Introdução à linguagem Perl

Assistente de ensino: Marcelo da Silva Reis[1]

Professor: Fabio Kon[1]

[1]Instituto de Matemática e Estatística, Universidade de São Paulo

MAC0211 - Laboratório de Programação I

10 de junho de 2010

IME - Instituto de Matemática e Estatística

Conteúdo (hoje):

Apresentação de Perl

- Origem da linguagem, principais características
- Executando programas em Perl

Tipos de variáveis

- Escalares, *arrays* e *hashes*
- Uso de referências
- Escopo das variáveis

Loops e construções condicionais

- For, while, foreach, ...

Exercícios

Para as próximas aulas:

- Expressões regulares: *matching*, processamento, ...
- Mais manipulação de *arrays*
- E/S, manipulação de arquivos
- Subrotinas
- Depurando códigos em Perl
- CGI/Perl
- Perl em Bioinformática (se der tempo ...)

Conteúdo

Resumo da história da linguagem

- Linguagem criada por Larry Wall em 1987
- Desenvolvida para processamento de textos
- **P**ractical **e**xtraction and **r**eport **l**anguage
- Hoje em dia utilizada para muitas outras aplicações:
  - administação de sistemas
  - bioinformática
  - aplicações *web*, etc.

Principais características

- Algumas influências: C, awk, Pascal, sed, Unix shell
- Desenvolvida para ser prática (fácil de usar, eficiente, completa), ao invés de "bela" (elegante, minimal) [1]
- Várias facilidades para processamento de texto estão "embutidas" na linguagem
- Atualmente na versão 5.12 (Perl 6 em desenvolvimento desde 2000).

[1] fonte: CPAN.org.

"Hello, World!"

Nosso primeiro programa em Perl (`hello-world.pl`):

```
#!/usr/bin/perl -w

use strict;

my $mensagem = "Hello" . ", " . 'world!';

print "$mensagem\n";
```

Executando programas em Perl

1. Utilizando diretamente o interpretador Perl:

   ```
   bash$ perl -w hello-world.pl
   ```

2. Alterando as permissões do arquivo .pl (o caminho do interpretador é especificado na primeira linha do código):

   ```
   #!/usr/bin/perl -w
   ```

Conteúdo

Tipos de variáveis

Os cinco tipos de variáveis fundamentais em Perl são:

- **escalares:** podem ser números, strings ou referências
- **array:** uma lista ordenada de escalares
- **hash:** um mapeamento de strings para escalares
- **manipulador de arquivo:** um mapeamento para um arquivo ou dispositivo
- **subrotina:** um mapeamento para uma subrotina

  Uma subrotina declarada é considerada variável, pois ela pode ser redefinida

Exemplos

Exemplos de declarações, uma variável de cada tipo:

```
my $foo;    # um escalar, default "undef"

my @foo;    # um array, default lista vazia

my %foo;    # um hash, default hash vazio
```

Cada tipo de variável (exceto subrotinas e arquivos) tem um *sigil* diferente.

Escalares

- Representam uma variável simples
- Podem ser strings, números ou referências

Escalares

- Representam uma variável simples
- Podem ser strings, números ou referências
- Com o "use strict" precisam ser declarados antes do uso. Exemplos:

```
my $dia = 10;
my $mes;
my $pi_nao_tao_preciso = 3.14;
$mes = "junho";
```

Escalares

- Representam uma variável simples
- Podem ser strings, números ou referências
- Com o "use strict" precisam ser declarados antes do uso. Exemplos:

```
my $dia = 10;
my $mes;
my $pi_nao_tao_preciso = 3.14;
$mes = "junho";
```

- "Castings" automáticos entre tipos. Exemplo:

```
print "Hoje, $dia/$mes, tem aula de MAC0211\n";
```

Arrays

Arrays em Perl são tratados como uma lista de escalares.
Exemplos:

```
my @meses = ("maio", $mes, "julho");   # $mes == "junho"

my @numeros = (13, 42, 3);

my @mistura   = ("jan", 42, 3.14);
```

Mais sobre *arrays*

- Como em C, *arrays* começam com índice zero. Exemplo:

```
if ($dias[0] eq 'dom'){
    ...
}
```

Mais sobre *arrays*

- Como em C, *arrays* começam com índice zero. Exemplo:

```
if ($dias[0] eq 'dom'){
    ...
}
```

- "Modo escalar":

```
if (@dias <= 7){    # == scalar(@dias)
    ...
}
```

Mais um pouquinho sobre *arrays*

- Ordenando um *array* em ordem crescente (numérica ou lexicográfica):

```
my @numeros_ordenados = sort @numeros;
my @meses_ordenados = sort @meses;
```

Mais um pouquinho sobre *arrays*

- Ordenando um *array* em ordem crescente (numérica ou lexicográfica):

```
my @numeros_ordenados = sort @numeros;
my @meses_ordenados = sort @meses;
```

- Invertendo a ordem do *array*:

```
my @numeros_inv = reverse @numeros;
```

Hashes

- Em Perl, *hashes* são uma coleção de escalares indexados por chaves (um único elemento por chave). Exemplo:

```
my %meses = ("1", "jan", "2", "feb");
```

Hashes

- Em Perl, *hashes* são uma coleção de escalares indexados por chaves (um único elemento por chave). Exemplo:

```
my %meses = ("1", "jan", "2", "feb");
```

- Uma outra declaração para o *hash* acima:

```
my %meses = (1 => "jan", 2 => "feb");
```

Hashes

- Em Perl, *hashes* são uma coleção de escalares indexados por chaves (um único elemento por chave). Exemplo:

```
my %meses = ("1", "jan", "2", "feb");
```

- Uma outra declaração para o *hash* acima:

```
my %meses = (1 => "jan", 2 => "feb");
```

- Acessando um valor de um *hash*:

```
$meses{"1"};    # devolve "jan"
```

Alocação dinâmica de *arrays* e de *hashes*

- A alocação de *arrays* e de *hashes* sempre é dinâmica, mesmo quando inicializando a variável com valores.

Alocação dinâmica de *arrays* e de *hashes*

- A alocação de *arrays* e de *hashes* sempre é dinâmica, mesmo quando inicializando a variável com valores.
- O interpretador gerencia para a gente a alocação dinâmica da memória! :-)

Alocação dinâmica de *arrays* e de *hashes*

- A alocação de *arrays* e de *hashes* sempre é dinâmica, mesmo quando inicializando a variável com valores.
- O interpretador gerencia para a gente a alocação dinâmica da memória! :-)
- Em um *hash*, chaves não-inicializadas têm valor padrão `undef`:

```
my %meses = (1 => "jan", 2 => "feb");
if(!defined( $meses{3} ) ){
    $meses{3} = "mar";
}
```

Uso de referências

1. Uma *referência* é um escalar que aponta para uma variável qualquer

Uso de referências

1. Uma *referência* é um escalar que aponta para uma variável qualquer

2. Ou seja, um escalar pode ser referência para arrays e *hashes*

Uso de referências

1. Uma *referência* é um escalar que aponta para uma variável qualquer

2. Ou seja, um escalar pode ser referência para arrays e *hashes*

3. Arrays e *hashes* são coleções de escalares

Uso de referências

1. Uma *referência* é um escalar que aponta para uma variável qualquer
2. Ou seja, um escalar pode ser referência para arrays e *hashes*
3. Arrays e *hashes* são coleções de escalares
4. Corolário: podemos utilizar arrays e *hashes* com seus elementos (escalares) sendo referências para outros arrays e *hashes*

Exemplo de referências

```
# Para refer\^encia utilizamos {} no lugar de ()
#
$hash = { desc => "um escalar", sigil => '$' };

print $hash->{"sigil"}; # imprime '$'


# Para refer\^encia utilizamos [] no lugar de ()
#
$array = [ 1958, 1962, 1970, 1994, 2002, 2010 ];

print $array->[5];  # ser\'a?!
```

Exemplo (adaptado do CPAN) de *hashes* em um *hash*

```
my %var = (
    scalar  => {
                 desc => "unico item",
                 sigil => '$'
               },
    array   => {
                 desc => "lista ordenada de itens",
                 sigil => '@'
               },
    hash    => {
                 desc => "pares de chave/item",
                 sigil => '%'
               }
);

print "Escalares tem um $var{'scalar'}->{'sigil'}";
```

IME - Instituto de Matemática e Estatística

Outro exemplo

```
my @vetor = (42, "towel");

my %var = (
        array   =>  [ @vetor ],
        hash    => {
                      desc => "key/value pairs",
                      sigil => '%' }
);

print "A Verdade: $var{'array'}->[0]\n";
```

Variáveis especiais

Perl tem várias variáveis especiais; algumas delas:

```
$_

@_

@ARGV

%ENV

$1, $2, $3, ...
```

Escopo das variáveis

- É possível declarar variáveis sem utilizar o `my`:

```
$pi = 3.14;
```

Escopo das variáveis

- É possível declarar variáveis sem utilizar o `my`:

```
$pi = 3.14;
```

- Todavia, isso cria uma variável global onde quer que a variável seja declarada, o que é uma má prática de programação.

Escopo das variáveis

- É possível declarar variáveis sem utilizar o `my`:

  ```
  $pi = 3.14;
  ```

- Todavia, isso cria uma variável global onde quer que a variável seja declarada, o que é uma má prática de programação.

- **Solução**: utilizar o `my` (que cria variáveis locais, caso a declaração seja dentro de laços e/ou de subrotinas).

  Melhor ainda: utilizar o `my` em conjunto com o `use strict`

Conteúdo

For e While

São muito parecidas com as suas equivalentes em C:

```
for (my $i = 0; $i <= 10; $i++){
    ....
}

while( condicao ){
    ....
}

do{
    ....
}while( condicao );
```

Um exemplo interessante de while

```
while(<STDIN>){
   # captura em $_ uma linha da entrada padr\~ao
   # e dentro do loop pode ser realizado
   # algum processamento utilizando o $_
}
```

O comando chomp remove o caracter de fim de linha de uma variável.

If, then, else,...

Também é bem parecido com o de C:

```
if ( ( condicao_1 ) && ( condicao_2 ) ){
    ....
}
elsif ( ( condicao_3 ) || (condicao_4) ){
    ....
}
else{
    ....
}
```

Condicionais

Em Perl é possível realizar construções condicionais que realizam ações de acordo com a avaliação do primeiro termo:

```
1 && 1 || 0 and print "1 == true! :-)";

#
# Se a funcao devolve 1, imprime "Encontrado!",
#          caso contrario imprime "Xii.."
#
busca($element) and print "Encontrado!" or print "Xii..";
```

Foreach

O loop `foreach` é muito mais amigável para a manipulação de listas e de *hashes*. Dois exemplos com listas:

```
foreach (@meses) {
    print "Mes: $_\n";
}
```

```
print $numeros[$_] foreach 0 .. 2; # array com 3 elem.
```

Conteúdo

Exercício I (Learning Perl, 3.1)

Escreva um programa em Perl que leia da entrada padrão uma lista de strings (uma palavra por linha) e, ao final do processo, imprima a lista em ordem reversa.

Dicas (sintaxes úteis):

```
while(<STDIN>){
   # captura em $_ uma linha da entrada padr\~ao
}

my @array = reverse @outro_array;
$array[2] = "blabla";

chomp $_;

print "Imprimindo e " . $array[2] . "concatenando!\n";
```

IME - Instituto de Matemática e Estatística

Exercício II

Escreva um programa em Perl que leia da entrada padrão inteiros positivos e armazene-os em duas listas: uma para números menores que 10 e outra para maiores. As duas listas devem ser acessadas através de um hash, usando as chaves “menores” e “maiores”.

Dicas (sintaxes úteis):

```
while(<STDIN>){
   # captura em $_ uma linha da entrada padr\~ao
}
```

```
1 < 0 and print "Zero > Um!" or print "Zero <= Um!";
```

```
my $hash = (um => [@array_1], dois => [("a", "b")]);
```

Referências

1. Perl.org. `http://www.perl.org/`.
   Acesso em 9 de junho de 2010.

2. Comprehensive Perl Archive Network.
   `http://www.cpan.org/`.
   Acesso em 9 de junho de 2010.

3. Livros da O'Reilly:

   - *Learning Perl*.
   - *Programming Perl*.