Perl

Practical Extract and Report Language

Continut:

Note:

1. Cele 4 paradoxuri Perl sunt evidentiate cu culoarea roz.
2. Un exemplu de aplicatie CGI scrisa in Perl de mine, se gaseste la adresa:
http://www.scs.ubbcluj.ro/~mr28602/serverstatus.cgi

1. Ce este Perl ?

Perl este un limbaj de programare orientat spre extragerea , prelucrarea si prezentarea informatiei. Este disponibil pe o multime de platforme: Unix, MVS, VMS, MsDos, Macintosh, OS/2, Amiga si altele. Perl contine functii foarte puternice pentru manipularea textelor. El combina facilitatile si scopurile multor limbaje de comnezi (scripting). Perl a cunoscut succesul de curand, fiind folosit in programarea World Wide Web, la crearea formularelor electronice si in general ca o legatura intre sisteme, baze de date si utilizatori.

2. Resurse Perl.

Acestea sunt cateva resurse pentru invatarea limbajului Perl.

web: www.cis.ufl.edu/perl
www.perl.oreilly.com
grupuri usenet: news:comp.lang.perl.anounce news:comp.lang.perl.misc
manual : Larry Wall, Tom Christiansen & Randall L. Schwartz, Programming Perl, 2nd Edition September 1996, 670 pages, O'Reilly and Associates, Inc., ISBN 1-56592-149-6, $39.95. The "Camel Book".

3. Utilizatori Perl.

Doua categorii de programatori indragesc Perl. Administratori de sistem, deoarece imbina foarte elegant comenzi sistem pentru manipularea datelor si proceselor, si are facilitati de cautare care usureaza cautarile si afisarea de informatie intr-un sistem. Dezvoltatorii de aplicatii Web pe servere unix, deoarece este mai usor de invatat decat C si ofera mai multe functii decat acesta, respectiv pentru validarea datelor si gestionarea de baze de date simple.

Codul Perl din acest document ruleaza sub Perl 4.036 (ultima versiune din seria 4) cat si sub Perl 5.0. Perl 5.0 adauga facilitati pentru programarea orientata obiect.

4. Stilul de programare Perl.

Multe programe utile scrise in Perl sunt scurte. Sa presupunem ca vrem sa schimbam acelasi text in mai multe fisiere. In loc sa editam toate fisierele sau sa construim niste comenzi criptice in find, awk, sau sed , putem scrie o simpla comanda:

```
perl -e 's/rosu/negru/gi' -p -i.bak *.html
```

Acesta comanda, tastata la un promp Unix, executa programul Perl scris intre ghilimele. Acest program executa o singura operatiune: inlocuieste cuvantul rosu cu cuvantul negru , global, intr-un mod case-insensitive. Restul liniei de comenzi specifica, ca rularea sa se faca pentru fiecare fisier cu extensia .html din directorul curent, iar daca vre-un fisier trebuie modificat (test.html) atunci se va crea o copie de siguranta cu extensia .bak (test.html.bak).

Pentru cei acomodati cu stilul de programare C sau Pascal, programul de mai sus poate fi expandat in forma urmatoare astfel incat sa urmareasca stilul de programare din cele doua limbaje de mai sus: (Pascal si C):

```
#!usr/local/bin/perl -w
#  File: schimb
# Program Perl pentru substituirea cuvantului rosu cu cuvantul negru in toate
# specificate in linia de comanda

$vechi = 'rosu';
$nou = 'negru;
$nrschimbari = 0;

# Separatorul de inregistrari in fisierele de intrare este definit de variabila
globala        # Perl: $/ .Poate fi orice sir de caractere. In mod normal este \n . Aici il
specificam     # ca fiind ca fiind null, deci intreg fisierul va fi citit ca un singur camp

undef $/;
```

```
# Presupunand ca programul a fost apelat cu parametrii schimb 1.html, 2.html,
# 3.html, atunci lista @ARGV va contine 3 elemente: ('1.html', '2.html, '3.html')
# Acestia pot fi accesati prin $ARGV[0], $ARGV[1], $ARGV[2]

foreach $file(@ARGV)
{
        if (! open(INPUT,"<$file"))
        {
                print STDERR "Nu pot deschide fisierul $bakfile \n";
                next;
        }

# Citesc fisierul de intrare ca si un singur camp de inregistrare
$data = <INPUT>;
close INPUT;

if ($data =~ s/$vechi/$nou/gi)
{
        $bakfile = "$file.bak";
        # Iesire daca nu pot salva fisierul de siguranta sau nu pot deschide fisierul
        # destinatie
        if (! rename($file, $bakfile))
        {
                die "Nu pot redenumi $file $!";
        }
        if (! open(OUTPUT,">$file"))
        {
                die "Nu pot deschide fisierul destinatie $file \n";
        }
        print OUTPUT  $data;
        close OUTPUT;
        print STDERR "$file inlocuit \n";
        $nrschimbari++;
}
else { print STDERR "$file nu a fost schimbat\n"; }
}
print STDERR "$nrschimbari fisiere inlocuite. \n";
exit(0);
```

Observam din programul de mai sus ca anumite elemente se aseamana cu limbajul C. De exemplu linia in linia !open(INPUT,...) semnul ! este operatorul boolean de negare si se foloseste identic ca si in C, orice valoare pozitiva este adevarata, iar orice valoare pozitiva cu ! in fata este fals, iar 0 cu ! in fata este adevarat. Totodata observam ca si constructia if...else este similara cu ceea din C.

In linia *$nrschimbari++* observam ca incrementarea unei variabile se face analog ca in C.

Filozofia Perl este :"There is more than one way", exista mai multe modalitati. Din acesta nobila libertate de exprimare rezulta prima din cele patru paradoxuri Perl: Programele Perl sunt usor de scris dar nu intotdeauna usor de citit. Pentru exemplificare, urmatoarele linii de cod Perl sunt echivalente.

```
if  ($x == 0) {$y = 10;} else {$y = 20;}
$y = $x == 0? 10 : 20;
$y = 20; $y = 10 if $x == 0;
unless ($x == 0) {$y = 0;} else {$y = 10;}
if ($x) {$y = 20;} else {$y = 10;}
$y = (10,20) [$x != 10];
```

Observam ca liniile 1, 2 si 5 sunt similare ca si sintaxa cu limbajul C. In multe aspecte Perl este similar cu C, dupa cum vom vedea si in cele ce urmeaza.

5. Stocarea si rularea programelor Perl.

Exemplul Hello World. Continutul fisierului *hello* este:

```
# !/usr/local/bin/perl -w
if( $#ARGV >= 0) {$who = join(' ', @ARGV); }
else {$who = 'World'; }
print "Hello, %who! \n";
```

Presupunem ca programul de mai sus este stocat in fisierul Unix ~/bin/hello. Programul poate fi rulat cu una din comenzile:

perl ~bin/hello
perl ~bin/hello oameni
perl hello (daca ne aflam in directorul ~/bin).

Pentru executarea acestui program ca si o comanda, trebuie parcursi urmatorii pasi:
Prima linie din program trebuie sa contina dupa #! calea spre comanda perl, asa cum s-a aratat si in exemplul de mai sus. Totodata in aceasta linie se pot specifica si optiuni de comanda, de exemplu -w (warnings - avertismente).
Pentru a permite citirea si executarea fisierului de catre toti utilizatorii trebuie introdusa comanda Unix:

```
chmod  a+rx ~/bin/hello
```

Se editeaza fisierul ~/.cshrc sau ~/.login pentru a aduga directorul bin in calea de cautare a fisierelor executabile. In aceste fisiere se adauga o linie in genul:

set path = ($path ~/bin)

Dupa acestea, programul se poate lansa in executie tastand *hello.*

6. Elemente Perl

Structurile de date in Perl.

Scalari pot valori numerice sau caractere, determinate de contextul in care apar.

Exemple:
123 12.4 5E-10 0xff (valoare hexazecimala) 0377(valoare octala)
' Eu sunt $nume si sunt in anul \n 2'
"Ce mai faci?" "Inlocuirea valorilor $x si \n in \" ghilimele."
`date` uptime -u`
$x $lista[5] $tabela('key')

Ghilimelele simple ' ' permit doar inlocuirea in textul cuprins intre ele a semnelor \\ si \. Ghilimelele duble " " permit inlocuirea in textul cuprins intre ele si a variabilelor gen $nume si a caracterelor de control gen \n (linie noua). Ghilimelele intoarse ` ` permit toate inlocuirile de mai sus , dupa care incearca sa execute sirul rezultat ca si o comanda sistem si intorc textul afisat de sistem ca urmare a executiei comezi reprezentate de sirul de caractere.

Sirurile de scalari (numite si liste) reprezinta scalari aranjati secvential.

Exemple:
('Luni', 'Marti', 'Miercuri', 'Joi', 'Vineri', 'Sambata', 'Duminica')
(1,2,3,4,5,6,7,8,9) echivalent cu (1..9)
(1,2,3,4,5,6,7) [2,4] echivalent cu (3,4,5)
@Lista

Sirurile asociative, ajuta la retinerea anumitor lucruri des folosite:

Exemple:
$ZileInLuna('Ianuarie') = 31; $Student('Muresan Robert') = 1;
$NumeStudent{28602} = 'Muresan Robert';
$Nota($NrStudent, $NrExamen) = 10;
%lista_intreaga

Perl 5 permite permite combinarea celor mai de sus, cum ar fi liste de liste sau siruri asociative de liste.

Conventii de notare in Perl.

Numele variabilelor scalare incepe cu $, chiar si atunci cand ne referim la un element dintr-o lista. Numele variabilelor care reprezinta liste incepe cu @, iar numele unei variabile care se refera la o lista asociativa incepe cu %.
Listele sunt indexate cu paranteze patrate [] si contin inchis intre ele un indice, indexarea incepand cu [0] (ca si in C). In Perl 5 indicii negativi inseamna indexare de la capatul listei.
De exemplu $Zile[5] este al saselea element al element al listei @Zile si

('Luni','Marti','Miercuri')[1] este egal cu 'Marti'

Listele asociative sunt indexate cu paranteze rotunde () care cuprind intre ele un sir de caractere.
$var, @var si % var sunt 3 variabile distincte.

```
@zile = (31,28,31,30,31,30,31,31,30,31,30,31);
                        # o lista cu 12 elemente
$#zile                  # Ultimul indice din lista zile; 11 in acest caz
$#zile = 7;             # Trunchiaza sau extinde lista zile la 7 elemente
@zile                   # ($zile[0], $zile[1],...)
@zile[3,4,5]            # = (30,31,30)
@zile('a', 'c')         # echivalent cu ($zile{'a'}, $zile{'c'})
%zile                   # (cheie1, valore1, cheie2, valoare2,...)
```

Limbajul Perl este case sensitive, astfel $VAR, $Var si $var reprezinta 3 variabile distincte.
Daca primul caracter dupa simbolurile $,%, sau @ este un caracter sau linii de despartire _ atunci restul numelui poate sa contina numere sau linii de despartire. Daca primul caracter este cifra atunci si restul numelui trebuie sa contina doar cifre. Perl are mai multe variabile ale caror prim caracter nu este alfanumeric. De exemplu $/ este separatorul de inregistrari. O variabila neinitializata are o valoare speciala nedefinit care poate fi aflata cu ajutorul functiei defined(). Valorile nedefinite se convertesc in functie de context in 0, null sau false.

Variabila $_ este folosita implicit de Perl cand o variabila necesara unei operatii nu a fost specificata. De exemplu:

```
<STDIN>;    # Atribuie o inregistrare din fisierul STDIN variabilei $_
print;      # Tipareste valoarea variabilei $_
chop;       # Elimina ultimul caracter din $_
@cuvinte = split; # Imparte sirul $_ in cuvinte despartite de spatii, care devin
# elemente succesive ale listei @cuvinte
```

Variabilele $_, $1, $2, $3 si alte variabile implicite folosite de Perl creaza al doilea paradox Perl: "What you don't see can't help you or hurt you", ceea ce nu poti vedea nu te poate ajuta sau ranii.

Functii si subrutine.

Functiile in Perl se declara cu simbolul & in fata, excpetie facand cazurile cand numele procedurii urmeaza dupa un cuvatn cheie cum ar fi sub.

Exemplu:

```
sub square{return $_[0] ** 2;}
print "5 la patrat este ", &square(5);
```

Numele handlerelor de fisiere nu incep cu caractere speciale, si pentru a nu intra in conflict cu alte cuvinte cheie, acestea sunt notate cu litere mari : INPUT, OUTPUT, STDIN, STDOUT, STDERR, etc..

7. Literali si Operatori

Exemple: Numere si Caractere

```
#!usr/bin/perl
print '013'  , 'este numar prim', 004, ' nu este numar prim.';
print 7 + 3,' ' , 7 - 3
$x = 7;
print $x
print 'Nu afiseaza variabila $x si linie noua \n .'
print 'Afiseaza $x si line noua \n'
$y = "Linie ce contin $x si se termina cu salt la linie noua \n."
print $y;
$y = "Con"."catena"."re";
print $y
```

Acest program va afisa:

013 este numar prim 4 nu este numar prim. 10 4 Nu afiseaza variabila $x si linie noua \n.Afiseaza 7 si linie noua
Linie ce contine 7 si se termina cu salt la linie noua
Concatenare

Observam ca instructiunea print nu sare la linie noua decat daca se specifica caracterul \n intre ghilimele "". Aceasta o greseala des intalnita in programele Perl. Totodata observam operatorul . (punct) care are ca efect concatenarea celor trei siruri de caractere.

Exemple: Comparatiile

O greseala des intalinta in programele Perl este confundarea operatorului de atribuire = cu operatorul de comparare numerica ==.

Totodata este recomandata folosirea operatorilor eq, ne, lt, gt pentru siruri de caracatere:

```
if ( $x eq 'unu')
{
        print 'Siruri egale.'
}
```

8. Bucle si sistemul I/O in Perl.

Exemplu: Parametrii in linia de comanda si bucle iterative

```
print "$#ARGV este indicele ultimului argument din linia de comanda.\n"

# Itereaza de la indicele 0 pana la indicele $#ARGV:
# Observam ca , constructia for este similara cu cea din C

for($i = 0; $i <= $#ARGV; $i++)
{
        print "Paramtrul $i este $ARGV[$i].\n";
}

# O alta varianta a buclei de mai sus

foreach $item(@ARGV)
{
        print "Cuvantul este: $item. \n";
}

# O varianta similara folosind de data asta variabila implicita Perl $_

foreach (@ARGV)
{
        print "Spun: $_ . \n";
}
```

Rularea programului va afisa:

```
> perl exemplu.pl Buna Dimineata, Elevi!
2 este indicele ultimului argument din linia de comanda
Parametrul 0 este Buna.
Parametrul 1 este Dimineata,.
Parametrul 3 este Elevi!
Cuvantul este: Buna.
Cuvantul este: Dimineata,.
Cuvantul este: Elevi!
Spun: Buna.
```

Spun: Dimineata,.
Spun: Elevi!.

Exemplu: Sistemul standard I/O

```
print STDOUT "Tastati un text: ";
while($input = <STDIN>)
{
        chop $input;
        print STDOUT "Ati tastat : $input \n"
        if ($input eq '') {print STDERR "Nu ati tastat nimic!\n"}
        print STDOUT "Mai tastati ceva sau apasati CTRL - D pentru terminare"
}
print STDOUT "Atat a fost."
```

Obs1. Conditia din directiva while este o directiva de asignare, asigneaza urmatoarea inregistrare de la intrarea standard, variabilei $input. La sfarsit de fisier acesta nu va asigna variabilei $input valoarea null, ci o valoare nedefinita, care in acest context este evaulata ca fiind null. Deci lini a while($input = <STDIN>) face trei lucruri: citeste o inregistrare, o asigneaza variabilei $input si verifica daca $input este nedefinita, in acest context ea fiind evaluata la valoarea null. In alte contexte, Perl evauleaza o variabila ca fiind zero sau null. De exemplu daca $i nu este definita, atunci $i++ va atribui variabilei $i valoare 1. De aici rezulta al treilea paradox Perl: Side effects can yield an elegant face or a pain in the rear. Efectele secundare pot conduce la lucruri elegante sau batai de cap.
Obs2. Campurile de date sunt in mod implicit delimitate de caracterul \n, care in exemplul de mai sus este inclus ca si ultimul caracter al variabilei $input. Functia chop elimina acest ultim caracter din variabila $input. In Perl 5 este introdusa o noua functie chomp care elimina ultimele caractere doar daca acestea sunt definite ca delimitatori de inregistrari, prin variabila globala $/.

Exemplu: Executarea unui string ca si porgram Perl

```
#!usr/bin/perl
for(;;){
print '(', join(', ',@Result),') ?';
last unless $Input = <STDIN>;
$? = ''; $@ = ''; $! = '';
@Result = eval $Input;
if ($?) {print 'status=', $?,' '}
if ($@) {print 'eroare=', $@,' '}
if($!) {print 'nreroare=',$!+0,': ',$!,' '}
}
```

Functia eval, evalueaza un sir de caractere interpretand-ul ca un program scris in Perl. $@ reprezinta mesajul de eroare rezultat din ultimul apel al functiei eval sau do.

Rularea programului va afisa:

>perl perls.pl
() ? **sqrt(2)**
(1.4142)

Exemplu: I/O cu fisiere

```
#! usr/bin/perl
# program pentru inversarea fiecarei linii dintr-un fisier.

# 1: Obtinerea numelui fisierelor din parametrii transmisi in linia de comanda

if($#ARGV != 1){
        die "Folosire: $0 fisier_sursa fisier_destinatie \n";
}
($infile,$outfile) = @ARGV;
if (! -r $infile){
        die "Nu pot citi fisierul de intrare $infile \n";

}

#2: Validarea fisierelor
# Daca prima parte a expresiei in cazul operatorului || (or) este adevarata, Perl
# nu mai evalueaza si restul continutului. Daca fisierul de intrare sa putut
# deschide atunci executia programului continua, altfel se evalueaza
instructiunea   # die si executia programului se incheie.

open($INPUT,"<$infile") || die "Nu pot deschide fisierul $infile ! \n";
if( -e $outfile){
        print STDERR "Fisierul $outfile exista!\n";
        until ($ans eq 'i' || $ans eq 'a' || $ans eq 'e'){
                print STDERR "Inlocuiesc, Adaugare, or iEsire?";
                $ans = getc(STDIN);
        }
        if ($ans eq 'e') {exit}
}

if ($ans eq 'a') {$mode = '>>'}
else {$mode = '>'}

open(OUTPUT,"$mode$outfile") || die "Nu pot deschide fisierul destinatie!";

#3: Citeste fiecare linie din fisierul sursa, inverseaza linia si o scrie in fisierul
# destinatie
```

```
while(<INPUT>){
        chop $_;
        $_ = reverse $_;
        print OUTPUT $_,"\n";
}

#4: Terminare

close INPUT, OUTPUT;
exit;
```

10. Procesare de date.

Exemplu: Tabel cu rezultatele studentilor.

Programul produce un raport cu rezultatele studentilor la examene, combinand date din doua fisiere ce contin informatii despre studenti respectiv notele obtinute de fiecare la examene.

Fisierul de intrare *studenti* are campurile delimitate prin : si contine pe fiecare linie numarul matricol, numele si anul in care este studentul.

28602:Muresan Robert:2
38601:Pop Ioan:3
48000:Pop Vasile:1

Fisierul de intrare *note* are campurile delimitate prin spatii si contine pe fiecare linie numarul matricol, examenul, si numarul de credite obtinute.

28602 1 4
38601 1 2
48000 1 1
28602 2 5
38601 2 3

Se observa ca Pop Vasile a lipsit la al 3-lea examen.

Raportul afisat de program arata in felul urmator:

Nr. Mat	Nume	1	2	Total
28602	Muresan Robert	4	5	9
38601	Pop Ioan	2	3	5
48000	Pop Vasile	1		1
	Total:	7	8	

Programul care a generat acest raport este urmatorul:

```
#! usr/bin/perl
# Catalog de Note - pentru demonstrarea lucrului cu sistemul I/O, liste asociative
# sortare si formatarea unui raport
```

```
# Programul accepta orice numar de studenti si examene, respectiv trateaza si
# cazurile cand anumite informatii lipsesc.
$studfile = 'studenti';
$notfile = 'note';

# Daca fisierele se pot deschide, atunci perl nu mai evalueaza si restul expresiei
ce      # urmeaza dupa ||

open (NUME,"<studfile") || die "Nu pot deschide fisierul $studfile! \n";
open(NOTE,"<notfile") || die "Nu pot deschide fisierul $notfile! \n";

# Construieste o lista asociativa cu informatiile despre studenti folosind ca si
# cheie, numarul matricol

while(<NUME>){
        ($nrmat,$nume,$an) = split(':',$_);
        $nume{$nrmat} = $nume;
        if (length($nume) > $maxlengthnume){
                $maxlengthnume = length($nume);
        }
}
close NUME;

# Construieste un tabel cu notele de la exemene

while(<NOTE>){
        {$nrmat,$nrexam,$nota} = split;
        $nota{$nrmat,$nrexam} = $nota;
        if ($nrexam > $maxnrexam){$maxnrexam = $nrexam;}
}
close NOTE;

# Tipareste raportul cu datele citite

printf "%6s %-${maxlengthnume}s ",'Nr. Mat','Nume';
foreach $nrexam(1..$maxnrexam){
        printf "%4d",$nrexam;
}
printf "%10s\n\n",'Total:';

#  Subrutina dupanume este folosita pentru sortarea sirului %nume
#  Functia "sort" transmite variabilele $a si $b subrutinelor pe care le apeleaza
# Functia " x cmp y " returneaza  -1 daca x < y , 0 daca x=y si 1 daca x > y.

sub dupanume {$nume{$a} cmp $nume{$b}}
```

```
# Ordoneaza numerele matricole astfel incat numele studentilor sa apara in
# ordine alfabetica
foreach $nrmat(sort dupanume keys(%nume)){
        # Tipareste creditele obtinute si totalul pentru fiecare student
        printf "%6d %-${maxlengthnume}s ", $nrmat,$nume{$nrmat};
        $total = 0;
        foreach $nrexam(1..$maxnrexam){
                printf "%4s",$nota{$nrmat,$nrexam};
                $total += $nota($nrmat,$nrexam);
                $examtot($nrexam) += $nota($nrmat,$nrexam);
        }
        printf "%10d\n",$total;
}

printf "\n%6s %${maxlengthnume}s ",' ',"Total: ";
foreach $nrexam(1..$maxnrexam){
        printf "%4d",$examtot{$nrexam};
}
printf "\n";
exit(0).
```

11. I/O folosind Pipe si Apelarea functiilor sistem.

Exemplu: Afisarea spatiului ocupat de fisiere pe disc.

```
#! usr/bin/perl
# Afisarea spatiului ocupat de fisierele specificate
# Acest program apeleaza functia sistem (Unix) du din care obtine numele
# fisierului si numarul de bytes ocupati, si reuneste aceste informatii cu alte
# informatii utile

$files = join(' ',@ARGV);

# Semnul | redirecteaza datele afisate de du catre programul nostru

if(! open(DUPIPE,"du -sk $files | sort -nr |")){
        die "Nu pot rula du! $!\n";
}
printf "%8s %-8s %-16s %8s %s\n",'K-Bytes','Login','Nume','Modificat','Fisier';

while (<DUPIPE>){
        # proceseaza informatiile afisate de du
        ($kbytes,$filename) = split;
```

```
        # apel sistem pentru aflarea mai multor informatii despre fisier
        ($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,$atime,$mtime,$ctime) =
    stat($filename);

        #apel sistem pentru a asocia  login si nume cu uid
        if($uid != $previous_uid){
            ($login,$passwd,$uid,$gid,$quota,$coment,$realname,$dir,$shell)
        = getpwuid($uid);
            $realname = split(', ',substr($realname,0,20));
            $previous_uid = $uid;
        }

        # Converteste timpul ultimei modificari intr-o forma mai usor de citit
        ($sec,$min,$hour,$mday,$mon,%myear) = localtime($mtime);
        $mmonth = $mon + 1;

        printf "%8s %-8s %-16s %02s-%02d-%02d %s\n",$kbytes,
    $login,$realname,$myear,$mmonth,$mday,$filename;
    }
```

Exemplu de rezultat afisat de program:

```
K-bytes Login       Nume              Modificat Fisier
 12345  mr28602   Muresan Robert  99-10-10   abc.txt
```

12. Cautare.

Cautarea implica folosirea unor sabloane numite *expresii regulare.* Dupa cum se va vedea, aceasta da nastere la al patrulea paradox Perl: Regular expresion aren't. Expresiile regulare nu sunt(regulare).

Operatorul =~ efectueaza cautarea dupa un sablon si inlocuirea. de exemplu, daca

```
$s = 'Unu doi trei patru";
```

atunci

```
if ($s =~/doi trei/) {print YES} else {print NO}
```

va tipari YES, deoarece sirul de caractere $s se potriveste cu sablonul "doi trei"

```
if($s =~/unu/){print YES} else {print NO}
```

va tipari NO deoarece sirul nu se potriveste cu sablonul. Adaugand insa optiunea i pentru a ignora diferenta intre caracterele mari si caracterele mici, atunci urmatoarele linii vor afisa YES.

```
if($s =~ /one/i) {print YES} else {print NO}
```

Sabloanele pot contine o serie larga de optiuni pentru a face cautarea cat mai flexibila si generala.De exemplu un .(punct) se potriveste cu orice caracter, exceptand caracterul \n.

```
if($s =~ /l.mp/){print YES}
```

va tipari YES pentru $s = "lamp", "lump", "slumped", dar nu si pentru $s = "lmp",sau "less amperes".
Parantezele () grupeaza elementele sablonului. Un asterisk * inseamna ca, caracterul, elementul, sau grupul de elemente anterior poate sa apara de mai multe ori sau deloc.Simila plus + inseamna ca elementul sau grupul de elemente anterior trebuie sa apara cel putin odata.Semnul intrebarii se potriveste cu o singura aparitie sau nici o aparitie.Exemple:

*/fr.*nd/ se potriveste cu "frnd", "friend", "front and back"*
/fr.+nd/ se potriveste cu "frond", "friend", "front and back"
dar nu se potriveste cu "frnd".
*/10*1/ se potriveste cu "11", "101", "1001", "100000001".*
*/b(an)*a/ se potriveste cu "ba", "bana", "banana", "bananananana"*
/flo?at/ se potriveste cu "flat" and "float"
dar nu se potriveste cu "flooat"

Parantezele patrate[] se potrivesc cu un singur caracter definit de ele. De exemplu:

[0123456789] se potriveste cu orice cifra
[0-9] se potriveste cu orice cifra
[0-9]+ se potriveste cu orice secventa de cifre
[a-z]+ se potriveste cu orice cuvant scris cu litere mici
[A-Z]+ se potriveste cu orice cuvant scris cu litere mari
[ab n] se potriveste cu sirul null "", sirul "b", orice numar de spatii,"banana an nnna"*
[^...] se potriveste cu orice caracter care nu este "..."
[^0-9] se potriveste cu orice caracter care nu este cifra

Acoladele permit o specificare mult mai precisa a sirurilor care se repeta. Astfel [0-9]{6} se potriveste cu orice sir de cifre de lungime 6, iar [0-9]{6,10} se potriveste cu orice sir de cifre care are lungime cuprinsa intre 6 si 10.
Sabloanele pot sa apara oriunde in sirul de caractere daca nu sunt ancorate.Semnul ^ pus in afara parantezelor patrate[], ancoreaza sablonul la inceputul sirului de caractere, iar caracterul $ ancoreaza sablonul la sfarsitul sirului de caractere.De exemplu:

/at/ se potriveste cu "at", "attention", "flat", si "flatter"
/^at/ se potriveste cu "at" si "attention" dar nu si cu "flat"
/at$/ se potiveste cu "at" si "flat", dar nu si cu "attention"
/^at$/ se potriveste doar cu "at".
/^at$/i se potriveste cu "at", "At", "aT", si "AT".
/^[\t]$/ se potriveste cu o linie goala, sau cu o linie ce contine oreice combinatie de spatii libere sau taburi*

Caracterul Backslash. In general caracterele se potrivesc cu ele insele, cu exceptie caracterelor speciale de control cum ar fi:+ ? . [] () {}. Acestea trebuie prefixate cu un backslash \ pentru a fi considerate ca si caractere.
Bara verticala | este operatorul binar sau. Exemplu:

```
if ($answer =~/^y | ^yes |^yeah/i){
        print "Afirmativ";
}
```

va afisa "Afirmativ" pentru: y, yes, Yes, Yeah, Yeah rigth,etc...

13.Despartire in cuvinte (parsing).

Cand se includ paranteze rotunde in sabloanele de cautare, secventele cu care se potrivesc acestea, pot fi accesate folosind variabilele $1,$2,$3, etc.. in ordine de la prima paranteza din stanga. Aceste secvente pot fi asignate ca si elemente succesive ale unui sir. Exemplu:

```
#! usr/bin/perl -w
$s = 'Acest sir contine 1 subsir de tip data 10/10/99 undeva.';

# sirul /d semnifica un sir de cifre
$s =~ /(\d{1,2})\/(\d{1,2})\/(\d{2,4})/;

# varianta 1:
print "\$1 = $1, \$2 = $2, \$3 = $3\n";

# varianta 2:
{$ziua, $luna, $an} = ($s =~ /(\d{1,2})\/(\d{1,2})\/(\d{2,4})/);
print "$ziua, $luna, $an\n";

#varianta 3:
{$data,$ziua, $luna, $an} = ($s =~ /(\d{1,2})\/(\d{1,2})\/(\d{2,4})/);
print "$data, $ziua, $luna, $an\n";
```

14. Utilizarea Perl in scrierea de programe CGI.

Limbajul Perl poate fi folosit cu succes in scrierea de aplicatii CGI (Common Gateway Interface), acestea fiind folosite pe serverele de Web. Un program CGI ruleaza pe platforma Web Serverului , astfel chiar daca WebServerul este o masina Unix si utilizatorul acceseaza programul CGI de pe o platforma Windows, aplicatia CGI se poate rula fara probleme pe masina Unix, iar rezultatele afisate de aceasta sunt afisate in browserul utilizatorului.

Exemplu: Afisarea starii serverului:

Aplicatia foloseste comenzile unix 'uptime', 'hostname', si 'w'.
Pentru a vedea rezultatele acestui program, vizitati adresa:
http://www.scs.ubbcluj.ro/~mr28602/serverstatus.cgi

Programul este:

```
#! usr/bin/perl
# Trimite mesajele de eroare catre utilizator, nu catre sistem
open(STDERR,'<&STDOUT'); $| = 1;

# Programul va furniza browserului, un fisier html
print "Content-type: text/html\n\n";
$host = `hostname`; chop $host;
$uptime = `uptime`;
$w = `w -s -h`;
print <<BUNCHASTUFF;
<Html><Head>
<H1>Informatii despre serverul $host</H1>
</Head><Body>
$uptime
<PRE>$w</PRE>
<HR>
</Body></Html>
BUNCHASTUFF;
exit;
```

Pentru folosirea aplicatiei este necesar sa setam fisierului serverstatus.cgi, drepturi de acces, citire, si executi cu comanda Unix:

chmod a+rx serverstatus.cgi