11. Listas Associativas (*hashes*)

Para além dos arrays, O PERL providencia uma segunda forma de agrupar elementos: as listas associativas (*hashes* na literatura Anglo-Saxónica).

Uma lista associativa é uma lista não ordenada de chaves e elementos, servindo a chave (que é normalmente do tipo string) para aceder a um elemento. Assim, ao contrário dos vetores, no qual o acesso se processa através de um índice, nas listas associativas o acesso processa-se através da chave que está associada ao elemento com a seguinte sintaxe **$Nome_da_Lista{chave}**.

Em PERL, as listas associativas são identificadas pelo carácter %, logo no início do identificador.

```
# hash com os meses (chaves) e o número de dias (elementos)
# Por uma questão de clareza as chaves encontram-se sublinhadas
%NumDiasMeses = (‘Janeiro’, 31, ’Fevereiro’, 28, ’Março’, 31);

# Acesso ao nº de dias do mês de Janeiro
print $NumDiasMeses{‘Janeiro’};
```

Listagem 33: Listas associativas para meses

11.1. Adicionar elemento à lista

```
# Código postais
%CodigoPostal = (‘2400’,’Leiria’,’3000’,’Coimbra’,’1000’,’Lisboa’);
print “O código postal 2400 é de : $CodigoPostal{‘2400’}\n”;


# Acréscimo de mais um elemento ao HASH
$CodigoPostal{‘2000’} = ’Porto’;
print “O código postal 2000 é de : $CodigoPostal{‘2000’}\n”;
```

Listagem 34: Listas associativas para códigos postais

```
#É possivel usar '=>' em vez da vírgula para separar os elementos de uma lista
%natural = ('Ana Cristina' => 'Braga', 'Jose Manuel' => 'Viana', 'Antonio Joaquim' => 'Aveiro');

print "A Naturalidade de Ana Cristina e: $natural{'Ana Cristina'}\n";

@a = keys %natural;
print "@a \n";

#Para alterar o valor de duas ou mais chaves temos de trabalhar em contexto de lista
$natural {'Ana Cristina'} = 'Aveiro';
```

```
#Para uma lista com os valores de uma lista Associativa poderemos fazer
@y = @natural{'Ana Cristina','Jose Manuel'}; #@y é uma lista composta pela naturalidade da 'Ana Cristina' e do 'Jose Manuel'
print "Y: @y\n";

#outra forma
@a = keys %natural; #Lista de todas as chaves
print "@a \n";
@a = values %natural; #Lista de todos os valores
print "@a \n";
```

11.2. Apagar um elemento de uma Lista Associativa

A eliminação de um elemento de uma lista associativa é conseguida através do operador *delete*.

Exemplo:

```
delete $ListaAssociativa{‘chave’};
```

```
#!/usr/bin/perl
%A = ( 7 => 'aaaaa', 'a' => 23, 12 => 123, 'b' => 'sssssssss');
@A = (1, 5, 3, 6, 4, 2, 1);    #NOTA: @A é uma variável distinta de %A

@b = keys %A; print "@b \n";

delete $A{'a'};
#os pares (7, 'aaaaa') e (12,123) de %A são removidos
delete @A{'7', '12'};

@b = keys %A;
print "@b \n";


delete @A[1..4];             #os elementos 5,3,6,4 de @A são removidos
print "@A \n";
```

11.3. Iterar uma Lista Associativa

A iteração de uma lista associativa (i.e., percorrer todos os seus elementos) requer o uso da função `keys` que devolve um array com as chaves da lista referenciada.

```
%CodigoPostal = (‘2400’, ’Leiria’, ’3000’,’Coimbra’, ’1000’,’Lisboa’);
foreach $Codigo (sort keys %CodigoPostal)
{
  print(“$Codigo – $CodigoPostal{$Codigo}\n”);
}
```

Listagem 35: Iterar uma lista associativa

Por sua vez, a função `values`, quando aplicada a uma lista associativa, devolve os elementos da lista (i.e. tudo com excepção das chaves).

```
%CodigoPostal = (‘2400’,’Leiria’, ’3000’,’Coimbra’, ’1000’,’Lisboa’);
# Constrói array apenas com os elementos da lista associativa
@Elementos = values( %CodigoPostal );
```

Listagem 36: elementos da lista associativa

11.4. Extração de dados para Arrays ou Listas Associativas

Quando se procede à leitura de dados a partir de ficheiros de texto, os vários elementos de entrada encontram-se possivelmente numa linha, quando normalmente o requerido é o acesso individual a cada palavra. Para tal, o PERL apresenta a função `split`.

```
# Simulamos uma linha de um ficheiro com uma variável escalar string
$StringdeNumeros = '12 24 34 56 78 23 21';
# Especificamos o espaço (' ') como separador
@Palavras = split(' ', $ StringdeNumeros);
```

Listagem 37: Uso da função split

```
Função split
- Como aceder a uma palavra no PERL ($Linha contém várias palavras)?
# o array @palavras é constituído pelas palavras de $Linha
@palavras = split (' ', $Linha); # palavras recebe as palavras da
$linha
- Como aceder aos caracteres de uma palavra no PERL ($Palavra contém
uma palavra)?
# o array @caracteres é constituído pelos caracteres de $Palavra
@caracteres = split (//, $Palavra);
```

Listagem 38: Mais exemplos de uso da função split

11.5. Outras funções úteis para trabalho com arrays ou listas associativas

each %HASH → Retorna o próximo par chave=>valor ou a próxima chave, dependendo do contexto
exists $HASH{$key} → Testa se uma determinada chave existe na HashTable

```
#!/usr/bin/perl
%A = ( 7 => 'aaaaa', 'a' => 23, 12 => 123, 'b' => 'sssssssss');
@p = each %A;                  #@p fica com o par (7,'aaaaa')
@k = each %A;                  #@k fica com o par ('a',23)
print "1:\t$p[0] $p[1]\n";
print "2:\t$k\n";
@p = each %A;                  #@p agora assume o valor (12,123)
@k = each %A;
print "3:\t$p[0] $p[1]\n";
print "4:\t$k\n";
```

```
#!/usr/bin/perl

%A = ( 7 => 'aaaaa', 'a' => 23, 12 => 123, 'b' => 'sssssssss');
print "12 é uma chave do array e o seu valor é: $A{12}\n" if exists

$A{12};
print "'12' é uma chave do array e o seu valor é: $A{'12'}\n" if
exists $A{'12'};

$i = 12;
print "$i é uma chave do array e o seu valor é: $A{$i}\n" if exists
$A{$i};

$i = 'aa';
print "$i não é uma chave do array\n" unless exists $A{$i};

$i = 'a';
print "$i não é uma chave do array\n" unless exists $A{$i};
```

Exercício 11

a) Escreva um programa "listaeconta.pl" que peça um ficheiro ao utilizador e devolva a lista de palavras encontradas, e quantas vezes se repete cada uma delas.

b) Escreva o programa “traduz.pl” que procede à tradução do conteúdo de um ficheiro com palavras em Português para Inglês. O nome dos ficheiros deve ser indicado através dos argumentos da linha de comando, sendo que oprimeiro argumento corresponde ao nome do ficheiro contendo o dicionário e o segundo argumento deve conter o nome do ficheiro a ser traduzido. O programa deve mostrar na saída padrão o ficheiro traduzido, sendo que nos casos em que não exista no dicionário uma palavra do ficheiro a traduzir, deve-se manter a palavra não traduzida.

O ficheiro dicionário deverá ter o seguinte formato:

```
rua-street
casa-home
sol-sun
frio-cold
chuva-rain
```

12. Funções

Para além, das muitas funções definidas pelo próprio PERL (e.g. chomp, print, printf, grep, etc.), é possível em PERL o utilizador definir as suas próprias funções.

12.1. Definição de funções

Em PERL, uma função do utilizador é definida com recurso à seguinte sintaxe:

```
sub Func
{
  expressão_1;
  expressão_2;
(...)
  expressão_N;
}
```

Listagem 39: Funções em PERL

De notar que Func representa o nome da função.

12.1.1. Visibilidade das funções

Em PERL as funções são "globais", pelo que se forem definidas duas funções com o mesmo nome, a segunda definição substitui-se à primeira sem que o programador seja notificado, exceto se estiver a executar o *script* com o modo de *warning* ativo (opção –w).

12.2. Chamada a funções

No PERL uma função é chamada especificando-se o seu nome, seguido de parêntesis que podem conter os eventuais parâmetros.

12.3. Parâmetros

Na chamada a uma função, é possível especificar parâmetros a serem passados à função, especificando-se esses mesmos parâmetros entre os parênteses. Dentro do código da função, o acesso aos parâmetros processa-se através de uma array especial: "@_"que contém todos os parâmetros que foram passados na chamada à função. Dado que, não

existe obrigatoriedade na definição de protótipos em PERL, as funções podem receber números variáveis de argumentos.

A listagem seguinte ilustra o conceito de passagem de parâmetros, calculando a soma de dois números.

```
sub Soma_2
{
   my $Soma;

   # Certifica-se que apenas dois parâmetros foram especificados
    # na chamada à função
   if ( @_ != 2 )
   {
   print("A função recebe dois e só dois parâmetros\n");
   return;
   }
   # Soma os dois parâmetros
   return( $_[0] + $_[1] );
}
```

Listagem 40: Passagem de parâmetros

A função "Soma_2", com a atribuição dos parâmetros a variáveis locais.

```
sub Soma_2
{
   my $Soma;
   my $Parcela_1,$Parcela_2;
   # Leitura do vector de parâmetros
   ($Parcela_1,$Parcela_2) = @_;
   # Soma os dois parâmetros
   return($Parcela_1 + $Parcela_2 );
}
```

Listagem 41: Variáveis locais para recolher os parâmetros

Por sua vez, a função Soma_N, generaliza a função anterior (Soma_2), permitindo a soma de N números.

```
sub Soma_N
{
   $NumParcelas = @_;
   print("Soma de $NumParcelas parcelas\n");
   $Soma = 0;
   foreach $Num(@_)
   {
      $Soma += $Num;
   }
   return( $Soma );
}
```

Listagem 42: Número de parâmetros

A função "Soma_N" pode então ser chamada das seguintes formas:

```
Soma_N(1);
print( “A soma = “, Soma_N(1,200,300,1), “\n”);
Soma_N(1,200,300,1,-12,29);
print (“Soma de 1+2+3+4+5 = ”, Soma_N(1..5), “\n”);
```

Listagem 43: Chamar funções

12.3.1. @_

O vetor de parâmetros “@_” é privado à função, pelo que qualquer modificação nesse vetor (i.e. nos parâmetros recebidos) não é visível no espaço exterior à função.

12.4. Acesso a parâmetros inexistentes

Quando se acede a parâmetros inexistentes (e.g. aceder a $_[3], quando só se tem um parâmetro), o valor obtido é indefinido (**undef**) , sendo que o PERL não emite nenhum erro, apenas um aviso, caso esses se encontrem ativados (opção -w).

No caso, de não serem empregues pela função parâmetros especificados na chamada à dita função, também nada é assinalado.

```
# Chamada à função “Soma_2” apenas com um parâmetro
# Ao parâmetro indefinido (i.e. a 2ª parcela que falta) é atribuído
# o valor zero.
print( “Soma = “, Soma_2(1),”\n”);
```

Listagem 44: Parâmetros inexistentes

12.5. Valores de retorno

O valor de retorno de uma função é o especificado pela palavra reservada **return**. Podem ser especificados vários valores de retorno. Atenda-se ao exemplo seguinte:

```
#/bin/perl -w
# Exemplo de uma função que devolve uma lista de valores
sub MultiReturn()
{
        $A = 100;
        $B = "string 200";
        return $A,$B;
}
```

```
sub Main()
{
        # Recebe lista de valores de “MultiReturn”
        ($Ret01,$Ret02) = MultiReturn();
        print("\$Ret01=$Ret01\n");
        print("\$Ret02=$Ret02\n");
}
# Chama função “main”
Main();
```

Listagem 45: Função a devolver lista de valores

13. Algumas funções de PERL

chdir DIRETORIO	Altera o diretório de trabalho para **DIRETORIO**. **Exemplo:** (mudar para a diretoria ~/eiso101/public_html e executar o comando ls) **$dir = ‘~/utilizador/public_html’;** **chdir $dir;** **print `ls`;**
mkdir NomeFicheiro	Cria uma diretoria com o nome **NomeFicheiro.** **Exemplo:** (criar a diretoria apagar) **mkdir apagar;**
rmdir NomeFicheiro;	apaga uma diretoria com o nome **NomeFicheiro.** **Exemplo:** (apagar a diretoria apagar) **rmdir apagar;**
opendir (HANDLE, NomeDirectoria)	abre uma diretoria acessível pela HANDLE.
readdir(HANDLE)	Lê a informação da diretoria
closedir (HANDLE)	fecha o acesso a diretoria **Exemplo:** listar o conteúdo da diretoria ~/eiso101/public_html **opendir(FILE, ‘~/eiso101/public_html’);** **print readdir(FILE);**

	closedir(FILE);

Nota: Todas estas funções e muitas outras podem ser encontradas no manual do PERL ou no endereço http://www.Perldoc.com/perl5.6/pod/perlfunc.html.

13.1. Outras funções de PERL

Outras funções úteis para diversas utilizações.

Funções para escalares ou strings chomp, chop, chr, crypt, hex, index, lc, lcfirst, length, oct, ord, pack, q/STRING/, qq/STRING/, reverse, rindex, printf, substr, tr///, uc, ucfirst, y///
Expressões regulares e verificação de padrões. m//, pos, quotemeta, s///, split, study, qr//
Funções Numéricas abs, atan2, cos, exp, hex, int, log, oct, rand, sin, sqrt, srand
Funções para manuseamento de listas pop, push, shift, splice, unshift , join, map, reverse, sort,
Funções para listar dados grep, qw/STRING/, unpack, pack
Funções para manusear arrays associativos delete, each, exists, keys, values
Funções de entrada e saída de dados binmode, close, closedir, dbmclose, dbmopen, die, eof, fileno, flock, format, getc, print, printf, read, readdir, rewinddir, seek, seekdir, select, syscall, sysread, sysseek, syswrite, tell, telldir, truncate, warn, write
Funções para manipular ficheiros ou diretórios operador -X, chdir, chmod, chown,chroot, fcntl, glob, ioctl, link, lstat, mkdir, open, opendir, readlink, rename, rmdir, stat,symlink, umask, unlink, utime
***Keywords* para controlo de fluxo** caller, continue, die, do, dump, eval, exit, goto, last,next, redo, return, sub, wantarray
***Keywords* relacionadas com o contexto do código** caller, import, local, my, our,package, use
Funções diversas defined, dump, eval, formline, local, my, our, reset, scalar, undef, wantarray
Funções para processos e grupos de processos alarm, exec, fork, getpgrp, getppid, getpriority, kill, pipe, qx/STRING/, setpgrp, setpriority, sleep, system, times, wait,

waitpid
***Keywords* relacionadas com módulos e blocos** do, import, no, package, require, use
Funções para comunicação utilizando sockets accept, bind, connect, getpeername, getsockname, getsockopt, listen, recv, send, setsockopt, shutdown, socket, socketpair
Funções para comunicação de processos System V msgctl, msgget, msgrcv, msgsnd, semctl, semget, semop, shmctl, shmget, shmread, shmwrite
Funções para obtenção de informação sobre o utilizador, o grupo e o ambiente. endgrent, endhostent, endnetent, endpwent, getgrent, getgrgid, getgrnam, getlogin, getpwent, getpwnam, getpwuid, setgrent, setpwent
Funções para obtenção de informação sobre a rede. endprotoent, endservent, gethostbyaddr, gethostbyname, gethostent, getnetbyaddr, getnetbyname, getnetent, getprotobyname, getprotobynumber, getprotoent, getservbyname, getservbyport, getservent, sethostent, setnetent, setprotoent, setservent
Funções relacionadas com o relogio gmtime, localtime, time, times
Novas Funções na versão 5 do Perl abs, bless, chomp, chr, exists, formline, glob, import, lc, lcfirst, map, my, no, our, prototype, qx, qw, readline, readpipe, ref, sub, sysopen, tie, tied, uc, ucfirst, untie, use

Exercício 12

Implemente uma função que determine se o número passado à mesma é par ou ímpar. Faça também um programa para testar a referida função.

14. Manipulação de ficheiros

À semelhança de outras linguagens de programação, no PERL, a utilização de um ficheiro para leitura ou escrita requer a abertura do ficheiro.

14.1. Abrir um ficheiro (Função open)

Para abrir um ficheiro a biblioteca de funções do PERL disponibiliza a função **open**.

Sintaxe: **open (FileVar, FileName);**

FileVar: nome que a utilizar no programa **PERL** para identificar o ficheiro (*file handle)*. Para esta variável é aconselhada a utilização de nomes escritos em maiúsculas, pretendendo-se com este procedimento tornar mais fácil a distinção das variáveis tipo de ficheiro das restantes variáveis do programa.

FileName: Nome do ficheiro que se pretende abrir. Se pretendermos abrir um ficheiro que não esteja na diretoria do programa, é necessário indicar o caminho completo do ficheiro.

Exemplos:

```
open(FICHEIRO, “ficheiro.txt”);
open(FICHEIRO, “/local/home/ficheiro.txt”);
```

14.2. Modo de acesso a ficheiros

Quando se abre um ficheiro é necessário indicar o modo de acesso ao ficheiro. Existem três modos de acesso a ficheiros:

Modo	Descrição
Leitura	Apenas permite a leitura do conteúdo do ficheiro
Escrita	Elimina o conteúdo do ficheiro e escreve por cima
Acrescentar	Acrescenta ao conteúdo do ficheiro

Tabela 7: Modos de abertura de ficheiros

Utilização da função “open” no modo:

leitura: open(FICHEIRO, “ficheiro.txt”);

Escrita: open(FICHEIRO, “>ficheiro.txt”);

Acrescentar: open(FICHEIRO, “>>ficheiro.txt”);

Nota: Não deve esquecer que ao abrir um ficheiro:

- O conteúdo do ficheiro é destruído se for aberto no modo escrita;
- Não é possível ler e escrever para o mesmo ficheiro em simultâneo;

14.3. Verificação do sucesso da operação "open"

Antes de se utilizar o descritor iniciado pela função *open*, deve-se verificar se o ficheiro foi corretamente aberto. Para esse efeito, a função *open* devolve um valor diferente de zero ("verdadeiro" na interpretação do PERL) se a operação tiver sido bem sucedida, e zero ("falso"), caso não tenha sido possível abrir o ficheiro (por exemplo, o ficheiro não existe, ou o utilizador não tem permissões para o mesmo, etc.).

O exemplo seguinte ilustra uma forma de testar o valor de retorno da função *open*, terminando a execução caso a operação de abertura falhe. Concretamente, a estrutura de controlo *unless* é utilizada para verificar se o ficheiro foi aberto com sucesso.
A função *die* permite enviar uma mensagem ao utilizador e termina o programa, sendo que a variável do sistema **$!** devolve a mensagem do sistemas descritiva do erro. Finalmente, a função *close* fecha o ficheiro que estava aberto.

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

unless (open(FICHEIRO, "dados.txt"))
{
    die ("Ocorreu um erro a abrir o ficheiro: $! \n");
}
print ("O ficheiro foi aberto com sucesso!\n");
close (FICHEIRO);
```

Listagem 46: *open* e *close*

O exemplo seguinte é equivalente ao anterior, sendo se empregou o operador lógico "||" em lugar do operado *unless*.

Exercício 13

Explique como funciona a solução apresentada com "||".

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

open(FICHEIRO, "dados.txt") || die ("Erro ao abrir o ficheiro: $! \n");
print ("O ficheiro foi aberto com sucesso!\n");
close (FICHEIRO);
```

14.4. Ler o conteúdo de um ficheiro linha a linha

A forma de ler os dados de um ficheiro é idêntica à forma de leitura de dados do STDIN (não é de admirar, dado que o "STDIN" é ele próprio um descritor de ficheiro, neste caso, associado à entrada padrão).

```
$linha = <FICHEIRO>;
```

```
#!/usr/bin/perl -w
open(FICHEIRO, "dados.txt") || die ("Ocorreu um erro a abrir o ficheiro:
$!\n");

print ("O ficheiro foi aberto com sucesso!\n");

$linha = <FICHEIRO>;
while ($linha ne "")
{
   print ($linha);
   $linha = <FICHEIRO>;
}
close (FICHEIRO);
```

Listagem 47: Ler linhas de um ficheiro

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

#-------------------------------------------------
# Programa que exibe o ficheiro "dados.txt"
# linha a linha, acrescido de um numero de linhas
# Patricio, 14-04-2000
#-------------------------------------------------

use strict;
my $i;
my $Linha;

open(FICHEIRO, "dados.txt") || die ("Ocorreu um erro a abrir o ficheiro: $!\n");

print ("O ficheiro foi aberto com sucesso!\n");

$i = 1;
foreach $Linha ( <FICHEIRO> )
{
        print ( "linha $i - $Linha");
        $i++;
}
close (FICHEIRO);
```

Listagem 48: Programa que lê o conteúdo de um ficheiro linha a linha e as apresenta no ecrã

14.5. Ler todo o conteúdo de um ficheiro para uma variável array (@array = <FICHEIRO>;)

```
#!/usr/bin/perl -w
#---------------------------------------------------------------------
# Programa que lê todo o conteúdo de um ficheiro para um array, sendo
# que cada elemento do array representa uma linha
#---------------------------------------------------------------------
open(FICHEIRO, "dados.txt") || die ("Ocorreu um erro a abrir o ficheiro!\n";
print ("O ficheiro foi aberto com sucesso!\n");
@array = <FICHEIRO>;
foreach $linha (@array)
{
   print ($linha);
}

close (FICHEIRO);
```

Listagem 49: Leitura completa de um ficheiro para um array

14.6. Escrita para ficheiro

A escrita para um ficheiro requer a abertura do ficheiro no modo escrita ou de acréscimo. Para se escrever no ficheiro pode-se utilizar a função **print** da forma indicada na listagem seguinte.

```
open (FICHEIRO, ">saida.txt");
# Atenção ao parêntesis
print FICHEIRO ("Mensagem enviada para o ficheiro.\n");
```

Listagem 50: Função *print* aplicada a ficheiros

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

open(F_DADOS, "dados.txt") || die ("Ocorreu um erro a abrir o ficheiro!\n";
open(F_COPIA, ">copiaDados.txt") || die ("Ocorreu um erro a abrir o
ficheiro!\n";

$linha = <F_DADOS>;
while ( $linha ne "")
{
   print F_COPIA ($linha);
   $linha = <F_DADOS>;
}
close (F_DADOS);
close (F_COPIA);
```

Listagem 51: Programa que copia o conteúdo de um ficheiro para o outro

14.7. Redireccionar os ficheiros de entrada e saída

Quando se executa um programa num sistema UNIX pode-se redirecionar o STDIN (entrada padrão) e STDOUT (saída padrão). Supondo que temos o programa **listar.pl** e pretendemos que os dados de entrada sejam obtidos do ficheiro **bdados.txt** e os dados de saída sejam direcionados para o ficheiro **sdados.txt**. Neste exemplo podíamos executar o programa da seguinte forma: **listar.pl < bdados.txt > sdados.txt**

Dentro do programa, sempre que aparece-se a instrução **$line = <STDIN>;** os dados eram lidos do ficheiro **bdados.txt**, e a instrução **print "$line";** os dados eram enviados para o ficheiro **sdados.txt**.

14.8. Determinar o estado de um ficheiro

Considere o exemplo:

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

unless (open(FICHEIRO, "prog.txt"))  # o mesmo que if (! open(...) )
{
   if (-e "prog.txt")
   {
        die ("O ficheiro existe mas não pode ser aberto.\n");
   }
   else
   {
        die ("O ficheiro não existe.\n");
   }
}
```

Listagem 52 – Estado de um ficheiro

Neste programa utilizamos um operador de teste de ficheiros **(-e "prog.txt"),** que nos permite saber se o ficheiro existe. Se existir o valor devolvido é um valor diferente de zero, se o ficheiro não existir devolve zero.

A tabela seguinte apresenta os operadores de teste de ficheiros.

Operador	Descrição
-b	Dispositivo tipo bloco
-c	Dispositivo tipo carácter
-d	É um diretório
-e	O nome existe
-f	É um ficheiro
-g	Tem o bit *setgid* marcado
-k	Tem o bit *sticky* marcado
-l	É uma ligação
-o	O utilizador é o dono
-p	É um pipe
-r	Tem permissões de leitura
-s	Ficheiro não vazio
-t	É um terminal
-u	Tem o bit stuid marcado
-w	Tem permissões de escrita
-x	Tem permissões de execução
-z	Ficheiro vazio
-B	É um ficheiro binário
-S	É um *socket*
-T	É um ficheiro de texto

Tabela 8: Operadores para ficheiros

14.9. Ler uma sequência de ficheiros

Considere o seguinte exemplo:

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

while ($linha = <>)
{
   print ("$linha\n");
}
```

Listagem 53: Leitura de vários ficheiros

Supondo que o programa é executado com parâmetros, da seguinte forma: **prog.pl fich1.txt fich2.txt fich3.txt**. A instrução **($linha = <>)** lê a primeira linha do ficheiro fich1.txt, depois lê a segunda, até á última, quando terminar de ler todas as linhas do primeiro ficheiro começa a ler a informação dos restantes ficheiros. Pelo formato, o operador "<>" é apelidado de operador diamante.

Exercício 14

Elabore, com recurso à linguagem PERL, o script **`.template2perl.pl`**, cujo propósito é o de gerar um ficheiro cujo nome é indicado como único parâmetro da linha de comando. O ficheiro a gerar destina-se a ser empregue como base para a escrita de um programa em PERL, sendo que conteúdo do ficheiro deve respeitar o formato da listagem abaixo mostrado que corresponde à execução do script da seguinte forma:

./**`template2perl.pl a.pl`**

Tenha em atenção que os campos sublinhados (**Hostname, Author, Created**) devem ser gerados dinamicamente pelo script, e correspondem, respectivamente, ao nome da máquina, login do utilizador que executa o script e à data de criação.

```
#!/usr/bin/perl -w
use strict;

# Filename: 'a.pl'
# Hostname: 'xlinux'
# Author: 'root'
# Created: 'Wednesday 20070509_12h39'
#=====================================================
# Config
#=====================================================
# Inserir a configuracao aqui
my (variaveis a declarar);
#=
# Code
#=
```

15. Passar valores pela linha de comando

15.1. Argumentos da linha de comando

A linguagem PERL permite a utilização da linha de comando para passar argumentos. Os argumentos são guardados numa variável especial do tipo array (@ARGV). Por exemplo, se executarmos o seguinte programa: **prog.pl fich1.txt fich2.txt fich3.txt**, a variável **@ARGV** terá os seguintes elementos ("**fich1.txt", "fich2.txt","fich3.txt**")

Para acedermos individualmente aos elementos da variável @ARGV, utilizam-se as técnicas de acesso a elementos de arrays, ilustradas no exemplo seguinte.

NOTA: Tenha em atenção, que em PERL, o vetor de argumentos "@ARGV" não contém o nome da script, sendo que $ARGV[0] corresponde ao primeiro argumento passado pela linha de comando. O nome da script está acessível através da variável especial **$0**.

@ARGV Contém a lista de argumentos passados para o scripts.
$ARGV[0] é o primeiro argumento.
$ARGV $ARGV (notar a ausência de parênteses rectos de indexação) contém o nome do fx que está a ser lido
$0 O nome do script que está a ser executado
$ Separador de linha na saída. Por omissão, é nulo, logo entre dois print (ou outra função de saída) é necessário definir "\n" quando: esta variável não está 'ativada' e se pretende terminar a linha
$/ Separador de linha na entrada. Por omissão é o newline (\n) $, Separador de campo na saída. Por omissão, é nulo, logo entre dois argumentos de um print (ou outra função de saída) é necessário definir "um separador" quando esta variável não está 'ativada' e se pretende separar os campos
$_ A variável de entrada, por omissão e argumento por omissão para diversas funções *buitin*. O man *perlfunc* descreve para cada função, o comportamento, em caso de omissão do argumento, se este for opcional.

```
#!/usr/bin/perl

#aviso de erro se não receber argumentos
if(@ARGV < 1){
  print "erro:   $0 fich1, fich2 .... fichn\n";
  exit;
}

$\ ="\n";
$, =' <-> '; #uma string qualquer
print "\n\t++++\n";
for (@ARGV){
        print;           #assume como argumento do print o valor $_
}
print "\n\t----\n";
$, =' :-> '; #uma string qualquer
foreach (@ARGV){ #por omissão $_ assume o valor iterado
        print "Tamanho do nome do fx $_",length;
}
print "\n\t****E agora o cat de todos os ficheiros recebidos como
argmentos***\n";
$/ = ord(0); #o separador de linha na entrada é o EOF (codigo ascii 0)

while (<>){      #lê o fx em cada iteração
        #$ARGV tem o nome do fx do qual se está a ler
        print "\n----------Ficheiro $ARGV --------------\n";
        #$_ por omissão contém o fx lido.
        # O print por omissão imprime $_
        print;
}
```

Listagem 54: Passagem de valores pela linha de comandos

```
#!/usr/bin/perl -w
use strict;

# Declaração de variáveis
my ($Argc);
my ($i);

#-----------------------------------------------
# Exemplo com os argumentos da linha de comando
# $ARGV[0]: 1º argumento da linha de comando
# Atenção $ARGV[0] não é o nome do script!
# O nome do script é obtido através de $0
# Executa o programa da seguinte forma
# nome.pl argumento_1 argumento_2 teste
#-----------------------------------------------

# Mostra nome do ficheiro a ser executado (i.e. script)
print("Nome do script: $0\n");

# Número de argumentos
$Argc = $#ARGV+1;
if ( $Argc > 0 ) {
  print("Existem $Argc argumentos da linha de comando: @ARGV\n");
}
else {
  print("Não existem argumentos da linha de comando\n");
}

$i = 0;
while ( $i < $Argc ) {
  print ("ARGV\[$i\]: $ARGV[$i]\n");  # Nota: \ antes de [ e ]
  $i++;
}
```

Listagem 55: Passagem de valores pela linha de comandos

15.2. Variável de Ambiente

Em PERL, existem ainda as variáveis de sistema, entre elas as mais utilizadas são a variável de ambiente, que é uma variável do tipo lista associativa (**%ENV**) onde são guardados diversos valores, como por exemplo o nome do utilizador que está a correr o programa (**$ENV{'USER'}**);

```
#!/usr/bin/perl –w

while(($chave, $valor) = each(%ENV))
{
  print "A chave $chave contem o valor de $valor\n";
}
```

Listagem 56: Listagem das variáveis de ambiente

Existe ainda o **$!** ou **$^E** que guardam o valor do último erro ocorrido;

```
#!/usr/bin/perl –w

open(FICHEIRO, "dados.txt") || die $!;
```

Listagem 57: Apresentação de Erros

16. Bibliografia

Para elaboração da ficha, foi empregue material, dos seguintes autores/locais:

- Documentação de Isidro Vila Verde - FEUP / Serprest
  http://paginas.fe.up.pt/~jvv/Assuntos/PERL/extradocs/perl-intro.html
  (Abril 2008)
- "Beginning PERL"
  http://www.perl.org/books/beginning-perl/ (Abril 2008)
- "Perl 5 Tutorial"
  http://www.cbkihong.com/download/perltut.pdf (Abril 2008)

17. Exercícios

1. Caça ao BUG, descubra os 4 erros no código que se segue:

```
# o meu código
#!/usr/bin/perl –w
use strict;

print ‘Escreve a palavra OPS: ’;
chomp($input=<STDIN>)
if($input=’OPS’){
  print “Obrigado!\n”;
} else {
  print “Não era essa a palavra!”;
}
```

2. Modificar o programa converte.pl, para converter graus Celsius em graus Fahrenheit.

3. Construa um programa em PERL que conte os dígitos de um número introduzido pelo utilizador.

4. Recorrendo à linguagem PERL, escreva o script *avg_e_std.pl*, que recebendo números através da entrada padrão (um número por linha), deverá devolver na saída padrão a contagem dos números, a indicação do menor e do maior número, bem como a média aritmética e o desvio padrão dos números. A saída deve apresentar o formato abaixo mostrado, quando a script é executada da seguinte forma:

```
seq 10000 | avgstd.pl

== SAIDA ==
#
# CONTAGEM=10000
# MENOR=1
# MAIOR=10000
# SOMA=50005000
# MEDIA=5000.5
# DESVIO_PADRAO=2886.75133151437
# CONTAGEM:MENOR:MAIOR:SOMA:MEDIA:DESVIO_PADRAO
10000:1:10000:50005000:5000.5:2886.75133151437
```

Nota: o desvio-padrão de um conjunto de números X é dado pela seguinte expressão:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

5. Elabore o programa EPrimo.pl, que verifica se o número indicado pelo utilizador através da entrada padrão é primo ou não. (sugestão: pode recorrer ao utilitário "fator" – man fator)

6. Aproveitando o programa EPrimo.pl, construa o programa ContaPrimos.pl que indica o número de primos existentes, entre 1 e 100000. Procure otimizar o seu código.

7. Elabore o programa Factor.pl, que dado um número, indica os seus fatores primos.

Sugestão: man fator

8. Escreva um *script*, que peça ao utilizador um número, adicione 3, depois multiplica-o por 2, subtrai 4, subtrai duas vezes o número introduzido pelo utilizador, adicione 3, e imprime o valor final.

9. Recorrendo à linguagem PERL, elabore a script "**plotprimos.pl**" que recebendo um, dois número na linha de comando ($numMin e $numMax) os deve interpretar,

respetivamente, como o valor menor e o valor maior de uma gama de números inteiros. A script deve então calcular, para cada inteiro compreendido na gama, o seu número de fator primos, produzindo um ficheiro de nome "primos_$numMin_a_$numMax.dat" e que contém em cada linha o número seguido do respetivo número de fatores primos (estando os valores separados por "|"). A linha do ficheiro deve estar ordenada, por ordem crescente.

```
plotprimos.pl 10 20

ficheiro primos_10_20.dat
10|2
11|1
12|3
13|1
14|2
15|2
16|4
17|1
18|3
19|1
20|3
```

10. Recorrendo à linguagem PERL

a) Escreva o script *multiplica_V1.pl*, que deverá mostrar o resultado da multiplicação dos números, fornecidos através de um ficheiro de texto (um número por cada linha).

b) Escreva o script *multiplica_V2.pl*, que deverá mostrar o resultado da multiplicação dos números, fornecidos através de um ficheiro de texto (um número por cada linha). Por exemplo, o script deverá poder ser executado da seguinte forma:

```
seq 1000 | ./multiplica_V2.pl
```

```
## Solução – "multiplica_V2.pl"
```

11. Recorrendo à linguagem PERL

a) Construa o *script* Txt2Html.pl, que a partir de um ficheiro de texto, escreve o código HTML, capaz de exibir o ficheiro de texto.

b) Repita a alínea anterior, de modo a nome do ficheiro de texto a ser convertido passe a ser especificado através da linha de comando. O conteúdo HTML do *script* deve ser escrito num ficheiro que mantém o nome do ficheiro original acrescido da extensão html.

Sugestão – esqueleto da página HTML

<HTML> <HEAD> <TITLE> ficheiro </TITLE> </HEAD>
<BODY>
Texto
</BODY>
</HTML>

12. Recorrendo a um *vetor associativo* (vulgo *hash*), construa uma script que efetua a contagem das palavras de um ficheiro de texto, cujo nome é indicado através do 1º argumento da linha de comando.

a) Indique quantas vezes, existe a palavras "*xpto*"

b) Mostre a frequência, de cada palavra, do ficheiro

c) Repita a alínea anterior, mas exibindo a lista ordenada da(s) palavra(s) mais frequente(s) para as menos frequente(s).

Nota:

```
@palavras = split ' ', $Linha; # palavras recebe as palavras da $linha
```

13. Recorrendo à linguagem PERL, construa o *script* Users2Html.pl, que permita saber os utilizadores com conta na máquina, e que construa um ficheiro, com o formato HTML, onde a informação dos utilizadores, deve ser colocada.
O nome do ficheiro deve ser index.html, e deve ser passado no momento em que se executa o programa. **Users2Html > index.html**

Sugestão – esqueleto da página HTML

<HTML> <HEAD> <TITLE> ficheiro </TITLE> </HEAD> <BODY> <H1> cabeçalho </H1> </BODY> </HTML>

14. Recorrendo à linguagem PERL, construa o *script* mail2Html.pl, que permita saber o endereço de correio electrónico dos utilizadores, com conta na máquina e que construa um ficheiro com o formato HTML, onde essa informação deve ser colocada. O nome do ficheiro deve ser index.html, e o programa deve ser executado sem parâmetros, da seguinte forma: mail2Html.pl.

18. Soluções

De seguida são mostradas as soluções para alguns dos exercícios propostos.

18.1. Resolução do exercício 1

```
# o meu código -> não pode estar aqui!
#!/usr/bin/perl –w
use strict;
my $input;
print ‘Escreve a palavra OPS: ’;
chomp($input=<STDIN>);
if($input eq ’OPS’){
  print ("Obrigado!\n");
} else {
  print ("Não era essa a palavra!\n");
}
```

18.2. Resolução do exercício 2

```
#!/usr/bin/perl -w
use strict;

my $fahr=0;
my $cel=0;

print "Introduza a temperatura em Celsius: ";
chomp ($cel=<STDIN>);
$fahr=32+$cel*9/5;
printf("%.2fº Celsius = %.2fº Fahrenheit\n", $cel, $fahr);
```

18.3. Resolução do exercício 3

```
#!/usr/bin/perl –w
use strict;

my $palavra="";
```

```
while($palavra ne "fim")
{
  print "Dá-me uma frase ou palavra: ";
  chomp($palavra=<STDIN>);
  print "Escreveste: $palavra \n\n";
};
print "O programa chegou ao fim\n";
```

18.4. Resolução do exercício 4

```
#!/usr/bin/perl -w
use strict;
my $num=0;
my $n=0;
my $conta=0;
print 'Escreve um número inteiro: ';
chomp($num=<STDIN>);
$n=$num;
while($num>0)
{
  $conta+=1;
  $num=int $num/10;
}
if($n==0)
{
  $conta=1;
}
printf("%d tem %d digitos\n", $n, $conta);
```

18.5. Resolução do exercício 5

```
#!/usr/bin/perl -w
use strict;
my $n;
do
{
  print 'Escreve um número entre 0 e 100: ';
  chomp($n=<STDIN>);
}
while($n<0 || $n>100);
print"Escreveu o número $n\n";
```

18.6. Resolução do exercício 13

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

$title = "Identificacao";
$h1 = Utilizadores;
my $cmd = "cut -f1 -d: /etc/passwd";
open (IN, "$cmd |") || die ("Erro a abrir o ficheiro: $!\n");

@lista = <IN>;

print "<HTML>\n";
print "<HEAD>\n";
print "<TITLE>",$title,"</TITLE>\n";
print "</HEAD>\n";
print "<BODY>\n";
print "<H1><center>",$h1,"</center></H1>";

foreach $user (@lista)
{
  print "<br>$user";
}

print "</BODY>\n";
print "</HTML>";
```

18.7. Resolução do exercício 14

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

$elm = “”;
$title = "Identificacao";
$h1 = Utilizadores;
$Maquina = `hostname –f`  # Nome da máquina para forma endereço de
mail
my $cmd = "cut -f1 -d: /etc/passwd";
open (IN, "$cmd |") || die ("Erro a abrir o ficheiro.\n");
open (OUT,”>index.html”) || ("Erro a abrir o ficheiro index.html.\n");

@lista = <IN>;
```

```
print OUT "<HTML>\n";
print OUT "<HEAD>\n";
print OUT "<TITLE>",$title,"</TITLE>\n";
print OUT "</HEAD>\n";
print OUT "<BODY>\n";
print OUT "<H1><center>",$h1,"</center></H1>";

foreach $user (@lista)
{
   $elm = “$user\@$Maquina”;
   print OUT "<br>$elm";
}

print OUT "</BODY>\n";
print OUT "</HTML>";
```