Ficha 2

Introdução ao Perl

24 de Fevereiro de 2015

1 Listas

Uma variável do tipo lista em começa com o caracter `@`. Assim, se quiséssemos criar uma variável para armazenar os números de 1 a 10 faríamos:

```
@lista = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
@lista = 1..10; # o mesmo que na linha anterior
print "valores: @lista\n";
print "terceiro valor: $lista[2]\n";
print "quarto valor a contar do fim: $lista[-4]\n";
print "segundo, quarto e penultimo valores: @lista[(1, 3, -2)]\n";
```

Também se podem definir listas de *strings*[1]. No caso de definir listas de strings existem uma série de construtores que costumam ser utilizados para escrever menos que se apresentam também.

```
$a="mota";
$b="autocarro";
@palavras = ("barco", 'bicicleta', "$a", '$b'); # Veja a diferenca entre as plicas e aspas
    nas strings
print "@palavras\n";

@a = q {barco bicicleta $a $b};
@c = qw {barco bicicleta $a $b};
@b = qq {barco bicicleta $a $b};
print "@a @b @c\n";
```

É claro que existe outra forma chamada `qx` que pode ser utilizada. No próximo exemplo, o programa imprime o nome de todos os ficheiros na directoria actual que tenham a extensão `.pl`.

```
@todos = qx{ls}; # Se estiver em Windows use dir em vez de ls
@programas = grep {/\.pl$/} @todos; # Daqui a bocado explicaremos o grep; as expressoes
    regulares serao explicadas mais tarde
print "@programas";
```

[1] Na verdade, só se podem fazer listas com escalares e como já sabemos tanto um número como uma string são considerados escalares

2 Iteração sobre listas

A iteração sobre listas usa uma forma especial que se apresenta a seguir:

```
@lista = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
for $x (@lista) {
    print "$x\n";
}
```

Mas pode também efectuar-se duma forma mais tradicional (que, por razões óbvias de simplicidade, não é muito utilizada):

```
lista = (1..10);
# A variavel $#lista contem o valor do ultimo indice da lista (vista como um array)
for($i = 0; $i <= $#lista; $i++) {
    print "$lista[$i]\n";
}
```

3 Funções sobre listas

Há algumas funções sobre listas muito úteis que todos os programadores de Perl devem conhecer[2].

join Junta várias listas numa única string;

split Separa uma string em vários elementos e retorna uma lista com esses elementos;

reverse Inverte uma lista;

grep Selecciona alguns elementos de uma lista;

map Opera uma transformação a todos elementos de uma lista e devolve uma nova lista;

shift Remove o primeiro elemento da lista e retorna-o;

unshift Adiciona elementos à cabeça de uma lista e retorna o novo número de elementos da lista;

push Adiciona elementos à cauda de uma lista e retorna o novo número de elementos da lista;

pop Remove o último elemento da lista e retorna-o;

splice Função muito poderosa de manipulação de listas que pode apagar e inserir elementos ao mesmo tempo, pode-se implementar muitas das outras funções de listas usando esta função;

sort Ordena uma lista.

A figura seguinte mostra exemplos de algumas funções sobre listas e da utilização correspondente da função `splice`.

[2] Lembre-se que pode executar o comando `perldoc` para consultar a documentação de uma função. Por exemplo `perldoc -f split` para consultar a documentação da função `split`

```
push(@a,$x,$y)        splice(@a,@a,0,$x,$y)
pop(@a)               splice(@a,-1)
shift(@a)             splice(@a,0,1)
unshift(@a,$x,$y)     splice(@a,0,0,$x,$y)
$a[$i] = $y           splice(@a,$i,1,$y)
```

As funções `split` e `join` são muito úteis para tratar texto separando-o em vários pedaços e depois tratando a lista correspondente e finalmente juntando os pedaços numa única string. O seguinte exemplo ilustra algumas funções de listas. O que ele faz é ler uma frase e imprimir a mesma frase primeiro ordenada alfabéticamente e depois por tamanhos de palavras. Experimente o programa usando por exemplo com a célebre frase de Hamlet.

```
$frase=<>;
chomp $frase; # Serve para tirar o \n do fim da linha

@palavras=split / /, $frase; # Vamos voltar ao split quando soubermos expressoes regulares
# Ordenado alfabeticamente
@alfa = sort @palavras;
# Ordenado por tamanho
@tam = sort {length $a <=> length $b} @palavras;
# Ordenado primeiro por tamanho e depois alfabeticamente
@tam_alfa = sort {length $a <=> length $b or $a cmp $b} @palavras;
print "@alfa\n";
print "@tam\n";

# Para mostrar o funcionamento do join vamos transformar a ultima lista numa string e
    imprimir a string
$tam_alfa = join " ", @tam_alfa;
print "$tam_alfa\n";
```

As funções `map`, `grep` e `sort` costumam usar um bloco de código associado. No caso da função `sort` como pode reparar usam-se as variáveis `$a` e `$b` dentro do bloco para comparação. Para além disso, surgem alguns comparadores estranhos dos quais ainda não tínhamos falado que serão explicados a seguir. No caso das funções `map` e `grep` o bloco de código usa a variável `$_` para se referir ao elemento actual que o bloco está a examinar. Segue-se um pequeno exemplo, algo insípido que primeiro cria uma lista com os números de 1 a 10 e depois uma lista com os seus quadrados.

```
@num = 1..10;
@quadrados = map {$_ ** 2} @num; # ** e' o operador da potenciacao
print "@quadrados\n";
```

Para ilustrar o funcionamento do `grep` mostra-se um pequeno programa que lê um número e imprime os seus factores (não é uma forma muito eficiente mas é elegante).

```
$num=<>;
@factores=grep {$num % $_ == 0} 1..$num; # % e' o operador do resto da divisao
print "@factores\n";
```

4 Dicionários

Uma variável do tipo dicionário começa com o caracter `%`. Vamos imaginar que queremos armazenar num dicionário o número de utente e o nome do mesmo utente. Poderíamos fazê-lo com o seguinte programa:

```
1 my %utentes = (1002 => "Rui Mendes", 1003 => "Miguel Rocha");
2 my $nome = $utentes{1003};
3 print "$nome\n";
4 $utentes{1004} = "Alberto Simoes"; # Adicionar mais um par chave/valor
```

5 Funções sobre dicionários

Há algumas funções sobre dicionários muito úteis que todos os programadores de Perl devem conhecer.

exists Verifica se uma dada chave existe;

delete Apaga um par chave/valor do dicionário;

keys Retorna a lista de chaves;

values Retorna a lista de valores;

each Itera segundo o par chave/valor;

A figura seguinte mostra exemplos de algumas funções sobre dicionários.

```
1 %ficha = (num => 100, nome => 'Larry Wall', morada => 'Campo dos Camelos', email => '
      perlmasta@perl.org', password => 'IL0v3P3rl');
2 @chaves = keys %ficha;
3 @valores = values %ficha;
4 print "chaves: @chaves\nvalores: @valores\n";
5 while(($chave, $valor) = each %ficha) {
6     print "$chave = $valor\n";
7 }
```

Repare que a função `each` não assume nenhuma ordem (bem, na verdade existe uma ordem interna) para a iteração. Por causa disso, caso se queira iterar pelos pares por, por exemplo, ordem alfabética de chaves não se pode usar esta função.

```
1 %ficha = (num => 100, nome => 'Larry Wall', morada => 'Campo dos Camelos', email => '
      perlmasta@perl.org', password => 'iL0v3P3rl');
2 for $chave (sort keys %ficha) {
3     print "$chave = $ficha{$chave}\n";
4 }
```

À primeira vista pode parecer que a função `exists` não é muito útil. Afinal, se o `use warnings` não estiver ligado, poder-se-ia pensar que as linhas 2 e 3 do próximo programa fossem equivalentes quando óbviamente não o são.

```
$dic{chave} = 0;
print "chave existe" if $dic{chave};
print "chave existe" if exists $dic{chave};
```

É recomendável que se utilize sempre a função `exists` quando se quer verificar se uma dada chave existe num dicionário.

Caso se pretenda apagar uma chave, é necessário utilizar o comando `delete` tal como se mostra a seguir:

```
my %utentes = (1002 => "Rui Mendes", 1003 => "Miguel Rocha");
delete $utentes{1002};
print "chave 1002 nao existe" unless exists $utentes{1002};
```

6 Exercícios

1. Escreva um programa que leia uma frase e imprima a mesma com todas as palavras começadas por uma letra maiúscula;
2. Escreva um programa que imprima todos os números entre 1 e 1000 que não sejam divisíveis por 5 nem por 7
3. Escreva um programa que funcione de forma semelhante ao **wc**, i.e., leia um texto e conte o número de carateres, de palavras e de linhas;
4. Escreva um programa que leia um texto e imprima todas as palavras do texto e o número de vezes que essa palavra aparece no texto;
5. Escreva um programa que conte a frequência de cada letra e que seguidamente imprima a letra e a frequência por ordem decrescente de frequências;
6. Escreva um programa que leia um texto, o separe em palavras e posteriormente calcule a frequência de letras isoladas e conjuntos de duas, três, quatro e cinco letras;
7. Estenda o programa anterior para qualquer número de letras;
8. Escreva um programa que leia uma frase de cada vez e que imprima as palavras que existem nessa frase por ordem alfabética (i.e., sem repetir as palavras);
9. Escreva um programa que seja um filtro que remove linhas duplicadas de um conjunto de ficheiros (i.e., ele imprime todas as linhas dos ficheiros sem duplicados);
10. Escreva um programa que leia um texto e remova os artigos e preposições (i.e., imprima todas as palavras do texto que não pertençam a nenhuma dessas classes);
11. Pretende-se que escreva um programa que leia um texto e conte co-ocorrências de palavras em frases (para conveniência utilize primeiro o programa definido na alínea anterior que remove artigos e preposições).
12. Faça um programa que leia um texto e crie um índice remissivo. Por exemplo, com o seguinte texto:

Texto

```
Tis but thy name that is my enemy;
Thou art thyself, though not a Montague.
What's Montague? it is nor hand, nor foot,
Nor arm, nor face, nor any other part
Belonging to a man. O, be some other name!
What's in a name? that which we call a rose
By any other name would smell as sweet;
So Romeo would, were he not Romeo call'd,
Retain that dear perfection which he owes
Without that title. Romeo, doff thy name,
And for that name which is no part of thee
Take all myself.
```

O seu programa deveria imprimir o seguinte resultado:

Resultado

```
a 2 5 6
all 12
and 11
any 4 7
arm 4
art 2
as 7
be 5
belonging 5
but 1
by 7
call 6
call'd 8
dear 9
doff 10
enemy 1
face 4
foot 3
for 11
hand 3
he 8 9
in 6
is 1 3 11
it 3
man 5
montague 2 3
my 1
myself 12
```

```
name 1 5 6 7 10 11
no 11
nor 3 4
not 2 8
o 5
of 11
other 4 5 7
owes 9
part 4 11
perfection 9
retain 9
romeo 8 10
rose 6
smell 7
so 8
some 5
sweet 7
take 12
that 1 6 9 10 11
thee 11
thou 2
though 2
thy 1 10
thyself 2
tis 1
title 10
to 5
we 6
were 8
what's 3 6
which 6 9 11
without 10
would 7 8
```

13. Pretende-se que implemente o comando `juncao` em Perl. Este comando deverá efetuar a junção entre as duas tabelas usando a *junção natural* assumindo que os vários campos das duas tabelas estão separados por *whitespace*. O comando terá duas formas de funcionamento conforme o nº de argumentos:

    2 Neste caso, os dois argumentos são dois ficheiros e a junção será pelo primeiro campo de ambos os ficheiros;

    4 Neste caso, os dois primeiros argumentos designam o nº do campo que é utilizado para cada um dos ficheiros enquanto que os restantes dois continuam a identificar os dois ficheiros que contém as tabelas. Segue-se um exemplo de utilização.

```
rui@omega:~/EL/exerc$ cat tab1
rcm     12
mpr     14
jja     18
prh     17
jcr     16
rui@omega:~/EL/exerc$ cat tab2
jcr     azerty  19
rcm     qwerty  14
prh     azerty  14
jja     querty  17
rui@omega:~/EL/exerc$ perl juncao.pl tab1 tab2
rcm     12 rcm  qwerty  14
jja     18 jja  querty  17
prh     17 prh  azerty  14
jcr     16 jcr  azerty  19
rui@omega:~/EL/exerc$ perl  juncao.pl 1 2  tab1 tab2
mpr     14 rcm  qwerty  14
mpr     14 prh  azerty  14
prh     17 jja  querty  17
```