PERL

Sobre

Facilidade de manipulação de Strings

Desenvolvida inicialmente por Lary Wall, lançamento em 1987

Derivada de SNOBOL e awk, duas linguagens voltadas para processamento de texto

Sebesta: Linguagem de script - Imperativa

Híbrida: Pré compilada para o interpretador PERL

Variáveis

Tipo dinâmico e declaração implícita, acesso deve ser feito com o sinal da declaração. Máximo de 251 caracteres.

Variáveis

Tipo dinâmico e declaração implícita, acesso deve ser feito com o sinal da declaração. Máximo de 251 caracteres.

Variáveis escalares são declaradas com o sinal cifrão precedendo o nome da variável.

```
$age = 25;

$name = "John Paul";

$salary = 1445.50;

print "Age = $age\n";

print "Name = $name\n";

print "Salary = $salary\n";
```

Variáveis

Tipo dinâmico e declaração implícita, acesso deve ser feito com o sinal da declaração. Máximo de 251 caracteres.

Variáveis escalares são declaradas com o sinal cifrão precedendo o nome da variável.

Arrays são declarados com sinal de arroba precedendo o nome da variável.

```
@ages = (25, 30, 40);

@names = ("John Paul", "Lisa",
"Kumar");

push(@names, "Vladimir");

print "ages = @ages\n";

print "names[0] = $names[0]\n";

print "names[1] = $names[1]\n";

print "names[2] = $names[2]\n";

print "names[3] = $names[3]\n";
```

Variáveis

Tipo dinâmico e declaração implícita, acesso deve ser feito com o sinal da declaração. Máximo de 251 caracteres.

Variáveis escalares são declaradas com o sinal cifrão precedendo o nome da variável.

Arrays são declarados com sinal de arroba precedendo o nome da variável.

Hashes são declaradas com sinal de porcentagem precedendo o nome da variável.

```
%data = ('John Paul' => 45, 'Lisa' =>
30, 'Kumar' => 40);

$data{'Jack'} = 80;


print "$data{'John Paul'}\n";

print "$data{'Lisa'}\n";

print "$data{'Kumar'}\n";

print "$data{'Jack'}\n";
```

Variáveis

Tipo dinâmico e declaração implícita, acesso deve ser feito com o sinal da declaração. Máximo de 251 caracteres.

Variáveis escalares são declaradas com o sinal cifrão precedendo o nome da variável.

Arrays são declarados com sinal de arroba precedendo o nome da variável.

Hashes são declaradas com sinal de porcentagem precedendo o nome da variável.

Ponteiros são escalares e devem ser declarados com a referencia para quem apontam.

```
$var = 10;
$r = \$var;
print "Valor de \$var: ", $$r, "\n";
print "Tipo da ref.: ", ref($r), "\n";
@var = (1, 2, 3);
$r = \@var;
print "Valor de \@var: ", @$r, "\n";
print "Tipo da ref.: ", ref($r), "\n";
```

Operadores

Suporte a exponenciação via operador **.

<=> para comparação de igualdades – (10<=>20) = -1, (10<=>10) = 0, (20<=>10) = 1.

Comentários são feitos com cerquilha (#) e comentários em bloco são como:

=begin comment

...

=cut

Operadores

Comparadores de string separados. Operadores de strings funcionam para números, mas o contrário não vale.

lt – Verdadeiro se o operando da esquerda vier primeiro em ordem alfabética. ("abc" lt "def")

gt - Verdadeiro se o operando da esquerda vier depois em ordem alfabética. ("def" gt "abc")

le – Idem lt, mas abrange a igualdade.

ge – Idem gt, mas abrange a igualdade.

eq – Verdadeiro se são iguais.

ne – Verdadeiro se não são iguais

cmp – Função de comparação (semelhante a <=>). ("abc" cmp "def") = -1, ("abc" cmp "abc") = 0, ("def" cmp "abc") = 1

Operadores

Ponto (.) concatena strings (ret. escalar) - $a = "primeiro " . "segundo";

x repete a string da esquerda a quantidade de vezes indicada a direita (ret. escalar) – $b = ("string " x 3);

Dois pontos (..) fornecem números em um intervalo (ret. array) - @c = (0..10); @d = ('a'..'f');

Condicional

0 (zero), '0', "", deixar o if sem argumentos, e undef sempre são falsos, qualquer outra coisa é verdadeira.

```
if (condição){

}

elsif (condição){

}

else {

}
```

Unless executa se a condição for falsa.

```
unless (condição){

}
```

também tem elsif e else.

Switch não suportado nativamente

```
print "\$n e par" if (($n % 2) == 0);
```

Repetição

while - Enquanto for verdadeira

until - Enquanto for falsa

for - Inicialização ocorre uma vez, verificação de condição de parada e incremento acontece todas as vezes

do while - Idem while mas com verificação ao final

foreach - Usado para varrer listas

Possível colocar Labels para os comandos (exceto continue).

next - Retorna ao laço para a próxima execução.

last - Para a execução do laço (como break).

continue - Continua a execução de um while ou foreach.

redo - Retorna ao laço.

goto - Direciona a uma label.

Subrotinas

```
sub Soma{
   $tam = @_; # $tam = scalar @_;
   $soma = 0;
   for (my $i = 0; $i < $tam ; $i++) {
      $soma = $soma + $_[$i];
   }
   return $soma;
}
```

Retornos permitem qualquer coisa

```
$V = Soma(10, 20, 30, 40);
print "A soma e: $V";
```

my usado para variáveis locais.

local usado para variáveis que são locais mas precisam ser vistas por subrotinas.

state usado para variáveis que não devem ser reiniciadas quando a subrotina é chamada novamente. Requer: use feature 'state';

My

```
$var = 10; $a = 0; $b = 1;

print "Var = $var\n";

for (my $var = 0; $var < 5; $var++)
{

  print "Var = $var\n";

  my $var = 4;

  print "Var2 = $var\n";

}

print "Var = $var\n";
```

```
Var = 10

Var = 0

Var2 = 4

Var = 1

Var2 = 4

Var = 2

Var2 = 4

Var = 3

Var2 = 4

Var = 4

Var2 = 4

Var = 10
```

Subrotinas

```
sub PrintHash{
   my (%hash) = @_;
   foreach $item (%hash){
      print "Item : $item\n";
   }
}

%hash = ('name' => 'Tom', 'age' => 19);

PrintHash(%hash);
```

```
@list = (10, 20, 30);

$cref = \&Soma;

$result = &$cref(@list);
```

Nesse caso & continua sendo obrigatório

Formatação de impressão

```
format EMPLOYEE =

@<<<<<<<<<<<<<<<<< @<<  @#####.##

$name $age $salary

------------------------------

.

format EMPLOYEE_TOP =

================================

Nome               Idade   Salario

================================

.

select(STDOUT);

$~ = EMPLOYEE;

$^ = EMPLOYEE_TOP;
```

```
@n = ("Empregado1", "Empregado2",
"Empregado3", "Empregado4");

@a = (20, 30, 4005, 3.57);

@s = (2000.00, 2500.00, 4000000.000,
458.587);

$i = 0;

foreach (@n){

   $name = $_;

   $age = $a[$i];

   $salary = $s[$i];

   $i = $i + 1;

   write;

}
```

Expressões Regulares

Para encontrar padrões em strings.

Opção	Aplicação	Opção	Aplicação
+	1 ou mais	\w	[a-z] e [A-Z]
*	0 ou mais	\s	Espaço
?	1 ou 0	\d	[0-9]
{n}	Exatos n	[aeiou]	Pelo o menos alguma das
{n,}	Pelo o menos n		letras
{n,m}	Entre n e m	\W	Negação de \w
^	Começo da linha	\S	Negação de \s
$	Fim da linha	\D	Negação de \d
.	Qualquer Caractere	a | b	Tanto a como b

Expressões Regulares

Para encontrar padrões em strings.

São usadas com a string, seguidas por sinal de igualdade e til e a expressão.

Opção	**Aplicação**
i	Torna a avaliação não sensível ao caso
x	Permite espaços na expressão
s	Tratar tudo como única linha (. aceita quebras de linha)
g	Verificação/troca global.
e	Executar código para troca.

Expressões Regulares

m/padrão/ - Retorna verdadeiro se a string possuir o padrão.

```
$s = "frase com ausencia de criatividade";
```

```
if ($s =~ m/sencia/){
  print "sencia na frase\n";
}
if ($s =~ /CRIAT/i){
  print "CRIAT na frase\n";
}
if ($s =~ m/e+/){
  print "e pelo o menos uma vez\n";
}
```

Expressões Regulares

m/padrão/ - Retorna verdadeiro se a string possuir o padrão.

```
$s = "frase com ausencia de criatividade";
```

```
if ($s =~ m/x+/){
  print "x pelo o menos uma vez\n";
}
if ($s =~ m/\s/){
  print "espaco na frase\n";
}
if ($s =~ m/\d/){
  print "digitos na frase\n";
}
```

Expressões Regulares

m/padrão/ - Retorna verdadeiro se a string possuir o padrão.

s/padrão/troca/ - Troca o padrão sempre que for encontrado por troca.

```
$s = "frase com ausencia de criatividade";
```

```
$s =~ s/a/A/; #Primeiro a por A

$s =~ s/a/A/g; #a's por A's

$s =~ s/(.*)d/$1.D/e; #Último d por D

$s =~ s/[ea]/1/g; #a's e e's por 1

$s =~ s/e$/E/; #e no final da string por E
```

Expressões Regulares

m/padrão/ - Retorna verdadeiro se a string possuir o padrão.

s/padrão/troca/ - Troca o padrão por troca.

tr/padrão/troca/ - Troca padrão sempre por troca.

c – Complemento

d – Deleta

s – Remove duplicados

```
$s = "frase com ausencia de criatividade";

$s =~ tr/[a-z]/[A-Z]/; #uppercase
$s =~ tr/ae/1/; #a's e e's por 1
$s =~ tr/ae/12/; #a's por 1 e e's por 2
$s =~ tr/ae/1/d; #a's por 1 e apaga e's
$s =~ tr/ae/1/c; #todo o resto vira 1
$s =~ tr/a/12/; #a's viram 1 (ignora 2)
```

Legibilidade e Facilidade de Escrita

```
sub ImprimeLista1{
  $tam = @_;
  for (my $i = 0; $i < $tam ; $i++) {
    if ($_[$i] =~ m/e/){
      print "\u$_[$i]";
    }
  }
}
```

Legibilidade e Facilidade de Escrita

```
sub ImprimeLista1{
  $tam = @_;
  for (my $i = 0; $i < $tam ; $i++) {
    if ($_[$i] =~ m/e/){
      print "\u$_[$i]";
    }
  }
}
```

```
sub ImprimeLista2{
  foreach (@_){
    if (/e/){
      s/(\w)(\w+)/uc($1).$2/e;
      print;
    }
  }
}
```

Confiabilidade

```
$t = <STDIN>;
if ($t < 50.0){
  #Código genérico
}
```

```
$t = $a / $b;
for (; $t < 50; $t++){
  #Código genérico
}
```

Outras

Custo - Linguagem de código aberto, sem cursos tarifados. "Baixa legibilidade" = Manutenção cara.

https://perl5.git.perl.org/perl.git

Portabilidade - Windows, Mac, Linux e algumas versões de celulares.

Generalidade