Arrays asociativos (Tablas Hash)

- El acceso a la información se realiza mediante una clave.
  - ✓ Estructura típica de diccionario: clave → dato.

```
my %guia;
$guia{"Jose"} = "96 386 43 00";
$guia{"Ana"} = "656 66 77 88";
```

Clave

Dato

Tablas evaluadas como listas

- Las tablas se pueden evaluar como listas, donde los elementos pares son las claves que identifican a los elementos impares (datos);

```
my %guia;
$guia{"Jose"} = "96 386 43 00";
$guia{"Ana"} = "656 66 77 88";

my @guiaLista = %guia;
```

```
@guiaLista → ( "Jose", "96 386 43 00", "Ana", "656 66 77 88")
```

Funciones: keys, values

- Permiten extraer la lista de claves y la lista de datos de una tabla.

```
my %guia = ("Jose", "96 386 43 00", "Ana", "656 66 77 88");
my @apellidos = keys %guia;
my @telefonos = values %guia;
```

("Jose", "Ana")

("96 386 43 00", "656 66 77 88")

Iterar sobre tablas

```
my %guia = ("Jose", "96 386 43 00", "Ana", "656 66 77 88");
my ($clave, $dato); #Declara 2 variables a la vez
while ( ($clave, $dato) = each (%guia) )
{ #hacer algo que NO cambie los valores de la tabla }
```

Extrae parejas clave-dato de la tabla

```
my %guia = ("Jose", "96 386 43 00", "Ana", "656 66 77 88");
foreach my $clave ( keys %guia )
{ #hacer algo con la tabla }
```

Recorre todas las claves de la tabla

Ordenar tablas

```
my %guia = ("Jose", "96 386 43 00", "Ana", "656 66 77 88");
foreach my $clave sort( keys %guia )
{
    print "$clave $guia{clave}\n"
}
```

Recorre por orden alfabético todas las claves de la tabla

```
Ana  656 66 77 88

Jose 96 386 43 00
```

Comprobar si un elemento existe

```
my %guia = ("Jose", "96 386 43 00", "Ana", "656 66 77 88");
print "Escribe el nombre de tu amigo\n";
my $amigo= <STDIN>;
chomp($amigo);
if (exists $guia{$amigo})
{
   print "Telefono de $amigo es $guia{$amigo}\n";
}
else { print "El teléfono de $amigo no está en la guia\n";
```

Comprueba si una determinada Clave se encuentra en una tabla

```
Escribe el nombre de tu amigo
Jose
Telefono de Jose es 96 386 43 00
```

Subrutinas en Perl

- Es posible definir subrutinas que encapsulan bloques de código.

```
#!/usr/bin/perl
use strict;
use warnings;
my $x = suma();
print $x, "\n";
sub suma
{
        my ($x, $y) = (1, 2);
        return ($x + $y);
}
```

Llamada a la subrutina

Subrutina

Variables locales

Parámetros

- No hay parámetros formales como en otros lenguajes.
- Hay un array de parámetros: `@_`

```
#!/usr/bin/perl
use strict;
use warnings;
my $x = suma(1,2);
print $x, "\n";
sub suma
{
        return ( $_[0] + $_[1]);
}
```

Parámetros (2)

- Versión con un número indeterminado de parámetros a sumar

```
my $x = suma(1,2,3,4,5);
print $x, "\n";

sub suma {
       my $s;
       foreach my $x (@_) {
              $s += $x;
       }
       return ($s);
}
```

Más sobre Expresiones Regulares

- Recordar/Extraer una o varias partes del encaje.

```
#!/usr/bin/perl
use strict;
use warnings;
while (defined (my $linea= <STDIN>))
{
       if ($linea =~ /^#[^!](.*)$/ )
       { print $1;}
}
```

Extraemos la parte que
Encaja dentro de los paréntesis
Con la variable $1

Más sobre Expresiones Regulares (II)

$linea="La profesora de teoría y la profesora de practicas"

- **Buscar y Reemplazar**

```
my $linea =~ s/profesora/Xaro/;
```

Sustituimos la cadena "profesora"
Por la cadena "Xaro"

- **Opciones:**
  - ✓ g: Reemplaza todas las ocurrencias del patrón
  - ✓ i: El encaje no es sensible a las mayúsculas/minúsculas
  - ✓ e: La cadena que se reemplaza se evalúa como una expresión regular en Perl

```
my $linea =~ s/profesora/Xaro/g;
my $linea =~ s/profesora/Xaro/ig;
```

¿?

Operadores sobre ficheros

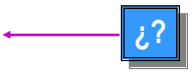
- ✓ -r –w –x Fichero se puede leer/escribir/ejecutar
- ✓ -e –z Fichero existe/ tamaño cero
- ✓ -s Fichero existe y no tiene tamaño cero, devuelve el tamaño
- ✓ -f –d Si es fichero, o directorio

```
# Comprobar si es un fichero o directorio
if ( -f $nom)
{
    if (-r $nom) { print "Es un fichero y se puede leer;}
    else {print "Es un fichero pero no lo podemos leer;}
}
elsif (-d $nom) { print "Es un directorio;}
```

Funciones sobre cadenas

- **length:** devuelve la longitud del string
- **lc:** devuelve la versión en minúscula de la expresión que le pasamos
- **uc:** devuelve la versión en mayúscula de la expresión que le pasamos
- **ucfirst, lcfirst:** devuelve solo el primer carácter de la expresión en mayúscula o minúscula respectivamente
- **index,reindex:** devuelve la primera posición (o última) de la subcadena en la cadena que le pasamos
- **substr:** Extrae una subcadena de una expresión con longitud l

```
$frase= "hola mundo, hola tierra"
$len= length($frase);
$may1= uc($frase);
$may2= ucfirst($frase);
$pos1= index($frase,"hola");
$pos2= index($frase,"hola",$pos1+1);
```



Ejemplos de programas PERL

Invertir cada una de las líneas de un archivo

```
#usr/local/bin/perl
## Invertir cada una de las lineas de un archivo
use strict;
use warnings;

## Comprobar los argumentos en línea del programa
## Numero de argumentos == 2
if ( $#ARGV != 1 ) {
       die "Uso: $0 entrada salida\n";
}

my ($inf, $outf) = @ARGV;
```


```
## Validar los archivos

#1: $inf se puede leer?
if ( ! -r $inf ) {
       die "$inf no se puede leer\n";
}
#2: $inf es un fichero?
if ( ! -f $inf ) {
       die "$inf no es un fichero\n";
}
#3: $inf es correcto, abrir
open (ENTRADA, $inf) || die "No se puede abrir $inf : $!\n";
```

```
#4: Existe $outf?
my $resp = 'r';
if ( -e $outf ) {
    print STDERR "El fichero de salida $outf ya existe!\n";
    do {
        print STDERR "reemplazar, añadir, salir (r,a,s)? ";
        $resp = getc (STDIN);
    }
    while ( $resp ne 'r' && $resp ne 'a' && $resp ne 's' );
    if ( $resp eq 's' ) { exit; }
}
```


```
#Se puede proceder
my $modo;
if ( $resp eq 'r' ) {
    $modo = ">"; #reemplazar
}
else {
    $modo = ">>"; #añadir
}
open (SALIDA, "$modo$outf") || die "No se puede abrir $outf : $!\n";
#leer linea, invertir, escribir
my $linea;
while ( defined ( $linea = <ENTRADA> ) ) {
    chomp ($linea);
    $linea = reverse ($linea);
    print SALIDA $linea, "\n";
}
close (ENTRADA);close (SALIDA);
```

ENTRADA

```
#!/usr/bin/perl
use strict;
use warnings;
print "Nombre de usuario? ";
my $usuario;
$usuario = <STDIN>;
chomp($usuario);
print "Hola, $usuario.\n";
```

SALIDA

```
lrep/nib/rsu/!#
;tcirts esu
;sgninraw esu
;" ?oirausu ed erbmoN" tnirp
;oirausu$ ym
;>NIDTS< = oirausu$
;)oirausu$(pmohc
;"n\.oirausu$ ,aloH" tnirp
```

Examen Junio 2003

Tenemos tres ficheros que contienen información sobre los alumnos matriculados en la asignatura de Herramientas de Programación: el fichero "matriculas.dat" contiene el nombre de los alumnos (nombre y apellidos) y su código de identificación; el fichero "notas.dat" contiene el código de identificación del alumno y siete calificaciones del mismo (de 0 a 10), las cinco primeras son notas de laboratorio y las dos últimas son las notas de dos exámenes realizados (nota de teoría). La nota final de cada alumno es el 20% de la nota de laboratorio y el 80% la nota de teoría. Por último, el fichero "convocatoria.dat" que contiene el código de identificación del alumno y la convocatoria en la que se encuentra.

Se pide, escribir un programa en Perl que realice unas estadísticas sobre las notas obtenidas por los alumnos, indicándonos los porcentajes de aprobados y suspendidos por convocatoria. Los resultados de estas estadísticas se escribirán en un fichero. La tabla estadística que se escribirá debe ser como la siguiente:

	1	2	3	4	5	6
Aprobados %						
Suspendidos %						

También queremos saber el alumno que ha obtenido la nota más alta y el que ha obtenido la nota más baja. De estos dos alumnos deseamos conocer sus nombres completos, su código de identificación y en qué convocatoria se encuentra.

Examen Junio 2003 (II)

Ayuda: Ejemplos de los contenidos de los ficheros.

"matriculas.dat"

```
Jesús Albert Blanco 40220
Vicente Cavero Garcia 89898
Xaro Benavent Garcia 43423
Juan de Mata Domingo Esteve 5555
```

"notas.dat"

```
89898 1.2 8.9 2.3 4.2 4.2 5.2 2.1
40220 5.6 5.7 8.2 4.2 7.8 8.2 9.5
5555 2.5 2.3 4.2 5.2 5.2 5.3 7.8
43423 8.5 6.7 8.2 9.2 5.6 6.7 8.9
```

"convocatoria.dat"

```
89898 5
43423 2
40220 1
5555 2
```

Septiembre 2003

Se desea realizar un programa en *Perl* para la detección de mensajes de correo electrónico potencialmente peligrosos, por ser susceptibles de distribuir un virus informático. El programa debe ser capaz de detectar los mensajes que contienen palabras que incluyan 3 vocales consecutivas. En caso de detectar ese tipo de palabras en un *email*, el programa debe indicar el remitente y el número de palabras de este tipo localizadas en el mensaje.

Para realizar el programa hay que saber que un mensaje de correo electrónico es un fichero texto, formado por unas líneas de cabecera y un cuerpo con el contenido del mensaje. Las líneas de cabecera siempre tienen el mismo formato:

<etiqueta>: <texto>

Ejemplos de líneas de cabecera:

De: Jesús Albert

Para: webmaster@perl.org

Asunto: Pregunta de examen

El número de líneas que componen la cabecera es variable y dependen del *email*. El cuerpo con el texto del mensaje se inicia inmediatamente después de la última línea de cabecera.

Finalmente, hay que tener en cuenta que las palabras *prohibidas* deben aparecer en el cuerpo del mensaje, no en las líneas de cabecera y el remitente del correo está indicado en la línea de cabecera con la etiqueta "De" (que siempre existe).