Arrays

- **Semánticamente, los arrays de Perl están cerca de las listas.**
  - ✓ El tamaño crece y disminuye dinámicamente.
- **Sintácticamente, están cerca de los arrays de *C*.**
  - ✓ Se puede acceder a los elementos mediante índices.
- **En algunos casos pueden llegar a ser algo ineficientes.**
- **Un array en Perl es siempre una lista de escalares.**

Formación de listas

```
#!/usr/bin/perl
use strict;
use warnings;
my @lista_1 = qw / a b c /;
my @lista_2 = ( 1, 2, 3, 4 );
my @todo = ( @lista_1, @lista_2 );
foreach my $i (@todo)
{
        print $i;
}
print "\n";
```

Operador de entrecomillado qw

Carácter de final de qw

Formación de listas

```
#!/usr/bin/perl
use strict;
use warnings;
my @lista_1 = qw / a b c /;
my @lista_2 = ( 1, 2, 3, 4 );
my @todo = ( @lista_1, @lista_2 );
foreach my $i (@todo)
{
        print $i;
}
print "\n";
```

Concatenación de elementos (listas)

Formación de listas

```
#!/usr/bin/perl
use strict;
use warnings;
my @lista_1 = qw / a b c /;
my @lista_2 = ( 1, 2, 3, 4 );
my @todo = ( @lista_1, @lista_2 );
foreach my $i (@todo)
{
        print $i;
}
print "\n";
```

```
@lista_1 = "a" "b" "c"
@lista_2 = 1 2 3 4
@todo = "a" "b" "c" 1 2 3 4
```

Recorrer todos los elementos de la lista `@todo`

Acceso a los elementos (escalares)

- Sea `@array` de ***n*** elementos:

  `$array[0]`, `$array[1]`, . . . , `$array[n-1]`,
  representan las n variables escalares que contiene el array.

- `$#array` : contiene el índice del último elemento del array.

  `$array[$#array]` es siempre el último elemento del array.

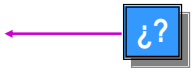
  `$#array+1` es siempre la longitud del array.

  `$#array` se puede modificar.

Subarrays

- En ocasiones es útil crear un nuevo array basado en un subconjunto de elementos de otro array.
  - ✓ Ejemplo:

```
#!/usr/bin/perl
use strict;
use warnings;
my @frase = qw / Esto es una lista /;
my @una_lista = @frase[ 2 .. $#frase];
my @inicio = @frase[ 0 .. 1];
@una_lista = @frase[1, $#frase];
```



Arrays como Pilas

- Una Pila es una lista con operaciones especiales de inserción y borrado.
- Funciones `push` y `pop` sobre arrays.

```
my @pila;
push (@pila, 5, 3, "hola");
my $dato = pop (@pila);
```

```
@pila = qw / 5 3 hola /;
```

```
$dato = "hola";
@pila es ( 5, 3 );
```

- Insertar en Colas: `shift` y `unshift`.

Funciones sobre arrays

- `sort` ordena los elementos de un array en orden alfabético
- `reverse` devuelve una lista en orden inverso dejando la original
- `join` toma un string como separador y un array, y convierte el array en un escalar intercalando el separador
- `split` Realiza la acción contraria a join

```
@colores = qw /azul rojo verde/
```

```
my @colores= (azul,rojo,verde);
my $union_col= join("&",@colores);
my @sep_col= split("&",$union_col;
```

```
$union_col= "azul&rojo&verde";
@sep_col= /azul rojo verde/;
```

Argumentos en línea

- `@ARGV` es el array que contiene los argumentos en línea del programa en Perl.
- `$#ARGV` es el índice del último argumento.

```
#!/usr/bin/perl
use strict;
use warnings;
#cuantos argumentos has pasado?
print "Has pasado ", $#ARGV+1, " argumentos.\n";
#mostrar los argumentos por lineas
foreach my $i (@ARGV)
{ print "$i\n"; }
```

Ejecución

```
sh-2.05$ perl argumentos.pl 1 2
Has pasado 2 argumentos.
1
2
sh-2.05$
```

Estructuras de Control en Perl

- Bloques (secuencias): `{ }`
- Condicionales: `if` / `unless`
- Iterativas:
  - ✓ `while`
  - ✓ `do – while`
  - ✓ `for`
  - ✓ `foreach`

if – elsif – else

```
if ( expresion )
{
        bloque de sentencias
}
elsif ( otra_expresion )
{
        bloque de sentencias
}
else
{
        bloque de sentencias
}
```

Opcionales

Operadores de comparación

Significado	Op. numérico	Op. String
menor que	<	lt
menor o igual que	<=	le
mayor que	>	gt
mayor o igual que	>=	ge
igual que	==	eq
distinto que	!=	ne
comparar con	<=>	cmp

Resultado 1 o 0, según se cumpla o no la condición establecida.

-1 sii $izq es menor que $der,
0 sii $izq es igual que $der,
+1 sii $izq es mayor $der

while / do - while

```
while ( expresion )
{
       bloque de sentencias
}
#-------------------------
do
{
       bloque de sentencias
}
while ( expresion )
```

for / foreach

```
for ( sent. inicial;  expresion; sent. incremento)
{
       bloque de sentencias
}
#-------------------------
foreach escalar ( array )
{
       bloque de sentencias
}
```

Manejo de archivos

- **Manejadores de archivos: nombres sin carácter inicial especial.**
  - ✓ Convención: nombres en mayúsculas.
    `ENTRADA`, `DATOS`, etc...
- **Archivos estándar: `STDIN`, `STDOUT`, `STDERR`.**
- **Operador de lectura: `< >`**
  `<ENTRADA>`: lee un línea del archivo, incluido "\n".

Abrir y cerrar archivos

- **`open`: conecta manejador con archivo físico.**

```
open ( ENTRADA, "notas.dat");
```

Leer

```
open ( SALIDA, ">notas.dat");
```

Escribir

```
open ( SALIDA, ">>notas.dat");
```

Añadir

- **`close (F)` : cierra el archivo.**
- **`open ( F, $nombre ) || die ("No puedo abrir $nombre.\n");`**

Variantes de entrada

- `@a = < FICH >`, lee todo el archivo en un array de líneas.
- ¿Qué es una línea? → Depende de la variable `$/`, que contiene el marcador de fin_de_línea (normalmente `\n`).
- Se puede cambiar `$/` para cambiar el concepto de línea.
  - ✓ Por ejemplo:

```
$/ = undef;
$todo = <FICH>;
## lee todo el archivo en un string.
```

Salida a archivo

- `print` admite como argumento opcional el manejador de archivo:

```
print FICH "Se puede escribir",
  "cualquier cosa \n", $dato;
```

  - ✓ No hay , detrás de FICH

Conectar con el S.O.

- `open` permite establecer una conexión con un proceso lanzado por el sistema operativo:

```
#!/usr/bin/perl
use strict;
use warnings;
my $linea;
open (F, "ls -l |");
while ( defined ( $linea = < F > ) ) {
       print $linea;
}
```

Invocar programas externos

- Es fácil llamar a programas externos mediante `system`.

```
if ( system ( "mail usuario < fichero.txt" ) != 0 )
{
       die "Error de sistema: $?";
}
else...
```

Estado devuelto por la última ejecución de una orden del sistema.