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

16 de junho de 2009

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, *hashes*, ...

Loops e construções condicionais

- For, while, foreach, ...
- Exercícios

Expressões regulares

- Matching, substituições, ...

E/S

- Entrada padrão, arquivos
- \+ Exercícios

Para quinta-feira:

- Subrotinas
- Depurando códigos em Perl
- CGI/Perl
- Perl em Bioinformática

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 5.10 (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

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 pré-declarados. Exemplos:

```
my $dia = 16;
my $mes = "junho";
my $pi_nao_tao_preciso = 3.14;
```

Escalares

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

```
my $dia = 16;
my $mes = "junho";
my $pi_nao_tao_preciso = 3.14;
```

- "Castings" automáticos entre tipos. Exemplo:

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

Arrays

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

```
my @dias = ("dom", "seg", "ter");

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

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

Mais sobre arrays

- Arrays são zero-indexados. Exemplo:

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

Mais sobre arrays

- Arrays são zero-indexados. Exemplo:

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

- "Modo escalar":

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

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 vetor:

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

Hashes

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

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

Hashes

- Em Perl, *hashes* são uma coleção de valores que são 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 valores que são 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"
```

Variáveis especiais

Perl tem várias variáveis especiais; algumas delas:

```
$_

@_

@ARGV

%ENV

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

(o significado de cada uma delas será explicado nas duas aulas)

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 padrao
   # e dentro do laco 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 ){
    ....
}
elsif ( condicao 2 ){
    ....
}
else{
    ....
}
```

Foreach

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

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

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


foreach my $chave (keys %meses) {
        print "Mes: $meses{$chave}\n";
}
```

Exercício (Learning Perl, 3.1)

Escreva um programa em Perl que leia da entrada padrão uma lista de strings e, ao final do processo, imprima a lista em ordem reversa (para fazer depois no PC, utilize Control+D ou o redirecionamento de arquivo).

Dicas (sintaxes úteis):

```
while(<STDIN>){ .... }
```

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

```
chomp $_;
```

Conteúdo

Expressões regulares

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

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

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

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

Substituições

São operações 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
```

Processamento (captura ou *parsing*)

Outra coisa legal é o processamento de expressões regulares. Exemplo (*parsing* de um email):

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

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

Conteúdo

Entrada padrão

Como vimos anteriormente, o STDIN é uma variável de arquivo que pode ser utilizada para ler linhas da entrada padrão:

```
while(<STDIN>){ .... }

my $uma_linha = <STDIN>;

my @varias_linhas = <STDIN>;
```

Arquivos

O básico para manipulação de arquivos é simples:

```
open(IN,  "<" ,  "input.txt" ) or die "Erro!";
open(OUT, ">" ,  "output.txt") or die "Erro!";
open(APP, ">>",  "append.txt") or die "Erro!";

printf OUT "Escrevendo no arquivo output.txt\n";

while(<IN>){ print $_; printf APP $_; }

close(IN);
close(OUT);
close(APP);
```

+ Exercício

Escreva um programinha Perl que abra um arquivo e imprima na tela todas as siglas de matéria do DCC (Exemplo: MAC211, mac5711), ao lado do seu número de ocorrências no arquivo.

Dicas:

```
open($arq, "<", "arq.txt") or die "blabla";

$string = uc $_; # joga todo o texto para u.case

$variavel =~ /b(bl)a/;
# $1 == 'bl', precedido de 'b' e sucedido por 'a'

# Matchings:
# \s+ -> com um ou mais espacos
# \w -> com uma ou mais caracteres alfa-numericos
# \.+ -> com um ou mais qualquer coisa
# \d* -> com zero ou mais numeros
```

Referências

1. Perl.org. `http://www.perl.org/`.
   Acesso em 10 de junho de 2009.

2. Comprehensive Perl Archive Network.
   `http://www.cpan.org/`.
   Acesso em 16 de junho de 2009.

3. Livros da O'Reilly:

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