Mais sobre a 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

15 de junho de 2010

IME - Instituto de Matemática e Estatística

Conteúdo

"Previously on MAC211..."
- Revisão da aula anterior

Mais sobre *arrays* e *hashes*
- Manipulando elementos de *arrays*
- Iterando *hashes*
- Exercício

Expressões regulares
- Casamento (*matching*)
- Substituições
- Filtragem
- \+ Exercício

Para a próxima aula

- E/S, manipulação de arquivos
- Subrotinas
- Depurando códigos em Perl
- Exercícios

(CGI fica como material suplementar)

Conteúdo

"Previously on MAC211..."
Revisão da aula anterior

Mais sobre *arrays* e *hashes*
Manipulando elementos de *arrays*
Iterando *hashes*
Exercício

Expressões regulares
Casamento (*matching*)
Substituições
Filtragem
+ Exercício

Características e Aplicações

- Originalmente projetada para processamento de textos
  - Várias facilidades para tal estão "embutidas" na linguagem
- Hoje em dia também utilizada para muitas outras aplicações:
  - administação de sistemas
  - bioinformática
  - aplicações *web*, etc.
- Desenvolvida para ser prática (fácil de usar, eficiente, completa), ao invés de "bela" (elegante, minimal) [1]

[1] fonte: CPAN.org.

Tipos de dados

Os cinco tipos de dados fundamentais em Perl são:

- **escalares:** podem ser números, strings ou referências
- **array:** armazenamento indexado de uma lista 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

Exemplos

Exemplos de declarações:

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

my @bar;    # um array, default lista vazia

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

Uso de referências

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

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

3. *Arrays* e *hashes* são coleções de escalares

4. Segue imediatamente que podemos utilizar *arrays* e *hashes* com seus elementos (escalares) sendo referências para outros arrays e *hashes*

Exemplo de referências

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

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


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

print $array->[5];  # quem sabe? :-)
```

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

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

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

IME - Instituto de Matemática e Estatística

Outro exemplo

```
my @vetor = (42, 13, 7);

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

print "Posicao 0 do vetor: $var{'array'}->[0]\n";
```

Foreach

O loop `foreach` é amigável para a manipulação de listas armazenadas em *arrays* (e, indiretamente, também de *hashes*):

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

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

Conteúdo

"Previously on MAC211..."
Revisão da aula anterior

Mais sobre *arrays* e *hashes*
Manipulando elementos de *arrays*
Iterando *hashes*
Exercício

Expressões regulares
Casamento (*matching*)
Substituições
Filtragem
+ Exercício

função split

A função `split` devolve uma lista de palavras de um *string*, separadas por um delimitador. Exemplo:

```
@array = split " ", "Foo Bar Baz";

foreach my $valor (@array){
  print $valor . " : ";
}
#
# Imprime "Foo : Bar : Baz : "
#
```

Inserindo e removendo elementos

- Utilizando funções nativas da linguagem, podemos inserir e remover elementos em *arrays* com facilidade.

Inserindo e removendo elementos

- Utilizando funções nativas da linguagem, podemos inserir e remover elementos em *arrays* com facilidade.

  `unshift:` insere um elemento à esquerda de um *array*

  `shift:` remove um elemento à esquerda de um *array*

  `push:` insere um elemento à direita de um *array*

  `pop:` remove um elemento à direita de um *array*

Inserindo e removendo elementos

- As operações atualizam automaticamente os índices dos escalares no *array*

Inserindo e removendo elementos

- As operações atualizam automaticamente os índices dos escalares no *array*

- Remoção de elementos de *array* vazio devolve `undef`

Inserindo e removendo elementos

- As operações atualizam automaticamente os índices dos escalares no *array*
- Remoção de elementos de *array* vazio devolve `undef`
- Exemplo:

```
my @array;     # array vazio
my $elem1;
my $elem2 = "a";
my $elem3 = "b";

push @array, $elem1;  #  (undef)

unshift @array, ($elem2, $elem3);  #  ("a", "b", undef)

my $elem4 = shift @array;  # elem4 eq "a"
```

Filas e pilhas

- Com as operações de inserção e de remoção, podemos trabalhar com filas e pilhas com facilidade!

Filas e pilhas

- Com as operações de inserção e de remoção, podemos trabalhar com filas e pilhas com facilidade!
- Para pilhas: `push`/`pop` ou `shift`/`unshift`
- Para filas: `push`/`unshift` ou `shift`/`pop`

Filas e pilhas

- Com as operações de inserção e de remoção, podemos trabalhar com filas e pilhas com facilidade!
- Para pilhas: `push`/`pop` ou `shift`/`unshift`
- Para filas: `push`/`unshift` ou `shift`/`pop`
- Número de elementos de uma fila ou pilha:

  ```
  scalar(@array) == $#array + 1
  ```

  - `$#array` é o índice do último elemento de `@array`

    Se `@array` tem zero elementos, então `$#array == -1`

Ordenando elementos

- Para ordenarmos elementos de uma lista ou de um *array*, utilizamos a função `sort`:

```
my @array = sort ('b', 'c', 'a');
# @array == ('a', 'b', 'c')
```

Ordenando elementos

- Para ordenarmos elementos de uma lista ou de um *array*, utilizamos a função `sort`:

```
my @array = sort ('b', 'c', 'a');
# @array == ('a', 'b', 'c')
```

- O padrão é a ordenação lexicográfica; para ordenação numérica, precisamos deixar isso explícito:

```
my @array1 = sort {$a <=> $b} (2, 3, 1);
# @array1 == (1, 2, 3)

my @array2 = reverse sort {$a <=> $b} (2, 3, 1);
# @array2 == (3, 2, 1)
```

Iterando os elementos de *hashes*

A função `keys` devolve uma lista das chaves de um *hash*:

```
my %hash = ("a" => "Foo", "b" => "Bar");

my @array = keys %hash;

print $array[$_] . " " foreach 0..$#array;
# imprime "a b"
```

`foreach` em *hashes*

Usando `keys` podemos iterar um *hash* a partir de suas chaves:

```
my %hash = ("a" => "Foo", "b" => "Bar");

foreach my $chave (keys %hash) {
        print $chave . ":" . $hash{$chave} . " ";
}
# imprime "a:Foo b:Bar "
```

foreach em *hashes*

Usando `keys` podemos iterar um *hash* a partir de suas chaves:

```
my %hash = ("a" => "Foo", "b" => "Bar");

foreach my $chave (keys %hash) {
        print $chave . ":" . $hash{$chave} . " ";
}
# imprime "a:Foo b:Bar "
```

Usando a função `values` podemos iterar um *hash* a partir de seus valores:

```
foreach my $valor (sort values %hash) {
        print $valor . " ";
}
# imprime "Bar Foo "
```

Exercício

Escreva um programa que leia da entrada padrão um arquivo texto, conte o número de ocorrências das palavras no mesmo e, ao final, imprima na saída padrão a lista das palavras (em ordem lexicográfica) e suas respectivas frequências. Dicas:

```
@array = split " ", $string;

!defined($var) and $var = 0 or $var++;

foreach my $chave (keys $hash){
  print $chave . " " . $hash{$chave} . "\n";
}

@array = sort ('b', 'c', 'a'); # ('a', 'b', 'c')
```

Obs: suponha que o arquivo não contenha pontuação.

IME - Instituto de Matemática e Estatística

Conteúdo

"Previously on MAC211..."
Revisão da aula anterior

Mais sobre *arrays* e *hashes*
Manipulando elementos de *arrays*
Iterando *hashes*
Exercício

Expressões regulares
Casamento (*matching*)
Substituições
Filtragem
+ Exercício

Expressões regulares

Expressões regulares têm amplo suporte na linguagem Perl; vamos introduzir algumas operações básicas:

Matching: trata-se da operação mais elementar. Exemplos:

```
if (/foo/){ ... }  # verdadeiro se $_ contem "foo"

if ($tmp =~ /foo/){  # verdadeiro se $tmp contem "foo"
   ...
}
```

Mais *matching*

```
if(/\d+[^\d+]$/){
  #
  # verdadeiro se $_ contem 1 ou mais digitos
  # seguido(s) de um ou mais nao digitos ate'
  # ao final de $_ .
}
```

```
$tmp =~ /^MAC211\s+is\s+.*$/ and print "Ok!";
#
# verdadeiro se $tmp contem "MAC211" no inicio,
# seguido de um ou mais espacos, "is", um ou mais
# espacos e um string de zero ou mais caracteres.
```

Substituições

Substituições são muito úteis na manipulação de strings:

```
s/foo/bar/; # substitui foo por bar em $_
```

```
$tmp =~ s/foo/bar/;
# substitui o primeiro foo encontrado por bar em $tmp
```

```
$tmp =~ s/foo/bar/g;
# substitui TODOS os foo por bar em $tmp
```

Filtragem (captura)

Outra coisa legal é a filtragem de textos utilizando expressões regulares. Exemplo de filtragem dos campos de um email:

```
if ($email =~ /([^\@]+)\@(.+)/) {
        print "Nome: $1\n";
        print "Hospedeiro: $2\n";
}
```

As expressões capturadas são definidas pelos parênteses, sendo armazenadas nas variáveis $1, $2, etc.

Mais exemplos de filtragem

```
$frase = "Use the Fork!";

$frase =~ /^Use\s+the\s+(.+)/
     and print "Dont use the " . $1;
#
# Verdadeiro, filtra "Fork!"
```

Sempre é bom testar se houve *matching* ao proceder a filtragem, pois em caso negativo todas as variáveis `$1`, `$2`, etc. ficam setadas como `undef`.

Exercício

Escreva um programinha Perl que leia da entrada padrão linhas contendo siglas de disciplinas do BCC e imprima na tela todas as siglas de matérias do DCC (*e.g.*, MAC211, mac5711), ao lado de seu respectivo número de ocorrências. Dicas:

```
while(<STDIN>){
  chomp $_; # remove o '\n' do $_
  ...
}

$string = uc $_;  # [a-z] -> [A-Z]

@array = split " ", $string;

$variavel =~ /a(bc)d/;
# $1 eq 'bc', precedido de 'a' e sucedido por 'd'
```

IME - Instituto de Matemática e Estatística

Mais dicas do exercício 2

```
!defined($var) and $var = 0 or $var++;

foreach my $chave (sort keys %hash){
  print $chave . " " . $hash{$chave} . "\n";
}

# Matchings:
# \s+ -> com um ou mais espacos
# \w -> com um caractere alfa-numerico
# [^\d] -> com um caractere nao-digito
# .+ -> com um ou mais qualquer coisa
# \d? -> com zero ou um digito
# ^\w* -> com zero ou mais alfa-numericos no inicio
# \d+$ -> com um ou mais digitos ao final
```

Referências

1. Perl.org. `http://www.perl.org/`.
   Acesso em 13 de junho de 2010.

2. Comprehensive Perl Archive Network.
   `http://www.cpan.org/`.
   Acesso em 13 de junho de 2010.

3. Livros da O'Reilly:
   - *Learning Perl*.
   - *Programming Perl*.

4. Verbete "Foo bar" na Wikipédia.
   `http://en.wikipedia.org/wiki/Foo_bar/`.
   Acesso em 10 de junho de 2010.