9. Captura da saída produzida pela execução de programas externos

Em PERL, uma das formas de se executar um programa externo e capturar a saída produzido por esse programa é recorrendo-se ao operador de "plicas invertidas", à semelhança da substituição de comandos da *shell*. Para tal, indica-se o programa externo (muitas vezes um comando da shell, que pode conter inclusivamente operadores especiais da *shell*, como *pipes* "|" e redireccionamentos ">" e ">>") com as eventuais opções da linha de comando, tudo isso, delimitado pelo operador plica invertida. Atenda-se ao seguinte exemplo, que executa o comando "ls" com as opções "-l –a" (produzindo portanto uma listagem em formato longo de todos os ficheiros do diretório corrente).

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

@SaidaProg_L = `ls -l -a`;
print  "\@SaidaProg_L = @SaidaProg_L\n";
```

Listagem 20: captura da saída produzida pela execução de um programa externo

NOTA: a temática da execução de programas externos é discutida mais adiante nesta ficha.

9.1. Execução de um programa externo num *script*

Em PERL, para executar um programa externo (por exemplo, um comando da BASH) pode recorrer-se à função `open`, especificando como modo de abertura o programa externo seguido do `pipe`. Concretamente, tem-se:

1. Criar uma variável com o comando a executar
   - **$cmd = "ls";**
2. Redireccionar a saída do comando para um ficheiro acedido pela variável IN_F (nome arbitrário)
   - **open (IN_F, "$cmd |");**
3. Guardar a saída do comando numa variável
   - **@lista = <IN_F>;**

```
#!/usr/bin/perl -w
print "Entre o comando a executar:";
$cmd = <STDIN>;
open(IN_F, "$cmd |") || die ("Ocorreu um erro.\n");
@user = <IN_F>;
print (“@user”);
```

Listagem 21: Execução de um programa externo num *script* PERL

```
#!/usr/bin/perl -w
#-----------------------------------------------------------------
# Programa que envia por correio electrónico
# ao utilizador $Utente a listagem corrente dos processos
# ordenados pela quantidade de memória empregue
# Equivale a seguinte linha de comando:
# ps aux  | sort -n -r +4 | mail -s "ps" Utente
#-----------------------------------------------------------------

# substituir pelo endereço electrónico pretendido
$Utente = "xpto@maquina.pt";

# Envio de um mail
open( MAIL, "| mail -s 'ps' $Utente") || die("MAIL: $!");
open( TMP, "ps aux| sort -n -r +4|") || die("TMP: $!");

@Tmp = <TMP>;
print MAIL @Tmp;

close ( MAIL );
close( TMP );
```

Listagem 22: Executar vários comandos num *script*

9.2. Operador de Substituição de Comando em PERL (plicas invertidas (``))

Como já anteriormente foi notado, e à semelhança do que sucede com a linguagem BASH, no PERL, o operador de substituição de comando (**plicas invertidas (``)**) permite a atribuição do resultado de um comando a uma variável. Deste modo, é

possível aceder aos resultados produzidos, por determinada sequência de comandos. Considere o exemplo seguinte:

```
#!/usr/bin/perl -w
#-----------------------------------------------------------------
# Programa que constrói o endereço de correio electrónico de todos
# os utilizadores registados na máquina
# Recorre ao operador de "substituição de comando"
# Patrício, 14-04-2000
#-----------------------------------------------------------------

# Variável Maquina recebe o nome completo do computador (e.g. sounix)
$Maquina = `hostname -f`;
$Data = `date +%d-%m%Y`;   # Recolhe data do sistema

# Array de utilizadores recebe o nome de todos os utilizadores do
# sistema
@utilizadores = `cut -f1 -d: /etc/passwd | sort`;
$i = 1;

chomp(@utilizadores);
foreach $User ( @utilizadores )
{
  print ("$i $User\@$Maquina");
  $i++;
}

# Imprime data
print ("Data: $Data\n");
```

Listagem 23: Operador "plicas invertidas"

10. Vetores (*arrays*)

Até agora lidamos com elementos ESCALARES, isto é, com um só valor. O PERL suporta também o conceito de array, que representa um vetor de elementos escalares (isto é, um array é formado por vários escalares).

O símbolo "@" identifica uma variável como sendo um array.

```
#!/usr/bin/perl –w

$a = "120";                  # "variável a" na forma de escalar
@a = ('1', '2', '3', '4');     # o array a
@ num = (1, 2, 3, 4, 5);       # Array de números
@ num = (1 .. 5);            # Array especificado através de uma gama de números

# Array de strings
@strings = ('um','dois','três','quatro','cinco');

# Alternativa no array de strings: qw (quoted word)
@strings = qw( um dois três quatro cinco);

@misto = (3.1415, 'zero',20);              # Array misto
@misto2 = (@num, @strings);                # outro array misto

@list = (7,3,4,12,'a',"Um exemplo de um elemento de uma 'lista' ",8,'c');

$list[2] = 5;          #altera o 3º elemento de @list

#@list2 é uma copia da sub-lista contida em @list entre a posição 1 e 4
@list2 = @list[1..4];

@list3 = (1..20);                  #$list3 é criada com os elementos de 1 a 20
@list3[3,4,5] = (23, 33, 43); #altera os valores do 4º,5º e 6º elemento de @list3

@list4 = @list[0, 3, 6];  #@list4 é uma cópia da sub-lista contida em @list composta
                          #pelos 1º,4º e 7º elementos

print '@list =',"@list \n";
print '$list[5]=',"$list[5] \n";
print '@list3 =',"@list3 \n";
print '$#list=',"$#list \n";       #a variável $#list contém o último índice da lista @list
```

Listagem 24: Arrays em PERL

10.1. Acesso aos elementos de um array

O acesso a um elemento de um array processa-se através da seguinte sintaxe:

$Nome_do_array[índice], em que **índice** representa a posição do elemento pretendido (tal como no C, o primeiro elemento tem índice 0). O PERL, como linguagem flexível que é, admite índices negativos. Por exemplo, o índice "-1" refere-se ao último elemento do array (i.e. o mesmo que $array[$#array]).

```
@Pares = (2, 4, 6, 8, 10);
print (" $Pares[0]\n");  # imprime primeiro elemento do array
print ("$Pares[10]\n"); # acesso a um valor indefinido do array
```

Listagem 25: Exemplo que usa arrays

10.1.1. Acesso ao último elemento de um array

O número de elementos de um array consegue-se através de $#Nome_do_array acrescido em uma unidade

```
@vector = (1, 3, 5, 7, 9);
$j = 0;
while ($j <= $#vector ) #índice do último elemento
{
  print("Vector: $vector[$j]\n");
  $j++;
}
```

Listagem 26: Obter o número de elementos do array

Nota – uma alternativa mais expedita de conseguir o mesmo do que o ilustrado, no exemplo anterior seria:

```
foreach $elm (@vector )
{
  print ("$elm\n");
}
# Nota – outra forma ainda mais simples para mostrar o array, seria:
print ("@vector\n"); # imprime todos os elementos separado por espaço
```

Listagem 27: Percorrer array sem obter o número de elementos

10.1.2. Contextos – escalares e vetoriais

No PERL, consoante o contexto em que se insere, uma variável do tipo array, pode assumir um valor escalar, ou um valor vetorial.

```
@vetor_1 = (0, 2, 4, 5, 8);
@vetor_2 = @vetor_1;      # cópia do vetor_1 (contexto vectorial)
$NumElementos = @vetor_1;   # Nº de elementos do array (contexto escalar)
```

Listagem 28: Cópia de arrays

O PERL disponibiliza a função *scalar* para a criação de um contexto escalar em situações em que o mesmo não é necessariamente explícito.

Exemplo:

```
# mostra o número de elementos de um vector (contexto escalar)
@vogais = qw( a e i o u);
print (“Número de vogais: scalar(@vogais)\n”);
```

Listagem 29: O contexto escalar de um array

10.1.3. Ordenação de arrays

A ordenação dos elementos de um array pode ser conseguida através da função “sort”. De lembrar que a referida função ordena em função dos códigos da tabela ASCII.

```
# Vector de strings
@Strings = (”Paris”, ”Lisboa”, ”Madrid”, ”Barcelona”, ”Leiria”, ”Porto”);
@Strings_ordenadas = sort @Strings;
print ”@Strings_ordenadas”;
```

Listagem 30: Ordenação de vectores (*arrays*)

A ordenação acima apresentada funciona corretamente com strings. Contudo, para a ordenação de números torna-se necessário acrescentar uma função (anónima) de ordenação, função que deve comparar dois elementos individuais (qualquer algoritmo de ordenação se reduz, na sua essência, a comparar dois elementos). No PERL, a função é indicada da seguinte forma

```
  sort { $a <=> $b }
```

A listagem seguinte exemplifica a ordenação de um vector de número, recorrendo-se ao sort do PERL.

Exemplo:

```
# Mostra o vector numérico @Nums_L ordenado
my @Nums_L = (1974, 1, 2, 27.12, 26.0, 567, 23);
my @Nums_Sorted_L = sort{ $a <=> $b } @Nums_L;
print("\@Nums_L=@Nums_L\n");
print("\@Nums_Sorted_L=@Nums_Sorted_L\n");
sort { $a <=> $b }
```

Listagem 31: Ordenação de vectores (*arrays*) numéricos

10.1.4. Iterando os elementos de um vetor através do foreach

Dado em PERL um vetor se tratar de uma lista, é possível iterar os elementos de um vetor, através da estrutura de iteração foreach. Por exemplo, considere o seguinte código:

```
#!/usr/bin/perl -w
# Código que itera pelos 10 elementos do vector ″@Numeros_L″
@Numeros_L = (1..10);
foreach $Num (@Numeros_L)
{
   print ″$Num=’$Num’\n″;
}
```

Listagem 32: Iteração do conteúdo de um vector através do foreach

10.1.5. Outras funções úteis para utilização de listas

push LIST, VALUES → Copia os valores VALUES para o fim da lista LIST
pop LIST → Remove da lista LIST o último elemento
shift LIST → Retira o 1º elemento da LIST; Em caso de omissão da LIST usa o@ARGV ou o @_
unshift LIST, VALUES → Copia os valores VALUES para o início da LIST; Em casode omissão da lista LIST usa o @ARGV ou o @_
join EXPR, LIST → Faz a junção dos elementos da LIST numa string usando a EXPR para unir os elementos

```
#!/usr/bin/perl
@list = (1,5,3,9,7);
@l =('a','bbbbbbbbbbbb');
push @list,@l;       #insere os elementos de @l no final da lista @list
push @list,33;

#insere o elemento 44, seguido do 55 no final da lista
push @list,44,55;
#remove o elemento 1 da lista e imprime-o
print "2:\t",shift @list,"\n";
unshift @list,100;   #insere o elemento 100 no inicio da lista
$i = 200;
unshift @list,$i;

#imprime cada elemento da lista numa nova linha
print "----\n",join("\n",@list),"\n";

#$res é uma string composta pelos elementos de @list separados por '+'
$res = join("+",@list);
print $res,"\n";
```

split PATTERN, EXPR → Separa o valor de EXPR de acordo com o PATTERN

```
#!/usr/bin/perl
$s ="Isto é uma cadeia de caracteres separados por espaços";
@r = split(/ /,$s);          #@r é uma lista das palavras da frase acima <= $s foi separada por espaços
print "1:\t",$r[0],"\n";
print "2:\t",$r[$#r],"\n";
$s ="Isto é uma frase que, por motivos de demonstração, está separada por vírgulas";
@r = split(/,/,$s); #$s é separada pelas vírgulas
print "3:\t",$r[0],"\n"; print "4:\t",$r[$#r],"\n";
```

reverse LIST → Inverte a LIST

```
#!/usr/bin/perl
@list = (1,5,3,9,7);
print join(";",@list),"\n";
@tsil = reverse @list;
print join(":",@tsil),"\n";
```

Exercício 10

a) Escreva um programa que peça cinco palavras ao utilizador e que as mostre de forma ordenada na saída padrão (ecrã).

Nota: Use a função *push(array,elemento(s))* para adicionar elementos ao array.

b) Recorrendo ao operador "*plica invertida*" elabore a script "utilizadores.pl" que deve mostrar por ordem alfabética, os logins de todos os utilizadores de um sistema Linux.

Sugestão: para cada login existe uma linha no ficheiro das "passwords", i.e. o /etc/passwd.