3.- LENGUAJE Perl

Practical Extraction and Report Language

http://www.ebb.org/PickingUpPerl/

¿Qué es Perl?

- Lenguaje de programación interpretado.
- Lenguaje con potentes funciones de manipulación de textos.
- Combina (de manera ecléctica) características de muchos lenguajes de órdenes.

Primer programa en Perl

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

Primer programa en Perl

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

El archivo se debe ejecutar como un programa *Perl*

Primer programa en Perl

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

Pragma:
`strict` = Compilación estricta del código
`perl -c <archivo>`
`warnings` = Emitir avisos durante ejecución

Primer programa en Perl

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

Escribir en `stdout`

Primer programa en Perl

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

Declaración de variable. En modo `strict` todas las variables se deben declarar.

Primer programa en Perl

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

Asignación a la variable de la siguiente línea de `stdin`.

Primer programa en Perl

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

Elimina carácter de fin de línea al final de la variable

Primer programa en Perl

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

Imprime en `stdout` el nombre del usuario.

Datos en Perl

- **Escalares ($): valores numéricos o strings (char) determinado en función del contexto.**

```
123   12.67   "Nombre de usuario"   `date`   'Don\'t'
```

  - '' (comillas simples), no admiten sustitución, excepto \\ y \'
  - "" (comillas dobles), admiten sustitución de variables, $usuario y de códigos de control, \n.
  - `` (comillas simples invertidas), admiten sustitución y se intenta ejecutar el contenido como una orden de la shell.
- **Arrays (@): listas de valores**
- **Arrays asociativos (%) (tablas hash)**

Segundo programa en Perl

```
#!/usr/bin/perl
use strict;
use warnings;

my $amo = 'Isabel';
my $perro = 'Toby';
my $cantidad = 1.5;
my $que = 'comida para perros';

print "El perro de $amo, $perro,
se ha comido $cantidad kilos de $que.\n";
```

Variables escalares (inicializadas)

Segundo programa en Perl

```
#!/usr/bin/perl
use strict;
use warnings;

my $amo = 'Isabel';
my $perro = 'Toby';
my $cantidad = 1.5;
my $que = 'comida para perros';

print "El perro de $amo, $perro,
se ha comido $cantidad kilos de $que.\n";
```

Comillas dobles admiten sustitución.

Segundo programa en Perl

```
#!/usr/bin/perl
use strict;
use warnings;

my $amo = 'Isabel';
my $perro = 'Toby';
my $cantidad = 1.5;
my $que = 'comida para perros';

print "El perro de $amo, $perro,
se ha comido $cantidad kilos de $que.\n";
```

El cambio de línea es un carácter válido para formar un string.

Segundo programa en Perl (ejecución)

```
#!/usr/bin/perl
use strict;
use warnings;

my $amo = 'Isabel';
my $perro = 'Toby';
my $cantidad = 1.5;
my $que = 'comida para perros';

print "El perro de $amo, $perro,
se ha comido $cantidad kilos de $que.\n";
```

```
sh-2.05$ perl programa2.pl
El perro de Isabel, Toby,
se ha comido 1.5 kilos de comida para perros.
sh-2.05$
```

Segundo programa en Perl (variante)

```
#!/usr/bin/perl
use strict;
use warnings;

my $amo = 'Isabel';
my $perro = 'Toby';
my $cantidad = 1.5;
my $que = 'comida para perros';

print "El perro de ${amo}ita, $perro,
se ha comido $cantidad kilos de $que.\n";
```

En caso de ambigüedad en el nombre, delimitar el identificador.

Segundo programa en Perl (variante + ejecución)

```
#!/usr/bin/perl
use strict;
use warnings;

my $amo = 'Isabel';
my $perro = 'Toby';
my $cantidad = 1.5;
my $que = 'comida para perros';

print "El perro de ${amo}ita, $perro,
se ha comido $cantidad kilos de $que.\n";
```

```
sh-2.05$ perl programa2b.pl
El perro de Isabelita, Toby,
se ha comido 1.5 kilos de comida para perros.
sh-2.05$
```

Variables indefinidas

- ¿Qué valor tiene una variable escalar si no se le da un valor?

```
use strict
my $sinValor;
```

- ¿Qué valor tiene `$sinValor`? **`undef`**

Ejemplo con variable `undef`

```
#!/usr/bin/perl
use strict;
use warnings;

my $conValor = 'Hola';
my $sinValor;

print "$conValor $sinValor\n";
```

Ejecución con `warning`

```
sh-2.05$ perl tres.pl
Use of uninitialized value in concatenation (.) at tres.pl line 8.
Hola
sh-2.05$
```

Ejemplo con `undef` y `defined()`

```
#!/usr/bin/perl
use strict;
use warnings;

my $conValor = "Hola";
my $sinValor;

print "definida \$sinValor == ", defined ($sinValor),
       ", definida \$conValor == ", defined ($conValor), "\n";

$sinValor = $conValor; #cambiar valores
$conValor = undef; #ahora indefinida

print "definida \$sinValor == ", defined ($sinValor),
       ", definida \$conValor == ", defined ($conValor), "\n";
```

Ejemplo con `undef` y `defined()` (ejecución)

```
#!/usr/bin/perl
use strict;
use warnings;

my $conValor = "Hola";
my $sinValor;

print "definida \$sinValor == ", defined ($sinValor),
       ", definida \$conValor == ", defined ($conValor), "\n";

$sinValor = $conValor; #cambiar valores
$conValor = undef; #ahora indefinida

print "definida \$sinValor == ", defined ($sinValor),
       ", definida \$conValor == ", defined ($conValor), "\n";
```

```
sh-2.05$ perl cuatro.pl
definida $sinValor == , definida $conValor == 1
definida $sinValor == 1, definida $conValor ==
sh-2.05$
```

Expresiones regulares (Repaso)

- **Expresión regular: patrón que define conjuntos de cadenas de caracteres.**
- **Se forman mediante operadores que combinan expresiones más pequeñas.**
- **Construcción:**
  - ✓ La mayoría de los caracteres son expresiones regulares que se representan a sí mismos (p.ej., letras y dígitos).
  - ✓ Existen algunos símbolos que son metacaracteres (tienen un significado especial):
    - Si van precedidos del carácter "\" ("se escapan") se traducen por sí mismos.

Formación de expresiones regulares

- **Lista de caracteres encerrada entre `[` y `]`: representa a cualquier carácter de esa lista (`or`).**
  - ✓ Ejemplo: `[0123456789]` = cualquier dígito.
  - ✓ `[0-9]` = cualquier dígito ("–" indica un rango)
- **Existen clases de caracteres identificadas de forma predefinida:**

  `[:alnum:]`, `[:alpha:]`, `[:crntl:]`,
  `[:digit:]`, `[:graph:]`, `[:print:]`,
  `[:punct:]`, `[:space:]`, `[:upper:]`,
  `[:xdigit:]`
  - ✓ Los caracteres [y] forman parte del nombre
  - ✓ Ejemplo: `[[:alnum:]] = [0-9A-Za-z]`

Formación de expresiones regulares (2)

- **`.` = cualquier carácter**
- **`^` = cadena vacía al principio**
- **`$` = final de línea**
- **`\<` = cadena vacía al inicio de una palabra**
- **`\>` = cadena vacía al final de una palabra**
- **`\w` = `[[:alnum:]]`**

Todas las palabras de principio de línea que empiecen por vocal

```
$ egrep "[^aeiou]"
```

La palabra "dos" a principio de línea

```
$ egrep "^\<aeiou\>"
```

Formación de expresiones regulares (3)

- **Operadores de repetición:**
  - ✓ `?` = El elemento que precede es opcional y aparece, como mucho, una vez.
  - ✓ `*` = El elemento que precede aparece cero o más veces.
  - ✓ `+` = El elemento que precede aparece una o más veces.
  - ✓ `{n}` = El elemento que precede aparece exactamente `n` veces.
  - ✓ `{n,}` = El elemento que precede aparece `n` o más veces.
  - ✓ `{n,m}` = El elemento que precede aparece, al menos, `n` veces pero no más de `m` veces.

NIF

```
$ egrep "[0-9]{8}[A-Z]{1}"
```

Números de teléfono en formato: 96-354-35-50

```
$ egrep "[0-9]{2}.[0-9]{3}.[0-9]{2}.[0-9]{2}
```

Formación de expresiones regulares (4)

- **Dos expresiones regulares se pueden concatenar:**
  - ✓ La expresión resultante representa a cualquier cadena formada concatenando 2 subcadenas que se ajusten a las subexpresiones.
- **Dos expresiones regulares se pueden unir mediante el operador `|` (infijo):**
  - ✓ La expresión resultante representa a cualquier cadena que se ajuste a cualquiera de las expresiones (or).
- **Reglas de precedencia:**

  Repetición > Concatenación > Unión

Ejemplos (Expresiones regulares)

Expresion	Selecciona
a.b	axb aab abb aSb a#b ...
[abc]	a b c
[aA]	a A
[aA][bB]	ab aB Ab AB
[A-Za-z]	A B C ... Z a b ... Z
[0-9]*	Cadenavacía 0 1 9 00 99 9999
[0-9][0-9]*	0 1 9 00 99 9999
^.*$	Cualquier línea completa

Ejemplos II (Expresiones regulares)

Expresion	Selecciona
abc+d	abcd abccd abcccd abccccccd
abc*d	abd abcd abccd abccccccd
123[a|b]*	123 123a 123b 123ab 123aaaab
123a{2,4}	123aa 123aaa 123aaaa
123a{2,}	123aa 123aaa 123aaaaaaaa
ab[c|C|d]	Abc abC abd
\<hola|adios|ciao\>	Hola adios ciao
^hola|adios	hola (principio de línea) y adios (en cualquier posición)
^(hola|adios)	hola o adios (principio de línea)

Programa con Expresiones Regulares

```
#!/usr/bin/perl
use strict;
use warnings;

my $codigo = 0;
my $vacias = 0;
my $comentarios = 0;
my $linea;

while ( defined ($linea=<STDIN>) )
{
        if ( $linea =~ /^\n/ ) #string "encaja con" expr. regular
                { $vacias++; }
        elsif ( $linea =~ /^#[^!]/ )
                { $comentarios++; }
        else { $codigo++; }
}

print "Lineas de codigo: $codigo, ";
print "comentarios: $comentarios y vacias: $vacias.\n";
```

Objetivo:
Contabiliza líneas con código, con comentarios y vacías de un script introducido por teclado.

Programa con Expresiones Regulares

```
#!/usr/bin/perl
use strict;
use warnings;

my $codigo = 0;
my $vacias = 0;
my $comentarios = 0;
my $linea;

while ( defined ($linea=<STDIN>) )
{
        if ( $linea =~ /^\n/ ) #string "encaja con" expr. regular
                { $vacias++; }
        elsif ( $linea =~ /^#[^!]/ )
                { $comentarios++; }
        else { $codigo++; }
}

print "Lineas de codigo: $codigo, ";
print "comentarios: $comentarios y vacias: $vacias.\n";
```

Lee línea mientras haya alguna.

Programa con Expresiones Regulares

```
#!/usr/bin/perl
use strict;
use warnings;

my $codigo = 0;
my $vacias = 0;
my $comentarios = 0;
my $linea;

while ( defined ($linea=<STDIN>) )
{
        if ( $linea =~ /^\n/ ) #string "encaja con" expr. regular
                { $vacias++; }
        elsif ( $linea =~ /^#[^!]/ )
                { $comentarios++; }
        else { $codigo++; }
}

print "Lineas de codigo: $codigo, ";
print "comentarios: $comentarios y vacias: $vacias.\n";
```

Operador “matching”.

Comienza con un cambio de línea.

Programa con Expresiones Regulares

```
#!/usr/bin/perl
use strict;
use warnings;

my $codigo = 0;
my $vacias = 0;
my $comentarios = 0;
my $linea;

while ( defined ($linea=<STDIN>) )
{
        if ( $linea =~ /^\n/ ) #string "encaja con" expr. regular
                { $vacias++; }
        elsif ( $linea =~ /^#[^!]/ )
                { $comentarios++; }
        else { $codigo++; }
}

print "Lineas de codigo: $codigo, ";
print "comentarios: $comentarios y vacias: $vacias.\n";
```

[^...] encaja con caracteres que no son “...”

Comienza con “#” y NO sigue con “!”.