Operatori

• operator povezivanja -> “hello” • “world” # “helloworld”

x operator ponavljanja niza -> “fred” x 3 # “fredfredfred”

5 x 4 # “5555”

operatori usporedbe brojeva: < <= == >= > != vraćaju true ili false

operatori usporedbe znakovnih nizova: lt le eq ge gt ne

brojevi	znakovi
==	eq
!=	ne
<	lt
>	gt
<=	le
>=	ge

Komentari

ukoliko se stavi # komentar je sve šta se nalazi iza znaka # no samo u tom retku

Ugrađena upozorenja

Ukoliko se pri pokretanja opcija –w (perl –w program.txt) -> na zaslon se ispisuju upozorenja ukoliko postoje

Upozorenja se mogu uključiti ukoliko se u zaglavlje programa isto stavi –w -> #!usr/bin/perl -w

Varijable

($) skalarna varijabla -> imena skalarnih varijabli počinju znakom $, no prvi znak iza $ ne može biti znamenka, SKALARNE VARIJABLE U PERLU SE UVIJEK REFERENCIRAJU PREDZNAĆENE ZNAKOM “$”!!!!!

stringovi se stavljaju unutar ‘’ ili “”

da bi u niz uključili $ moramo staviti \ prije njega

```
$what = "brontosaurus steak";
$n = 3;
print "fred ate $n $whats.\n"; # not the steaks, but the value of $whats
print "fred ate $n ${what}s.\n"; # now uses $what
print "fred ate $n $what" . "s.\n"; # another way to do it
print 'fred ate ' . $n . ' ' . $what . "s.\n"; # an especially difficult way
```

Interpolacija varijabli -> ${}

Ukoliko želimo dobiti širenje varijabli određenu varijablu stavljamo unutar {} –> ${varijabla}

```
$what = "brontosaurus steak";
$n = 3;
print "fred ate $n ${what}s.\n"; # rezultat je = fred ate 3 brontosaurus steaks.
```

Boolean :

1. nedefinirana vrijednost odgovara false

2. nula odgovara logičkoj vrijednosti false

3. prazan niz (‘ ‘) je false

4. iznimka je niz ‘0’ -> false

Učitavanje podataka

podatci s tipkovnice mogu se učitati naredbom <STDIN>

Funkcija chomp

- operator chomp uklanja oznaku kraja reda iz znakovnog niza
- chomp($text = <STDIN>); -> učitavanje texta bez \n, može se upotrijebiti bez zagrada -> chomp $text, također može vratiti broj uklonjenih znakova $broj = chomp $text vraća 1

Petlje -> vitičaste zagrade su obavezne!!!!!!!!!!!!

If -> može se koristiti I u obliku if->elsif kao u c-u

```
$line = <STDIN>;
if ($line eq "\n") {
print "That was just a blank line!\n";
} else {
print "That line of input was: $line";
}
```

While

```
$count = 0;
while ($count < 10) {
$count += 2;
print "count is now $count\n"; # Gives values 2 4 6 8 10
}
```

defined -> koristi se kad se želi ispitati je li neka varijabla definirana ili ne

```
$madonna = <STDIN>;
if ( defined($madonna) ) {
print "The input was $madonna";
} else {

print "No input available!\n"; }
```

Polja

lista je uređeni niz skalarnih vrijednosti (skup podataka)

polje je varijabla koja sadrži listu (varijabla)

elementi polja su indeksirani slijednim cijelim brojevima počevši od 0:

$fred[0] = “yabba”;

$fred[1] = “dabba do”;

index može biti bilo koji izraz koji daje numeričku vrijednost no ako nije cjelobrojna reducira se na cjelobrojnu, ako se pohrani broj iza kraja polje se automatski proširuje

$polje[0] = ‘nula’;

$polje[1] = ‘jedan’;

$polje[33] = ‘dva’; -> nastaje 31 nedefinirani element

index zadnjeg elementa se može dobiti s (#) -> $#polje

$end = $#polje; -> trenutno 33

$#polje = 5; -> polje se smanjilo na 5 elemenata ostali su izgubljeni

index -1 -> $polje[-1] označava posljednji element

$prvi = $polje[-6]; -> nulti element u polju od njih 6

Liste

liste u program se navode kao niz vrijednosti odvojenih zarezima unutar oblih zagrada

@ime_liste = (1, 2, 3)

(1, 2, 3,) -> isto zarez se zanemaruje

() prazna lista

(1..5) -> lista brojeva (1, 2, 3, 4, 5)

(1.7..5.7)-> isto jer se brojevi pretvaraju u integer

(5..1) -> ne ide unatrag pa stoga je to prazna lista

(0, 2..6, 10) -> (0, 2, 3, 4, 5, 6, 10)

($a..$b) -> lista brojeva od a do b

(0..$#polje) -> lista indexa polja

push i pop || shift i unsift -> vezano za liste

pop -> uzima zadnji element polja

@polje = 5..9;

$tra = pop(@polje); -> polje je sada (5, 6, 7, 8);

$tra je = 9

može se koristiti I bez zagrada ukoliko nije dvosmisleno

push-> dodaje se novi element na kraj polja

push (@polje, 0) -> na kraj se dodaje 0

push @polje, 8; -> na kraj se dodaje 8

push @polje, 1..10 -> dodaje se još 10 elemenata

shift -> oduzima prvi član polja

unshift -> dodaje prvi član polja

Split/Join Funkcije

Split(/separator/, "polje/niz") -> razdvaja zadano polje/niz prema zadanom separatoru

Join $glue, $pieces -> spaja zadane komadiće sa ljepilom

$moje = join ":", 1, 2, 3, 4, 5; -> daje 1:2:3:4:5

Kratica qw

omogućuje zapisivanje liste riječi bez potrebe za navodnicima

qw/ fred barney Wilma / -> lista imena

perl dozvoljava izbor proizvoljnog graničnika za naredbu qw:

!!, ##, (), {}, [], <>

qw{

/usr/dic/word

/home/rootbeer/

}

Pridruživanje listi

($fred, $barney) = (“jedan”, “dva”);

($fred, $barney) = ($barney, $fred); -> zamjena vrijednosti

($fred, $barney) = (qw/ jedan dva/);

($wilma, $dina) = qw[fred] -> dino je undef

Referenciranje polja (@)

za referenciranje koristi se @:

@polje = qw /jedan dva tri/;

@tiny = ( ); -> prazna lista

@giant = 1e5 -> lista od 100 000 elemenata

@stuff = (@giant, undef, @giant); -> 200 001 element

polje može sadržavati samo scalar, ne i polje!!!!!!!!!!

@kopija = @polje; -> polje se kopira

Broj članova polja -> $#polje -> dodaje se znak # između naziva i znaka $

Petlja foreach

prolazi kroz sve vrijendosti u polju te izvršava blok naredbi za svaku od vrijednosti

ako se unutar petlje mijenja upravljačka varijabla, mijenja se element polja

```
@polje = qw \a b c d\;
foreach $element (@polje) {
print “$element”;
print “\n”;
}
```

ispis:

a

b

d

c

podrazumijevana varijabla: ukoliko se iza $ zaboravi ime varijable automatski se stavlja $_

Operator reverse

naredba reverse preuzima polje vrijednosti I vraća polje poredano od kraja prema početku (obrnuto)

```
@polje = 1..10;

@pom = reverse(@polje);

foreach $p(@pom) {

print "$p";

}
```

ispis: 10987654321

Operator sort

vraća polje sortirano po abecedi (ASCII) – velika slova prije malih, brojevi prije slova…

```
@polje = (4, 5, 3, 6, 3, 2, 3);

@pom = sort(@polje);

foreach $p(@pom) {

print "$p";

}
```

ispis: 233456

Forsiranje skalarnog konteksta

da bi se dobio broj umjesto cijelog polja stavlja se funkcija **scalar**, daje perlu uputu da koristi broj a ne riječi -> “, scalar (polje) ,”

```
@lines = qw ! talc wuartz jade obsidian !;

print "Ja imam ",scalar @lines," kamena.\n";
```

ispis: Ja imam 4 kamena.

Regularni izrazi

Regularni izrazi služe za prepoznavanje uzorka unutar riječi/rečenica

Korištenje jednostavnih izraza

Kako bi se našao traženi uzorak unutar polje $_ navedeni uzorak se stavlja unutar // zagrada

```
$_ = “yabba dabba do”;

Print “našao sam” if /abba/;
```

Ukoliko se želi tražiti više stvari može se navesti u zagradama npr. Traži se ime(jedan tab udaljeno) prezime u uvjet se stavlja /ime\tprezime/

Kvantifikatori

Postoje I posebni znakovi koji olakšavaju traženje određenih riječi/rečenica

Točka (.)

se koristi kako bi se našao izraz kojemu se ne zna jedan character osim praznine/razmanak

Ako se stavi /d.rko/ gleda se unutar niza za sve šta počinje na d iza njega ima neki character osim oznake za novi red (to ne vrijedi) te završava na rko -> dorko, darko, d=rko (svi takvi izrazi) -> TOČKA NADOMJEŠTA JEDAN KARAKTER NE VIŠE!!!!!!!) Ukoliko se točka želi pronaći unutar niza stavlja se prije nje oznaka backslash znači (d\.rko) -> Traži izraz d\.rko A NE VARIJACIJE PRIKAZANE GORE

Zvjezda (*)

Označava da li se izraz pojavio nula Ili više puta

/ja\t*ti/ će tražiti da li se unutar niza između jati nalazi jedan, dva, nula ili više tabova -> “jati”, “ja\t\t\t\tti”, ako se stavi .* tražiti će se točka unutar unesenog niza makar se niti jednom pojavila

Plus (+)

Označava da li se prethodni element pojavio jedan ili više puta

/fred +barney/ Traži da li su fred I barney odvojeni samo razmacima

Upitnik (?)

On znači da se idući element može I ne mora nalaziti unutar niza

/bamm-?bamm/ -> bamm-bamm, bammbamm znači da se crtica (-) može I ne mora pojaviti

Grupiranje u predloške

Kontrolni znakovi (\cchar)

Odgovaraju ASCII vrijednosti do 32 tj. Svi ASCII brojevi do 32

Npr. \cH -> control+H (ASCII backspace znak)

Klasa […]/[^…]

Klasa znakova […] te negirana klasa znakova [^…] dozvoljava nizanje znakova koje želimo/ne želimo pronaći -> [a-z] -> sva mala slova od a do z

[^\d] -> ne želimo broj tj. Sve osim broja

Kratice \w, \d, \s, \W, \D, \S

Kratice za znakovni karakter, broj te karakterne klase. Word karakter je često ASCII alfanumerički karakter + "_", no lista alfanumeričkih znakova može uključiti dodatne znakove ovisno o implementaciji

POSIX karakterne klasa [:alnum:]

Primjer -> [:lower:] -> kada se piše kao [[:lower:]] -> jednako je kao [a-z] u ASCII

Lista POSIX klasa

Alnum	Letters and digits.
Alpha	Letters.
Blank	Space or tab only.
Cntrl	Control characters.
Digit	Decimal digits.
Graph	Printing characters, excluding space.
Lower	Lowercase letters.
Print	Printing characters, including space.
Punct	Printing characters, excluding letters and digits.
Space	Whitespace.
Upper	Uppercase letters.
Xdigit	Hexadecimal digits.

Grupiranje pomoću m//

Ukoliko želimo jednostavno pronaći neki točni izraz (znači znamo šta točno želimo točno koji izraz) tada možemo koristiti grupiranje pomoću (m -> ono šta želimo pronaći možemo staviti unutar bilo kakvih zagrada ili znakova kao kod qw -> || <> || ^^ [] {} // !! ,, () %% ## itd...

Npr. Ukoliko želimo pronaći sve izraze koji počinju sa http:// inače bi tražili ovako: /http:\/\// no puno lakše je pomoću m%http://%

Sidra

Odgovaraju pozicioniranju unutar ulaznog niza

Početak linije/stringa: ^, \A

Odgovara početku od kuda se traži tekst, in multiline mode ^matches after any newline. Some implementations support \A, which matches only at the beginning of the text.

End of line/string: $, \Z, \z

$ matches at the end of a string. In multiline mode, $ matches before any newline. When supported, \Z matches the end of string or the point before a string-ending newline, regardless of match mode. Some implementations also provide \z, which matches only the end of the string, regardless of newlines.

Start of match: \G

In iterative matching, \G matches the position where the previous match ended. Often, this spot is reset to the beginning of a string on a failed match.

Podudaranje riječi: \b, \B, \<, \>

Word boundary metacharacters match a location where a word character is next to a nonword character. \b often specifies a word boundary location, and \B often specifies a not-word-boundary location. Some implementations provide separate metasequences for start- and end-of-word boundaries, often \< and \>.

Binding operator (=~)

Operator koji kaže usporedi desnu stranu sa stringom na lijevoj, vraća da ili ne

```
my $some_other = "I dream of betty rubble.";
if ($some_other =~ /\brub/) {
print "Aye, there's the rub.\n";
}
```

Grouping, capturing, conditionals, and control

Capturing and grouping parentheses: (...) and \1, \2, etc.

zagrade () označuju dvije funkcije:

1. Grupiranje
2. Hvatanje

Parentheses perform two functions: grouping and capturing. Text matched by the subpattern within parentheses is captured for later use. Capturing parentheses are numbered by counting their opening parentheses from the left. If backreferences are available, the submatch can be referred to later in the same match with \1, \2, etc.

Ukoliko se unutar zagrada stavi regularni izraz nakon pronalaska traženog rezultat se sprema u varijable koje su tipa $(broj) -> broj ovisi o količini pronađenih različitih uzoraka (tj. Ako je pronađeno 4 slična uzorka oni će biti pohranjeni u varijable $1, $2, $3, $4 znači svaki će imati svoju varijablu)

Recepti

Ukoliko se stavi {broj} to označuje koliko se puta taj znak/brojke ponavlja tj. koliko ih ima ili ako se stavi {min,max} -> da li se ponavlja minimalno min ili maksimalo max puta

Removing leading and trailing whitespace

s/^\s+//
s/\s+$//
Matches: " foo bar ", "foo "
Nonmatches: "foo bar"

Numbers from 0 to 999999

/^\d{1,6}$/ -> kreće se od početka do kraja niza (^i $ oznake), limit polja je 6 ({6})
Matches: 42, 678234
Nonmatches: 10,000

Valid HTML Hex code

/^#([a-fA-F0-9]){3}(([a-fA-F0-9]){3})?$/ -> kreće se od početka do kraja stringa te se može dogoditi da prvih skupinu znakova ima max 3 te I drugih ima max 3 {3}
Matches: #fff, #1a1, #996633
Nonmatches: #ff, FFFFFF –> ne mogući slučajevi jer minimalno može biti 3, a max 6 znamenki

U.S. Social Security number

/^\d{3}-\d{2}-\d{4}$/ -> prvi broj ima 3 znamenke, drugi 2, treći 4 te su odvojeni minusom
Matches: 078-05-1120
Nonmatches: 078051120, 1234-12-12

U.S. zip code

/^\d{5}(-\d{4})?$/ -> prvi broj ima 5 znamenaka, drugi 4, a iza tih 4 se može I ne mora nalaziti nastavak (“?”)
Matches: 94941-3232, 10024
Nonmatches: 949413232

U.S. currency

/^\$\(d{1,3}(\,\d{3})*|\d+)(\.\d{2})?$/
Matches: $20, $15,000.01
Nonmatches: $1.001, $.99

Match date: MM/DD/YYYY HH:MM:SS

/^\d\d\/\d\d\/\d\d\d\d \d\d:\d\d:\d\d$/
Matches: 04/30/1978 20:45:38
Nonmatches: 4/30/1978 20:45:38, 4/30/78

Leading pathname

/^.*\//
Matches: /usr/local/bin/apachectl
Nonmatches: C:\\System\foo.exe

Dotted Quad IP address

/^(\d|[01]?\d\d|2[0-4]\d|25[0-5])\.(\d|[01]?\d\d|2[0-4]
\d|25[0-5])\.
(\d|[01]?\d\d|2[0-4]\d|25[0-5])\.(\d|[01]?\d\d|2[0-4]
\d|25[0-5])$/

Matches: 127.0.0.1, 224.22.5.110

Nonmatches: 127.1

MAC address

/^([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2}$/

Matches: 01:23:45:67:89:ab

Nonmatches: 01:23:45, 0123456789ab

Email

/^[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z_+])*@([0-9a-zA-Z][-\w]*
[0-9a-zA-Z]\.)+[a-zA-Z]{2,9}$/

Matches: *tony@example.com*, *tony@i-e.com,* tony@mail.
example.museum

Nonmatches: .@example.com, *tony@i-.com*, *tony@example.a*

HTTP URL

/(https?):\/\/([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+
[a-zA-Z]{2,9})
(:\d{1,4})?([-\w\/#~:.?+=&%@~]*)/

Matches: *https://example.com*, http://foo.com:8080/bar.html

Nonmatches: *ftp://foo.com*, *ftp://foo.com/*

Supported Metacharacters

\a	Alert (bell).
\b	Backspace; supported only in character class (outside of character class matches a word boundary).
\e	Esc character, x1B.
\n	Newline; x0A on Unix and Windows, x0D on Mac OS 9.
\r	Carriage return; x0D on Unix and Windows, x0A on Mac OS 9.
\f	Form feed, x0C.
\t	Horizontal tab, x09.
octal	Character specified by a two- or three-digit octal code.
\x*hex*	Character specified by a one- or two-digit hexadecimal code.
\x{*hex*}	Character specified by any hexadecimal code.
\c*char*	Named control character.
\N{*name*}	A named character specified in the Unicode standard or listed in PATH_TO_PERLLIB/unicode/Names.txt; requires use charnames ':full'.
[...]	A single character listed, or contained in a listed range.
[^...]	A single character not listed, and not contained within a listed range.
[:*class*:]	POSIX-style character class valid only within a regex character class.
.	Any character except newline (unless single-line mode, /s).
\C	One byte; however, this may corrupt a Unicode character stream.
\X	Base character, followed by any number of Unicode combining characters.
\w	Word character, \p{IsWord}.
\W	Nonword character, \P{IsWord}.
\d	Digit character, \p{IsDigit}.
\D	Nondigit character, \P{IsDigit}.
\s	Whitespace character, \p{IsSpace}.
\S	Nonwhitespace character, \P{IsSpace}.
\p{*prop*}	Character contained by given Unicode property, script, or block.
\P{*prop*}	Character not contained by given Unicode property, script, or block.
^	Start of string, or, in multiline match mode (/m), the position after any newline.
\A	Start of search string, in all match modes.
$	End of search string or the point before a string-ending newline, or, in multiline match mode (/m), the position before any newline.
\Z	End of string, or the point before a string-ending newline, in any match mode.
\z	End of string, in any match mode.
\G	Beginning of current search.
\b	Word boundary.
\B	Not-word-boundary.
(?=...)	Positive lookahead.
(?!...)	Negative lookahead.
(?<=...)	Positive lookbehind; fixed-length only.
(?<!...)	Negative lookbehind; fixed-length only.
/i	Case-insensitive matching.
/m	^ and $ match next to embedded \n.
/s	Dot (.) matches newline.
/x	Ignore whitespace, and allow comments (#) in pattern.
/o	Compile pattern only once.
(?*mode*)	Turn listed modes (one or more of xsmi) on for the rest of the subexpression.
(?-*mode*)	Turn listed modes (one or more of xsmi) off for the rest of the subexpression.
(?*mode*:...)	Turn listed modes (one or more of xsmi) on within parentheses.
(?-*mode*:...)	Turn listed modes (one or more of xsmi) off within parentheses.
(?#...)	Treat substring as a comment.
#...	Treat rest of line as a comment in /x mode.
\u	Force next character to uppercase.
\l	Force next character to lowercase.
\U	Force all following characters to uppercase.
\L	Force all following characters to lowercase.
\Q	Quote all following regex metacharacters.

\E	End a span started with \U, \L, or \Q.
(...)	Group subpattern and capture submatch into \1,\2,...and $1, $2,....
n	Contains text matched by the *n*th capture group.
(?:...)	Groups subpattern, but does not capture submatch.
(?>...)	Atomic grouping.
...|...	Try subpatterns in alternation.
*	Match 0 or more times.
+	Match 1 or more times.
?	Match 1 or 0 times.
{*n*}	Match exactly *n* times.
{*n*,}	Match at least *n* times.
{*x*,*y*}	Match at least *x* times, but no more than *y* times.
*?	Match 0 or more times, but as few times as possible.
+?	Match 1 or more times, but as few times as possible.
??	Match 0 or 1 times, but as few times as possible.
{*n*,}?	Match at least *n* times, but as few times as possible.
{*x*,*y*}?	Match at least *x* times, and no more than *y* times, but as few times as possible.
(?(*COND)...|...)*	Match with if-then-else pattern, where *COND* is an integer referring to a backreference, or a lookaround assertion.
(?(*COND*)...)	Match with if-then pattern.
(?{*CODE*})	Execute embedded Perl code.
(??{*CODE*})	Match regex from embedded Perl code.
(?<name>...)or(?'name'...)	Named capture group.
\k<*name*> or \k'*name*'	Backreference to named capture group.
%+	Hash reference to the leftmost capture of a given name, $+{foo}.
%-	Hash reference to an array of all captures of a given name, $-{foo}[0].
\g{*n*} or \g*n*	Back reference to the *n*th capture.
\g{-*n*} or \g-*n*	Relative backreference to the *n*th previous capture.
(?n)	Recurse into the *n*th capture buffer.
(?&NAME)	Recurse into the named capture buffer.
(?*R*)	Recursively call the entire expression.
(*?(DEFINE)...*)	Define a subexpression that can be recursed into.
(*FAIL)	Fail submatch, and force the engine to backtrack.
(*ACCEPT)	Force engine to accept the match, even if there is more pattern to check.
(*PRUNE)	Cause the match to fail from the current starting position.
(*MARK:name)	Marks and names the current position in the string. The position is available in $REGMARK.
(*SKIP:name)	Reject all matches up to the point where the named *MARK* was executed.
(*THEN)	When backtracked into, skip to the next alternation.
(*COMMIT)	When backtracked into, cause the match to fail outright.
/p	Mode modifier that enables the ${^PREMATCH}, ${MATCH}, and ${^POSTMATCH} variables.
\K	Exclude previously matched text from the final match.

Potprogrami

potprogram počinje naredbom **sub (ime potprograma) {}**, definicije funkcije se mogu nalaziti bilo gdje u program (globalne su)

potprogram se poziva znakom **&**;

&(ime potprograma);

kako bi program vratio vrijednost potrebno je račun staviti u zadnji redak potprograma

ukoliko se potprogramu šalju vrijednosti one se pohranjuju u polje @_, gdje je $_[0] prvi poslani broj, a $_[1] je drugi broj

```
sub pro {

$_[0] + $_[1];

}

print "Unesite brojeve\n";

$prvi = <STDIN>;

$drugi = $prvi;

$rez = &pro($prvi, $drugi);

print "Rezultat je $rez\n";
```

Naredba return

Return za privremeno izlažene iz potprograma te vraćanje trenutne vrijednosti

Privatne varijable

varijable se deklariraju operatorom **my**

Formatirani ispis

Uz print postoji I **printf** funkcija koja služi za formatirani ispis, način korištenja isti kao kod c-a

Znak postotka se može uključiti sa %%

%10s -> desno poravnanje ispisa

%-10s -> lijevo poravnanje ispisa

%10.3f -> 10 znamenki od kojih su 3 decimalne

Argumenti naredbenog retka

Perl ima posebno polje @ARGV, pristup elementima je kao u svakom drugom polju, operator **<>** koristi elemente polja **@ARGV** kao imena datoteka iz kojih čita podatke

```
@ARGV = qw < dat1.txt >;

while (<>){

chomp;

print "$_\n";

}
```

Ispis: ispisuje sadržaj datoteke dat1.txt

Naredba say (samo od verzije 5.10 vrijedi)

Služi isto kao I print no ona automatski dodaje novi red na kraj ispisa, a može se služiti I za upisivanje u datoteku, say (FILEHANDLE) “nešto za upis u datoteku”;

Pipeline/Cjevovod

Pipeline se ostvaruje naredbom u cmd-u:

Perl ./(ime programa) <dino >Wilma -> programov input je datoteka dino a output je datoteka Wilma

Kontrolne strukture

Unless

U if strukturi tijelo se ispunjava ako je uvjet istinit, no u unless tijelo se ispunjava samo ako je uvjet ne istinit, također se poslije unless može koristiti else kao u običnom if/else značenju no else ovaj puta označava da je uvjet bio istinita

```
Unless ($fred != “da”){

Print “ne\n”;

}

If ($fred = “da”){

}

Else {

Print “ne\n”;

}
```

Until

Kada se želi obrnuti uvjet while petlje koristi se until

```
Until ($j > $i){

$j *= 2;

Print "$j\n";
```

} -> za i=20, j=1 ispisivalo bi j = 2, 4, 8, 16, 32 -> znači sve dok se uvjet ne bi ispunio tj. Dokle god j nebi bio veći od i

Petlja se vrti dok uvjet ne vrati istinu/ponavlja se toliko puta koliko je ne istinita

For

For se koristi kao u c-u

```
For($I = 0; $i< 10; $i++){

Print "$i "; -> ispisuje 0 1 2 3 4 5 6 7 8 9
```

Modificirani izrazi

Izrazi se mogu pojednostaviti ukoliko se zamijene mjesta unutar petlje ->

```
If ($n < 10) {

Print "Manje";
```

} -> postaje print "Manje" if $n<10;

```
$i*=2 until $i > $j;

&greet($_) foreach @person;
```

Kontroliranje petlji

Operator **last** -> u c-u istovjetan operator je break -> služi za izlaženje iz petlje

Operator **next** -> služi za skakanje na iduću iteraciju unutar petlje petlja se ne izvodi do kraja ukoliko ima još nego se skoči odmah na iduću iteraciju

Operator **redo** -> služi poput next samo što ne skaže na iduću iteraciju nego odmah iza zaglavlja petlje

```
    While(){

    Ovdje se dolazi s redo

    nešto

    }
```

Hash

Hash je struktura koja nam pomaže za lakše snalaženje s pojmovima tj. Da nam upotreba nekih pojmova bude laganija

$hash{$some_key} = vrijednost–> na mjesto “hash” se stavlja željeno ime, “some_key” je oznaka/ključ s kojim ćemo dohvatiti vrijednost iza znaka jednakosti

```
$family_name{"fred"} = "flintstone";
$family_name{"barney"} = "rubble"; -> Ovdje smo unutar grupe family name dodijelili ime fred  te njemu dodijelili prezime

foreach $person (qw< barney fred >) {
print "I've heard of $person $family_name{$person}.\n";
}
```

- ➔ ispisuje se I’ve heard of fred flinstone.
- ➔ I’ve heard of Barney Rubble.

Da bi se koristio cijelim hash koristi se operator “%”

%some_hash = (“foo”, 12, 14, “bar”, 15.2, 1e-12, “bye\n”); -> ovako smo unutar some_hash stavili sve ove ključeve I vrijednosti (ključ, vrijednost, ključ, vrijednost, ključ, vrijednost, ključ, vrijednost)

@polje = %some_hash; -> sada smo cijelim hash prebacili u jedno polje koje možemo ispisati

%novi_hash = %some_hash; -> kopirali smo some_hash u novi_hash

%novi = reverse %novi_hash; -> redoslijed elemenata smo obrnuli u novom hashu (prije (ključ, vrijednost, ključ, vrijednost) kasnije (vrijednost, ključ, vrijednost, ključ))

Kako bi se vrijednosti mogle lakše dodavati ključevima postoji drukčiji način:

```
my %last_name = ( # a hash may be a lexical variable
"fred" => "flintstone",
"dino" => undef,
"barney" => "rubble",
"betty" => "rubble",
); -> odvajamo zarezom svaku skupinu podataka ključ => vrijednost
```

Hash funkcije

Keys, values -> koriste se kako bi se u polje spremili ili ključeve ili vrijednosti no ukoliko umjesto polja koristimo varijablu dobivamo samo broj ključeve/vrijednosti

```
my %hash = ("a" => 1, "b" => 2, "c" => 3);
my @k = keys %hash;
my @v = values %hash;

my $broj = keys %hash; -> dobivamo broj 3
```

Funkcije exsists / delete

Exsists-> vraća istinu ukoliko traženi ključ postoji unutar hasha bez obzira na vrijednost

Delete -> briše zadani element hasha

```
if (exists $books{"dino"}) {
print "Hey, there's a library card for dino!\n";
} -> vraća istini ako se dino nalazi unutar liste ključeve keys %books

my $person = "betty";
delete $books{$person}; # Revoke the library card for $person
```

Datoteke

Otvaranje datoteke vrši se operatorom **open**, navodi se identifikator I ime

Filehandle -> I/O veza između perlovom procesa I vanjskog svijeta, to je ime veze

Open CONFIG, “dino” #z a čitanje po default (CONFIG je filehandle datoteke dino koja se otvara)

Open CONFIG, “<dino” #za čitanje

Open CONFIG, “>dino” #za pisanje

Open CONFIG, “>>dino” #za dopisivanje

Datoteka se zatvara naredbom **close**

Prijevremeni izlaka se postiže naredbom **die** -> ispisuje grešku na stderr I prekida izvođenje programa

Naredba **warn** -> služi za upozorenje, a radi na sličnom principu kao die samo što ne mora odmah završiti program

If (! open CONFIG, “>>dino”){

Die “Cannont create CONFIG file: $!”;

} **poseban operator $! Sadrži poruku sustava o razlogu pogreške**

Ako ne želimo ispis retka I imena programa moramo završiti s \n die “nema dovoljno\n”;

Ispis u datoteku korištenjen print I printf, datoteka mora biti otvorena za pisanje ili dopisivanje

Print LOG “Upisujem u log datoteku\n”;

Printf STDERR (“Do sada preneseno %d posto.\n”, $done/$total * 100);

Podrazumijevani identifikator je stdout no to se može promijeniti naredbom **select**

Select BEDROCK

Print “upisujem u bedrock datoteku”;

Na kraju prebaciti na stdout

Select STDOUT;

Operatori upravljanja datotekama

die "Oops! A file called '$filename' already exists.\n"
if -e $filename; -> ukoliko postoji datoteka s istim imenom ispisuje se poruka

foreach (@lots_of_filenames) {
print "$_ is readable\n" if -r; # same as -r $_
} -> ukoliko je datoteka čitljiva ispisuje se poruka

-r	File or directory is readable by this (effective) user or group
-w	File or directory is writable by this (effective) user or group
-x	File or directory is executable by this (effective) user or group
-o	File or directory is owned by this (effective) user
-R	File or directory is readable by this real user or group
-W	File or directory is writable by this real user or group
-X	File or directory is executable by this real user or group
-O	File or directory is owned by this real user
-e	File or directory name exists
-z	File exists and has zero size (always false for directories)
-s	File or directory exists and has nonzero size (the value is the size in bytes)
-f	Entry is a plain file
-d	Entry is a directory

-l	Entry is a symbolic link
-S	Entry is a socket
-p	Entry is a named pipe (a “fifo”)
-b	Entry is a block-special file (like a mountable disk)
-c	Entry is a character-special file (like an I/O device)
-u	File or directory is setuid
-g	File or directory is setgid
-k	File or directory has the sticky bit set
-t	The filehandle is a TTY (as reported by the isatty() system function; filenames can’t be tested by this test)
-T	File looks like a “text” file
-B	File looks like a “binary” file
-M	Modification age (measured in days)
-A	Access age (measured in days)
-C	Inode-modification age (measured in days)

my @original_files = qw/ fred barney betty wilma pebbles dino bamm-bamm /;
my @big_old_files; # The ones we want to put on backup tapes
foreach my $filename (@original_files) {
push @big_old_files, $filename
if -s $filename > 100_000 and -A $filename > 90;
}

Ukoliko je datoteka veća od 100kb I nije bila čitana 90 dana

if(-r $file and -w $file) {
... } –> ukoliko je datoteka za čitanja I pisanje onda…

Virtualni file handler -> “_” -> služi kako bi se ponovila stara vrijednost koju smo koristili

if(-r $file and -w $file) {
... } –> ukoliko je datoteka za čitanja I pisanje onda…

postaje

if(-r $file and -w _) {

... } -> _ zamjenjuje pisanje $file po drugi puta

Od verzije 5.010 mogu se nizati operatori if(-r –w –M $file){nešto}

Baze podataka (DBI Modul)

Connect Funkcija -> služi za spajanje na bazu podataka

```
Use DBI;

$dbh = DBI->connect($data_source, $username, $password);
```

Nakon spajanje na bazu podataka prolazi se kroz niz priprema, pokretanja te čitanja podataka:

```
$sth = $dbh->prepare("SELECT * FROM foo WHERE bla");
$sth->execute();
@row_ary = $sth->fetchrow_array;
$sth->finish;
```

Nakon završenja treba se maknuti s baze:

$dbh->**disconnect**();

Stat / Istat funkcije

Stat funkcija se koristi kako bi se dobilo 13 podataka o datoteci (tko je napravio, kada, kakva je…)
Istat funkcija se koristi kada se želi dobiti informacije o simboličkom linku (ne koristi se u biti)

```
my($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime, $ctime, $blksize, $blocks) = stat($filename);
```

$dev and $ino
The device number and inode number of the file. Together they make up a "license plate" for the file. Even if it has more than one name (hard link), the combination of device and inode numbers should always be unique.

$mode
The set of permission bits for the file, and some other bits. If you've ever used the Unix command *ls -l* to get a detailed (long) file listing, you'll see that each line of output starts with something like -rwxr-xr-x. The nine letters and hyphens of file permissions† correspond to the nine least-significant bits of $mode, which would, in this case, give the octal number 0755. The other bits, beyond the lowest nine, indicate other details about the file. So, if you need to work with the mode, you'll generally want to use the bitwise operators covered later in this chapter.

$nlink
The number of (hard) links to the file or directory. This is the number of true names that the item has. This number is always 2 or more for directories and (usually) 1 for files. You'll see more about this when we talk about creating links to files in Chapter 13. In the listing from *ls -l*, this is the number just after the permissionbits string.

$uid and $gid
The numeric user ID and group ID showing the file's ownership.

$size
The size in bytes, as returned by the -s file test.

$atime, $mtime, and $ctime
The three timestamps, but here they're represented in the system's timestamp format: a 32-bit number telling how many seconds have passed since the *Epoch*, an

arbitrary starting point for measuring system time. On Unix systems and some others, the Epoch is the beginning of 1970 at midnight Universal Time, but the Epoch is different on some machines. There's more information later in this chapter on turning that timestamp number into something useful.

Vrijeme

my $now = gmtime;
print "$now"; -> ispisuje trenutno lokalno vrijeme -> Wed Feb 25 16:16:37 2009

Operatori nad bitovima

Expression	Meaning
10 & 12	Bitwise-and—which bits are true in both operands (this gives 8)
10 | 12	Bitwise-or—which bits are true in one operand or the other (this gives 14)
10 ^ 12	Bitwise-xor—which bits are true in one operand or the other but not both (this gives 6)
6 << 2	Bitwise shift left—shift the left operand the number of bits shown by the right operand, adding zero-bits at the least-significant places (this gives 24)
25 >> 2	Bitwise shift right—shift the left operand the number of bits shown by the right operand, discarding the leastsignificant bits (this gives 6)
~ 10	Bitwise negation, also called unary bit complement—return the number with the opposite bit for each bit in the operand (this gives 0xFFFFFFF5, but see the text)

Operacije nad direktorijima

Glob

Glob služi kako bi se našli file-ovi koji završavaju sa određenom ekstenzijom

```
#!user/bin/perl
@pom = glob "*";          -> u polje pom stavljamo popis svih file-ova koji se nalaze unutar trenutnog foldera
@drugi = glob "*.txt";    -> u polje drugi stavljamo popis svih file-ova koji imaju ekstenziju .txt
print "Sad ide prvi: @pom\n";
print "Sad ide drugi: @drugi\n";

@dir = glob "/etc";                  -> iz direktorija u kojem se trenutno
@dir_files = glob "$dir/* $dir/.*";  -> u polje stavljamo imena svih fileova koji imaju/nemaju ekstenziju uključujući I foldere
```

Ukoliko se želi može se upotrijebiti alternativna sintaksa za globbing (<*>)

```
@pom = glob "*";    postaje    @pom =<*>;
```

Brisanje file-ova

Unlink operator služi za Brisanje file-ova, kao povratnu vrijednost vraća broj uspješno izbrisanih file-ova

```
unlink "slate", "bedrock", "lava"; -> pobrisali smo 3 file-a
unlink glob "*.o"; -> brišemo sve file-ove koji završavaju na .o

foreach my $file (qw(slate bedrock lava)) {
unlink $file or warn "failed on $file: $!\n";
}
```

Preimenovanje file-ova

Preimenovanje se obavlja **rename** funkcijom, funkcija vraća vrijednost true/false ovisno o uspješnosti Operacije te tu vrijednost sprema u $! Kako bi se ispisala pravilna poruka o problemu
rename “staro_ime”, “novo_ime”;

Kreiranje i premještanje direktorija

Za Kreiranje direktorija koristi se naredba **mkdir**
mkdir "ime”, dozvole_pristupa
(0755 npr. Daje nama potpunu kontrolu no ostali imaju samo prava za čitanje)

```
my ($name, $perm) = @ARGV; # first two args are name, permissions
mkdir $name, oct($perm) or die "cannot create $name: $!";
```

za brisanje praznih direktorija koristimo funkciju **rmdir** (sličnom kao unlink)

```
foreach my $dir (qw(fred barney betty)) {
rmdir $dir or warn "cannot rmdir $dir: $!\n";
}
my $temp_dir = "/tmp/scratch_$$"; # based on process ID; see the text
mkdir $temp_dir, 0700 or die "cannot create $temp_dir: $!";
...
# use $temp_dir as location of all temporary files
...
unlink glob "$temp_dir/* $temp_dir/.*"; # delete contents of $temp_dir
rmdir $temp_dir; # delete now-empty directory
```

za mijenjanje dozvola prisupa koristi se **chmod**

```
chmod 0755, "fred", "barney";
```

Mijenjanje prava pristupa

chown -> mijenja se vlasništvo te grupno vlasništvo datoteke

```
my $user = 1004;
my $group = 100;
chown $user, $group, glob "*.o";
```

getpwnam -> služi za prevođenje korisničkog imena iz tekstuanlog oblika u brojčani
getgrnam -> za prevođenje grupnog imena u broj

```
defined(my $user = getpwnam "merlyn") or die "bad user"; ->  defined služi kako bi se potvrdilo da povratnu vrijednost nije nedefinirana što će se vratiti ukoliko korisnik I grupa nisu postojeći
defined(my $group = getgrnam "users") or die "bad group";
chown $user, $group, glob "/home/merlyn/*";
```

Mijenjanje vremenskih oznaka

utime -> koristi se za mijenjanje kada se datoteka zadnji puta modificirala

$sada = time; -> trenutno vrijeme

$prije = $sada – 24*60*60; -> sekunde po danu
utime $sada, $prije, glob "*"; -> stavlja pristup na sada, a mod na dan ranije

Formatiranje podataka sa sprintf

sprintf -> uzima vrijednost poput printf I sprema je u string
$datum = sprintf "%4d/%2d", $godina, $dan; -> u datum smo spremili nešto poput 2038/01

Operator <=> isto radi I cmp funkcija
Uspoređuje dva broja te vraća -1, 0, 1 kako bi se sortirali automatski
Sub po_broju { $a ⇔ $b }

Smart Match Operator (~~)

(~~)

- ➔ gleda oba operanda te odlučuje kako će ih usporediti, ako ih gleda kao brojeve napravi brojčani usporedbe, ako ih gleda kao stringove napravi stringovnu usporedbu, ako je jedan od operanda regularni izraz tada radi usporedbu uzorka
- ➔ Pomoću njega se mogu uspoređivat I polja samo stavimo polje s jedne I druge strane

```
say "I found Fred in the name!" if $name ~~ /Fred/;
if @imena1 ~~ @imena2 {nešto}
```

Example	Type of match
%a ~~ %b	Hash keys identical
%a ~~ @b	At least one key in %a is in @b
%a ~~ /Fred/	At least one key matches pattern
%a ~~ 'Fred'	Hash key existence exists $a{Fred}
@a ~~ @b	Arrays are the same
@a ~~ /Fred/	At least one element matches pattern
@a ~~ 123	At least one element is 123, numerically
@a ~~ 'Fred'	At least one element is 'Fred', stringwise
$name ~~ undef	$name is not defined
$name ~~ /Fred/	Pattern match
123 ~~ '123.0'	Numeric equality with "numish" string
'Fred' ~~ 'Fred'	String equality
123 ~~ 456	Numeric equality

The given Statement

Given-when je kontrolna struktura ekvivalentna strukturi switch-case u c-u

```
given( $ARGV[0] ) {
when( /fred/i ) { say 'Name has fred in it';  continue}
when( /^Fred/ ) { say 'Name starts with Fred'; continue }
when( 'Fred' ) { say 'Name is Fred'; break }
default { say "I don't see a Fred" }
}
```

Procesi

system

Najlakši način da bi se napravio proces dijete je sa funkcijom system

```
system "date";
```

ispis:

```
The current date is: ned 01.03.2009
Enter the new date: (dd-mm-yy)
```

exec

služi za pokretanje programa ili sistemskih funkcija

```
exec "date";
die "date couldn't run: $!";

exec "bedrock", "-o", "args1", @ARGV;
```

Korištenje `` za output

```
my $now = `date`; # grab the output of date
print "The time is now $now"; # newline already present
```

Sleep funkcija

sleep (izraz); -> program "spava" onoliko sekundi koliko se stavi u izraz