Unos programas en PERL

Luis Gerardo de la Fraga

9 de agosto de 2002

Resumen

Se presentan cuatro programas de PERL. En el primero se introducirá el uso de PERL programando el algoritmo de Eratóstones para hallar números primos. En los siguientes tres programas se visualizará el poder de este lenguaje como un medio para conectar varias aplicaciones.

1 Introducción

PERL es un lenguaje de muy alto nivel, en donde no debemos teclear tantas instrucciones para realizar lo que queramos.

En PERL no hay declaracion de variables. Sólo existen tres tipos de datos: variable simples, arreglos y tablas de hash (semejantes a un arreglo, sólo que el índice puede ser cualquer cosa). Y su sintaxis es muy parecida a la del lenguaje C, excepto que todas las variables deben comenzar con el signo $ y las llaves {} son obligatorias para indicar inicio y fin de bloque (en C, si el bloque consiste de una sola sentencia, ella misma forma el fin de bloque).

El programa "Hola mundo" en PERL puede verse en la Fig. 1.

```
#!/usr/bin/perl

print "Hola mundo\n";
```

Figura 1: El programa "hola mundo" en PERL

Si salvamos este programa en un archivo llamado `hola.pl`, para poderlo ejecutar debemos cambiarle los permisos y ejecutarlo propiamente de la siguiente manera:

```
$ chmod 700 hola.pl
$ ./hola.pl
```

2 Números primos

La forma más simple de hallar un número primo es por medio del *algoritmo de la criba* de Eratóstones: se divide n por todos los números hasta $\lfloor\sqrt{n}\rfloor$. Eratóstenes es un astrónomo y poeta griego quien hace más de dos mil años calculó la medida de la cincumferencia de la Tierra con un error del 2% de su valor real.

El algoritmo de Eratóstones queda en pseudocódigo como sigue:

1. Se tiene un arreglo, $p[]$, de tamaño n.
2. Se llena cada celda del arreglo con el mismo índice
3. Se calcula $k = \lfloor\sqrt{n}\rfloor$
4. Para cada entero $2 \leq i \leq k$ y donde $p[i]$ no tenga marca:
   (a) Se pone una marca en todos los múltiplos de i.
5. Los números primos menores a n están el lugar del arreglo $p[]$ en donde no hay marca.

Y realizando la codificación en PERL del pseudocódigo de arriba para encontrar los número primos menores a n, se obtiene lo siguiente:

```
#!/usr/bin/perl

# Algorimto de Eratóstenes para hallar
# los números primos menores a n
#
# Fraga 24/07/2002

$n = 50;  # El valor de n

for ( $i=1; $i<=$n; $i++ ) {
    $p[$i] = $i;
```

```
}

$k = int( sqrt($n) );

$i=2;
while ( $i <= $k ) {
    while ( $p[ $i ] == 0 ) {
        $i ++;
    }
    # los múltiplos de $i
    for ( $j=2; $j<=$n; $j++ ) {
        $a = $i * $j;
        $p[ $a ] = 0;
    }
    $i++;
}

# Imprimimos los números encontrados
for ( $i=1; $i<=$n; $i++ ) {
    if ( $p[$i] != 0 ) {
        printf ( "%d\n", $p[$i] );
    }
}
```

3 Una programa que suma dos números

Este es un programa simple, le llamaremos `prog`:

```
#!/usr/bin/perl
#
# Fraga 23/03/1999
#
if ( $#ARGV == 0  ) {
    die "Args: a b\n";
}
elsif ( $#ARGV == 1 ) {
    $a = $ARGV[0];
    $b = $ARGV[1];
}
else {
    print "De el valor de a = ";
    $a = <STDIN>; chop $a;
    print "De el valor de b = ";
    $b = <STDIN>; chop $b;
}
$c = $a + $b;
print "$a + $b = $c\n";
```

Su lógica es simple: si se da un solo argumento, `$#ARGV == 0`, el programa termina. Si se dan dos argumentos al programa, `$#ARGV == 1`, estos son números que se van a sumar. Y si no se dan argumentos se pregunta por cada número. Al final imprime los dos números y su suma.

4 Llamadas repetidas al programa

¿Cómo llamar varias veces al programa? Hay dos soluciones. La primera es hacer otro programa en PERL que llame las veces necesarias al programa `prog`:

```
#!/usr/bin/perl

# Primera opcion para llamar
# al programa prog
‘prog  1 2 > sale‘;
‘prog  3 4 >>sale‘;
‘prog  5 6 >>sale‘;
‘prog  7 8 >>sale‘;
```

La salida de este programa se almacena en el archivo `sale` que se muestra a continuación.

```
1 + 2 = 3
3 + 4 = 7
5 + 6 = 11
7 + 8 = 15
```

5 Otra solución

Una solución más elegante para llamar repetidas veces al programa `prog` es crear un archivo con la lista de argumentos, en este caso serán pares de números que guardaremos en el archivo `data.txt`:

```
1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
17 18
```

Y creamos un programa en PERL, supongamos que se llama `p1`, que recibe como argumento el nombre del archivo con los datos. El programa `p1` quedaría como:

```
#!/usr/bin/perl
# Segunda opcion para llamar
# al programa prog

die "Sin argumento me lamento\n"
    if ( $#ARGV < 0 );

open( INPUT, "$ARGV[0]" ) or
    die "Can't open $ARGV[0] file\n";

if ( -e "sale" ) {
    unlink "sale";
}
while ( <INPUT> ) {
    chop;
    'prog $_ >> sale';
}
close INPUT;
```

Y la salida al ejecutar este programa como `p1 data.txt` queda guardada en el archivo `sale`. El contenido de `sale` es:

```
1 + 2 = 3
3 + 4 = 7
5 + 6 = 11
7 + 8 = 15
9 + 10 = 19
11 + 12 = 23
13 + 14 = 27
15 + 16 = 31
17 + 18 = 35
```

Y aquí vemos como PERL nos ayuda a pegar el programa `prog` con los datos en el archivo `data.txt`. Es la flexibilidad y el poder de la programación en un lenguaje de muy alto nivel.

Ejemplo de un programa en PERL para generar una imagen PGM

```
#!/usr/bin/perl

# Perl program to construct a test image.
# Output: test.pgm image.
# Input: It's necessary edit $rows and $cols
#        values by hand
#
# Could be much more easy to use the command:
# pgmramp -ellipse 256 256 > output_image.pgm
#
# (pgmramp is part of NETPBM package)
#
# Fraga April, 2001

$rows = 256;
$cols = 256;

if ( $rows > $cols ) {
    $center = ($cols - 1 )/2;
}
else {
    $center = ($rows - 1 )/2;
}

for ( $i=0; $i<$rows; $i++ ) {
    $di = $center - $i;
    $di2 = $di * $di;
    for ( $j=0; $j<$cols; $j++ ) {
        $dj = $center - $j;
        $dj2 = $dj * $dj;

        $val = sqrt ( $di2 +  $dj2 );
        # Putting image values
        $img[ $cols * $i + $j ] =255-int($val);
    }
}
$val = save_pgn_img(\@img,$rows,$cols, "test.pgm");
if ( !$val ) {
    printf "All's ok!\n";
}
else {
    print "You have problems...\n";
}

# Subroutine to save a image in ASCII PGM format
# (see pgm man pages on NETPBM package)
sub save_pgn_img (\@$$$) {
    # ( @img, $rows, $cols, $filename );
    local (*myimg) = shift(@_);
    my $mycols = shift(@_);
    my $myrows = shift(@_);
    my $filename = shift(@_);

    my $n = @myimg;
    printf "$n $mycols $myrows $filename\n";

    open ( MYFILE, ">$filename" ) or return 1;

    print MYFILE "P2\n";
    print MYFILE "# file: $filename\n";
    print MYFILE "$mycols $myrows\n";
    print MYFILE "255\n";

    for ( my $i=1; $i <= $myrows*$mycols; $i++ ) {
        printf( MYFILE "%3d ", $myimg[$i-1] );
        if ( !($i%17) ) {
            print MYFILE "\n";
        }
    }
    close MYFILE;
    return 0;
}
```