = 1

Цикл While. Выполняет BLOCK до тех пор пока EXPR = true. Метка LABEL не обязательна и состоит из идентификатора, завершающегося символом ':'. Метка необходима при использовании внутри блока цикла управляющих операторов next, last и redo. Если метка отсутствует, то эти операторы ссылаются к началу ближайшего цикла. Блок после continue выполняется всегда перед тем, как вычисляется логическое выражение EXPR. Это подобно EXPR3 в предложении for,

поэтому в этом блоке удобно изменять счетчики и флаги цикла, даже если применяется оператор next.

Операторы управления циклом

next - подобен continue в C. Переходит к началу текущего цикла, т.е. повторяет итерацию.

```
M1:
while ($i < 6)
{
++$i;          # Увеличиваем счетчик на 1
next M1 if $i < 3; # Переходим в начало если $i < 3
++$i;          # иначе увеличиваем счетчик еще раз на 1
}
continue
{
print "$i "; # Результат: 1 2 4 6
}
```

last - подобен оператору break в языке C. Немедленно прерывает цикл. Блок continue пропускается.

```
M1:
while ($i < 6)
{
++$i;      # Увеличиваем счетчик на 1
last M1 if $i > 3; # Выход из цикла если $i > 3
++$i;      # иначе увеличиваем счетчик еще раз на 1
}
continue {
print "$i "; # Результат: 2 4
}
```

redo - начать новый цикл, не вычисляя EXPR и не выполняя continue блок.

```
M1:
while ($i < 6)
{
++$i;      # Увеличиваем счетчик на 1
redo M1 if $i == 3; # Далее пропустить для $i = 3
++$i;      # иначе увеличиваем счетчик еще раз на 1
```

```
}
continue {
print "$i "; # Результат: 2 5 7
}
```

Цикл for. LABEL for (EXPR1; EXPR2; EXPR3) BLOCK. Оператор for полностью аналогичен оператору for в C. Перед началом цикла выполняется EXPR1, если EXPR2 = true выполняется блок, затем выполняется EXPR3.

```
for ($i = 2; $i < 5; ++$i){
print $i, " "; # Результат: 2 3 4
}
print "\nПосле цикла i = $i\n"; # После цикла i = 5
```

Цикл foreach. LABEL foreach VAR (LIST) BLOCK. Переменной VAR присваивается поочередно каждый элемент списка LIST и выполняется блок. Если VAR опущено, то элементы присваиваются встроенной переменной $_. Если в теле блока изменять значение VAR то это вызовет изменение и элементов списка т.к. VAR фактически указывает на текущий элемент списка. Вместо слова foreach можно писать просто for - это слова синонимы.

```
@month = ("январь","февраль","март");  # Создали массив
foreach $i (@month)
{ print $i," ";       # Результат: январь февраль март
}
foreach $i (@month)
{ $i = uc($i); # Перевели в верхний регистр
}
print @ month; # Результат: ЯНВАРЬФЕВРАЛЬМАРТ
for $i (3,5,7)
{ print "$i "; # Результат: 3 5 7
}
```

Блоки и оператор switch

Блок не зависимо от того имеет он метку или нет семантически представляет собой цикл который выполняется один раз. Поэтому действие операторов цикла next, last, redo - аналогично описанному выше. Блоки удобны для построения

switch (переключатель) структур. В Perl нет специального оператора switch подобного языку C поэтому вы сами можете создавать удобные для вас конструкции. Опыт автора показывает что для простоты написания лучше всего подходит конструкция вида if ... elsif ... else ... хотя можно сочинить и нечто подобное:

```
SWITCH:
{
if ($i ==1 ) { .....; last SWITCH; }
if ($i ==2 ) { .....; last SWITCH; }
if ($i ==3 ) { .....; last SWITCH; }
$default = 13;
}
```

Выбирайте сами по своему вкусу.

Оператор goto

В Perl существует оператор goto. При создании больших производственных задач на последнем этапе, особенно при отработке ошибочных ситуаций конечно goto нужен.

В Perl реализовано три формы goto. goto - метка, goto - выражение и goto - подпрограмма.

goto метка - выполняет непосредственный переход на указанную метку.

goto - выражение - Вычисляет имя метки и делает соответствующий переход. Например, если мы хотим сделать переход на одну из трех меток "M1:", "M2:" или "M3:" в зависимости от значений переменной $i равной 0, 1 или 2 то это лучше сделать следующим образом:

```
goto ("M1", "M2", "M3")[$i];
```

здесь $i используется как индекс массива указанного непосредственно в выражении.

goto подпрограмма - довольно редкий случай т.к. всегда проще и надежней вызвать подпрограмму "естественным" образом.

POD операторы. Документирование программ

В Perl реализован очень удобный механизм для написания документации в момент создания программы. Для этого применяются специальные POD операторы. Если в теле программы интерпретатор встречает оператор начинающийся с символа '=' например:

= head Набор стандартных процедур

то пропускается все до слова '=cut'. Это удобно для включения длинных на несколько строк или страниц комментариев. Затем с помощью специальной программы pod можно отделить текст документации от текста программы.

Переменные

В Perl существует три типа структур данных: скаляры, массивы скаляров и хеши (hashes) - ассоциативные массивы скаляров. Обычно элементы массивов индексируются целыми числами, первый элемент - нулевой. Отрицательное значение индекса обозначает номер позиции элемента с конца. Хеши индексируются строками символов.

Имена скаляров всегда начинаются с символа '$' - даже когда обозначают элемент массива:

```
$var1                   # Простой скаляр 'var1'
$var1[0]    # Первый элемент массива 'var1'
$var1{'first'}          # Элемент с индексом 'first'
```

В случае использования имени массива "целиком" или его "среза" перед именем массива ставится символ '@':

```
@var1       #Все элементы массива var1 ($var1[0],$var1[1],..., $var1[n])
@var1[1,3,10]       # Элементы $var1[1], $var1[3], $var1[10]
@var1{'first','last'} #то же что и ($var1{'first'}, $var1{'last'})
```

Хеш "целиком" начинается с символа '%':

%var, %key, %years

Имена подпрограмм начинаются символом '&', если из контекста не видно, что это подпрограмма:

&sub1, &test_prog, test(12)

Имена таблиц символов всегда начинаются символом '*'.

Каждый тип переменных имеет свою область памяти поэтому $var1 и $var1[0] совершенно разные переменные, хотя $var1[0] часть массива @var1. Так же @var1 и %var1 - разные массивы переменных.

Имена переменных могут содержать любые буквенно-цифровые символы за исключением пробела и табуляции. Эти символы используются в качестве разделителей. Большие и малые буквы различаются поэтому $var1 и $Var1 - разные переменные. В Perl по умолчанию имена меток и указателей файлов пишут большими буквами.

Скалярные значения

Все данные в Perl это скаляры, массивы скаляров и хеши скаляров. Скалярные переменные могут содержать числа, строки и ссылки. Преобразование числа - строки происходит автоматически по умолчанию. Скаляр может иметь только одно единственное значение, хотя это может быть ссылка на массив скаляров. Так как Perl сам преобразовывает числа в строки и наоборот, то программисту нет необходимости думать о том, что возвращает функция.

В Perl не существует типов "строка" или "число" или "файл" или что-то еще. Это контекстно зависимый полиморфный язык для работы с текстами. Скаляр имеет логическое значение "TRUE" (истина), если это не нулевая строка или число не равное 0.

В Perl существует два типа нулевых (null) скаляров - определенные (defined) и неопределенные (undefined). Неопределенное значение возвращается, когда что-то не существует. Например, неизвестная переменная, конец файла или ошибка. С помощью функции defined() вы можете заранее обнаружить подобное состояние.

Количество элементов массива так же является скаляром и начинается символами $#. Фактически $#var1 - это индекс последнего элемента массива. Нужно помнить, что первый элемент имеет индекс 0, поэтому количество элементов определяется как $#var1+1 . Присвоение значения $#var1

изменит длину массива и разрушит "оставленные" значения. Присвоение значения элементу массива с индексом больше чем $#var1 увеличит размер массива, а присвоение ему нулевого списка - обнулит.

В скалярном контексте имя массива возвращает его длину (для списка возвращается последний элемент):

```
@var1 = (4, 3, 2, 1);# Присвоение значения элементам массива
$i = @var1;          # Использование скалярного контекста
print $i;            # Печать результата 4 - кол-во элементов
print @var1;                 # Списковый контекст, печать всех элементов.
```

Для принудительного получения скалярного значения удобно применять функцию scalar():

```
print scalar(@var1);# Вывод длины массива а не его значений
```

Хеш в скалярном контексте возвращает "true", если существует хотя бы одна пара "ключ-значение". Фактически возвращается строка типа 2/8 где 8 - количество выделенных "ячеек" памяти, а 2 - количество использованных.

Конструкторы скаляров

Числа пишутся стандартно:

```
123
123.123
0.12
.12E-10
0xABCD          # Шестнадцатиричная запись
0377            # Если 0 в начале - восьмеричная
123_456_123     # Так тоже можно для удобства чтения.
```

В хеше можно опускать кавычки, если индекс не содержит пробелов:

$var1{first} то же что и $var1{'first'}

Обратите внимание на то, что перед первой одинарной кавычкой должен стоять пробел, иначе строка воспримется как имя переменной, т. к. в именах разрешено использование одинарных кавычек. Запрещается в кавычках применять

зарезервированные литералы __LINE__ (номер текущей строки программы), __FILE__ (текущий файл). Для обозначения конца программы можно применять литерал __END__. Весь последующий текст игнорируется, но его можно прочитать, используя указатель файла DATA.

Слова в программе, не поддающиеся никакой интерпретации, воспринимаются как строки в кавычках, поэтому рекомендуется имена меток и указателей файлов писать большими буквами во избежание возможного "конфликта" с зарезервированными словами.

Конструкторы списков

Список - множество значений, перечисленных через запятую и заключенных в круглые скобки. В списковом контексте список возвращает последний элемент списка:

@var1 = (1, 2, 'привет', 1.2); # Присвоить значение элементам.где

$var1[0] = 1,
$var1[1] = 2,
$var1[2] = 'привет'
$var1[3] = 1.2

$var1 = (1, 2, 'привет', 1.2);

а здесь $var1 = 1.2 т.е. последнее значение списка.

Допускается применять в списке другие списки, но в полученном списке уже невозможно различить начало и конец включенных списков:

@s1 = (1, 2, 3); # Первый список
@s2 = (6, 7, 8); # Второй
@s = (0, @s1, 4, 5, @s2, 9, 10); # Включаем списки @s1 и @s2
print @s; # Результат: 012345678910 - значения без пробелов.

Список без элементов обозначается, как (), и называется нуль-списком. Списковое выражение можно употреблять как имя массива, но при этом брать в круглые скобки:

print ('январь','февраль','март')[1];

Результат: февраль

Список может быть присвоен списку, только если каждый элемент в списке в левой части выражения допустим по типу списку в правой части:

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

Контрольные вопросы

1. Что Вы знаете о синтаксисе Perl?
2. Что Вы знаете о декларациях?
3. Что Вы знаете о простом предложении?
4. Что Вы знаете о сложных предложениях?
5. Что Вы знаете о блоках операторов switch?
6. Что Вы знаете о goto?
7. Что Вы знаете о POD операторах?
8. Что Вы знаете о переменных?
9. Что Вы знаете о скалярных значениях?
10. Что Вы знаете о контексте?

Рекомендуемые источники

1. Колдаев В.Д. Основы алгоритмизации и программирования: Учебное пособие / В.Д. Колдаев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: ИНФРА-М, 2017. - 416 с. [Электронный ресурс]. http://znanium.com/bookread.php?book=336649.
2. Гагарина Л.Г. Технология разработки программного обеспечения: Учеб. пос. / Л.Г.Гагарина, Е.В.Кокорева, Б.Д.Виснадул; Под ред. проф. Л.Г.Гагариной - М.: ИД ФОРУМ: НИЦ Инфра-М, 2015. - 400 с. [Электронный ресурс]. http://znanium.com/bookread.php?book=389963.
3. Голицына О. Л. Программирование на языках высокого уровня: Учебное пособие / О.Л. Голицына, И.И. Попов. - М.: Форум, 2016. - 496 с. [Электронный ресурс]. http://znanium.com/bookread.php?book=139428.