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

18 de junho de 2009

IME - Instituto de Matemática e Estatística

Conteúdo

Perl em Bioinformática

- dada a restrição de tempo, não será possível mostrar, nesta aula, aplicações de Bioinformática que usam Perl... :-(

- todavia, seria possível apresentar um seminário nesse sentido, no segundo semestre

Conteúdo

Características e Aplicações

- Originalmente utilizada para processamento de textos
  - Várias facilidades para processamento de texto 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:** 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 (embora fazer isso não seja uma boa)

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

sub foo { ... }    # uma subrotina

# manipuladores de arquivos nao sao declarados:
#
open(FOO, ... ) # (uppercase opcional)
```

Uso de referências

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

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*

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

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 (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, 13, 7];

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

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

Manipulando arquivos

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

(a documentação do CPAN utiliza `$foo` para manipulação)

IME - Instituto de Matemática e Estatística

Foreach

O loop `foreach` é 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";
}
```

Expressões regulares

Exemplo de *matching* e processamento de expressões regulares:

```
$email =~ /([^@]+)@(.+)/;
defined $2 and print "User: $1\nHost: $2\n";
```

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

Conteúdo

Um exemplo simples de subrotina

```
sub logger {
  my $mensagem = $_[0];
  open my LOG, ">>", "meu.log" or die "Erro!";
  print $logfile $logmessage;
}

logger "Teste de subrotina";
```

- Como toda variável em Perl, subrotinas podem ser declaradas em qualquer parte do código!

Um exemplo simples de subrotina

```
sub logger {
  my $mensagem = $_[0];
  open my LOG, ">>", "meu.log" or die "Erro!";
  print $logfile $logmessage;
}

logger "Teste de subrotina";
```

- Como toda variável em Perl, subrotinas podem ser declaradas em qualquer parte do código!
- Porém, é uma boa prática declarar todas no início ou no final

Passando parâmetros

Um exemplo simples:

```
imprime_alunos (\@nome, \%idade, $numero_alunos);
```

No exemplo acima, os dois primeiros parâmetros são passados por referência, enquanto o último é passado por valor.

```
sub imprime_alunos {
  my ($nome, $age, $r) = @_;
  print "$nome->[$_] $age->{$nome->[$_]}" foreach 0..$r;
}
```

Os argumentos da subrotina são armazenadas no array `@_`:

Devolvendo valores

```
sub teste {
  return (1, 2, 3);
}
```

Em alguns casos, no final da subrotina, pode-se omitir o `return`:

```
sub lista {
  my $gamiarra = shift;  # shift @_;
  if ($gambiarra){
    return (1 => "foo", 2 => "bar");
  }
  undef;   # mesma coisa que "return undef"
}

my $verifica_gambiarra = lista($variavel);
```

Exercício

Elabore uma subrotina em Perl que receba uma lista de nomes, dois *hashes* de notas de provas (cujas chaves são os nomes da lista) e devolva um *hash* com a média aritmética, também mapeado pelos nomes. **Ferramentas:**

```
$c = $a / $b;
$ref_hash = \%hash;

sub minha_subrotina{
  my ($x, $y, $z) = @_;
  ...
  return $resultado;
}

foreach my $chave (keys %hash){ ... $hash{$chave} ... }
foreach (@array){ ... $_ # elem. de @array ... }
```

IME - Instituto de Matemática e Estatística

Conteúdo

Corrigindo programas em Perl

- Antes de tudo, muitos problemas são evitados utilizando a flag `-w` quando se executa um código Perl
  - Utilizar o `use warnings;` tem o mesmo efeito
  - O `use strict;` também é útil para prevenir erros
  - Executar `perl -c meu_codigo.pl` verifica sintaxe e typos

Corrigindo programas em Perl

- Antes de tudo, muitos problemas são evitados utilizando a flag `-w` quando se executa um código Perl
  - Utilizar o `use warnings;` tem o mesmo efeito
  - O `use strict;` também é útil para prevenir erros
  - Executar `perl -c meu_codigo.pl` verifica sintaxe e typos
- Feito isso, é comum corrigir código Perl utilizando o "comentar e imprimir"

Corrigindo programas em Perl

- Antes de tudo, muitos problemas são evitados utilizando a flag `-w` quando se executa um código Perl
  - Utilizar o `use warnings;` tem o mesmo efeito
  - O `use strict;` também é útil para prevenir erros
  - Executar `perl -c meu_codigo.pl` verifica sintaxe e typos
- Feito isso, é comum corrigir código Perl utilizando o "comentar e imprimir"
- Se os passos acima não resolvem, podemos utilizar o depurador Perl

Iniciando o depurador Perl

- Para iniciar o depurador Perl, basta digitarmos na linha de comando:

```
bash$ perl -d programa_bixado.pl
```

- Alternativamente, podemos modificar a primeira linha do código:

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

Alguns comandos básicos do depurador

- `h` : exibe uma tela de ajuda (`h h` para ajuda detalhada)
- `b n` : define um breakpoint na linha n (ex: `b 42`)
- `p var` : imprime o estado de uma variável (ex: `p $pe`)
- `r` : inicia a execução (`R` para reiniciar uma)
- `q` : sai do depurador

Exemplo

Agora vamos ver um exemplo simples de depuração de código Perl.

Conteúdo

O que é CGI?

- **C**ommon **G**ateway **I**nterface
- protocolo para chamar aplicações remotas através de um servidor *web*
- Tá, mas é qual a utilidade disso?

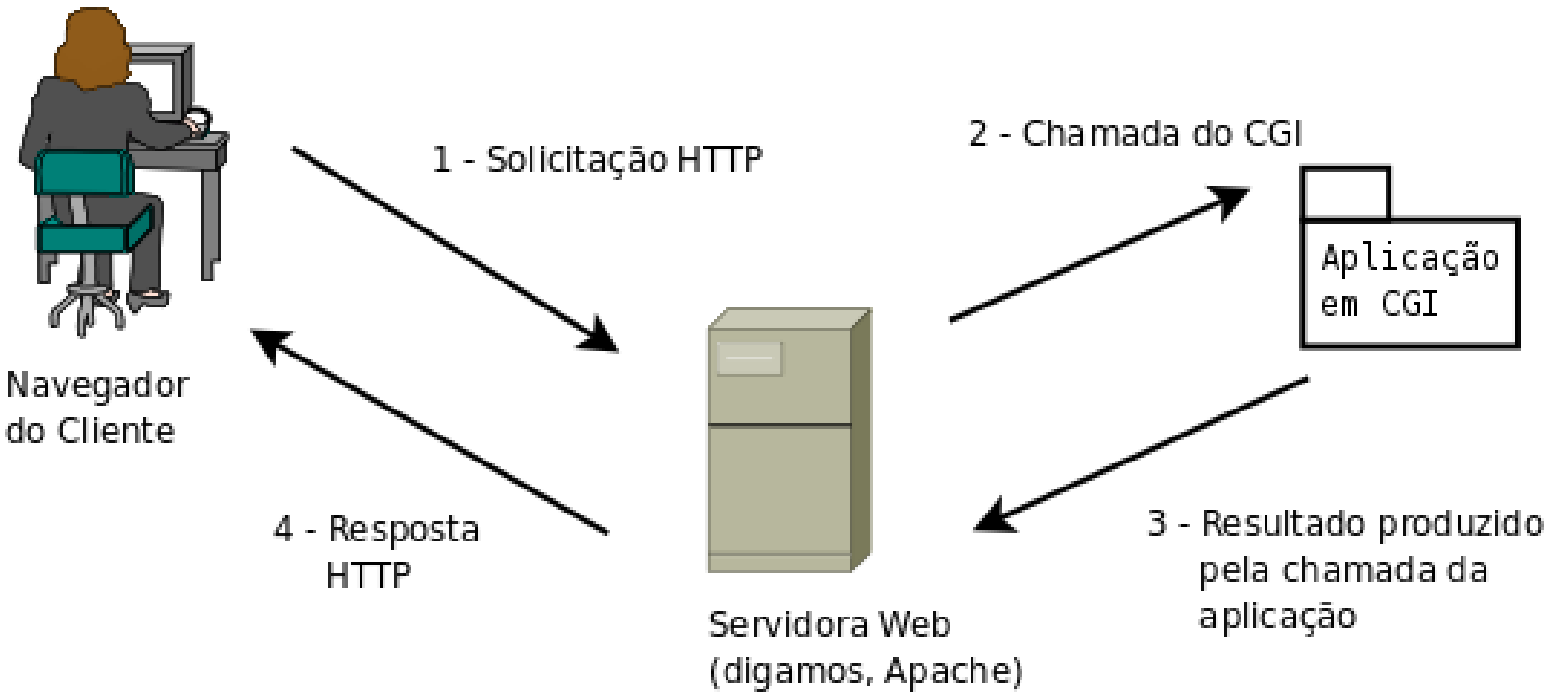
O que é CGI?

- **C**ommon **G**ateway **I**nterface
- protocolo para chamar aplicações remotas através de um servidor *web*
- Tá, mas é qual a utilidade disso?
  - permite disponibilizar recursos dinâmicos na *web*;
  - ao contrário de recursos estáticos (ex: páginas HTML), os dinâmicos contêm informações que podem mudar a cada solicitação

O que é CGI?

- **C**ommon **G**ateway **I**nterface
- protocolo para chamar aplicações remotas através de um servidor *web*
- Tá, mas é qual a utilidade disso?
  - permite disponibilizar recursos dinâmicos na *web*;
  - ao contrário de recursos estáticos (ex: páginas HTML), os dinâmicos contêm informações que podem mudar a cada solicitação
- Perl é, de longe, a linguagem mais utilizada em scripts CGI

Uma chamada de um script CGI



Roteiro de execução de CGI / Perl

1. Seu navegador solicita a uma servidora *web* a execução de um script CGI, com ou sem parâmetros. Exemplo:

   `http://www.usp.br/cgi-bin/foo.pl?p1=bar&p2=baz`

2. A servidora *web* recebe a solicitação, executa o programa Perl com os parâmetros (se houverem)

3. O programa Perl é processado com os parâmetros recebidos e devolve ao Apache um resultado (normalmente uma página HTML)

4. A servidora *web* devolve ao computador que fez a solicitação o resultado produzido pelo programa Perl

Servidora HTTP Apache

- servidora *web* de código aberto e livre
- uma dos componentes do *solution stack* LAMP [2]

LAMP: **L**inux, **A**pache, **M**ySQL, **P**erl

[2] `http://www.think-lamp.com/`

Mensagens HTTP

- Métodos :
  - GET: sem conteúdo no corpo na mensagem HTTP
    (o Apache passa os parâmetros para o CGI através da URL)
  - POST: com conteúdo no corpo da mensagem, em formato de string
    (o Apache passa os parâmetros utilizando o STDIN)
- Cabeçalho:
  - `Content-type`: text/html, application/pdf, etc.
  - `Host`: nome do host solicitante

Exemplos

Solicitação:

```
GET /cgi-bin/hello.pl HTTP/1.1
Host: localhost
```

Resposta:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 70
<HTML>
<HEAD><TITLE>Hello, world!</TITLE>
<BODY><P>Hello, World!</BODY>
</HTML>
```

“Hello, world!” em CGI / Perl

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

use strict;

print "Content-type: text/html\n\n";

print<<END_OF_PAGE;
<HTML>
<HEAD><TITLE>Hello, world!</TITLE>
<BODY><P>Hello, World!</BODY>
</HTML>
END_OF_PAGE
```

O ambiente de um programa CGI

- Permissões de scripts: normalmente eles têm as mesmas permissões que o servidor *web*
  - Por razões de segurança, esses scripts devem ser capazes de ler e de escrever apenas em áreas de seu domínio
- o E/S é feito principalmente através dos ponteiros `STDIN`, `STDOUT` e STDERR
- informações adicionais são coletadas através de variáveis de ambiente (em `%ENV`)

Ponteiros de arquivos em CGI / Perl

- `STDIN`: o programa lê os parâmetros do método POST através dele (nunca o leia quando o método for GET, pois pode "matar" a execução)
- `STDOUT`: o programa escreve a resposta ao Apache neste ponteiro
- `STDERR`: quando ocorre um erro, o Apache pode anexar a mensagem em `STDERR` à resposta a ser enviada ao cliente

Algumas variáveis de ambiente

CONTENT_LENGHT	o comprimento dos dados (em bytes) enviados em STDIN
QUERY_STRING	as informacoes na URL que estao apos o "?"
REQUEST_METHOD	o metodo utilizado na solicitacao (POST, GET, etc)

Exemplo de uso:

```
if ($ENV{REQUEST_METHOD} eq "GET"){
  $mensagem = $ENV{QUERY_STRING};
}
```

CGI.pm

- módulo Perl para programação de scripts CGI
- fornece uma boa API para manipular os dados de entrada e de saída (HTML ou XHTML)
- duas opções de interface: procedural e orientada à objetos
- página do CGI.pm:
  `http://stein.cshl.org/WWW/software/CGI/`

Exemplos de aplicações

Agora vamos ver alguns exemplos de aplicações.

+ Exercício (para fazer em casa)

- instale a servidora Apache em seu computador;
  - dê uma olhada se o daemon é executado, (`ps -ef | grep apache2`)
  - leia o arquivo de configuração (geralmente algo como `/etc/apache2/apache2.conf`)
  - verifique os caminhos dos diretórios `www` e `cgi-bin`, assim como se chamadas ao `localhost` em seu computador funcionam corretamente

+ Exercício (continuação)

- crie um arquivo TXT, com cada linha contendo um nome de aluno e 2 números (notas)
- faça um HTML com uma caixa de texto (para escrever um nome ou parte dele) e um botão de submissão
- por fim, escreva um script CGI em Perl que receba a solicitação do HTML do ítem acima e imprima uma tabela HTML que contenha o nome, notas e média de todos os alunos cujo nome case com a expressão fornecida.

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*.
   - *CGI Programming with Perl, Second Edition*

4. Verbete "Foo bar" na Wikipédia.
   `http://en.wikipedia.org/wiki/Foo_bar/`.
   Acesso em 17 de junho de 2009.