SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Kolegij: Programska podrška mjernih i procesnih sustava

Ognjen Krkač
(mat. br.: 0036368535)

PERL

(seminarski rad)

Zagreb, siječanj 2004.

SAŽETAK

Perl je skriptni interpreterski programski jezik opće namjene.

Perl je engleska skraćenica za Practical Extraction and Report Language.

Perl je prenosiv programski jezik za sve OS (UNIX, Linux, Windows, Mac, ...).

Perl je besplatan, 'open-source' jezik.

Perl je modularan (proširiv novim funkcijama i mogućnostima koje korisnici sami proizvedu).

Perl podržava proceduralno i objektno programiranje.

1. UVOD

Prvu inačicu Perla napisao je Larry Wall još davne 1987. godine. Larry je tada radio sustav izvješćivanja pogrešaka prilikom kompajliranja programa. Taj sustav poruka trebao je biti po strukturi jednak Usenetu. Budući da je tada *awk* bio neprimjeren za takve zadatke, a *C* jezik pak presložen, Larry se odlučio na stvaranje posebnog alata koji bi bio moćan s obradom teksta i stvaranjem izvješća. Taj alat prerastao je na kraju u programski jezik koji je Larry podijelio s ostalima putem Useneta. Tako je zaživio Perl - interpreterski jezik pisan u C-u i namijenjen samo UNIX platformama.

Jednostavnost, veća brzina izvođenja i fleksibilnost koju je Perl nudio naspram ostalih jezika u to vrijeme za tu primjenu, učinile su ga vrlo popularnim. Zbog toga se Perl brzo razvijao obuhvaćajući sve više novih mogućnosti (koje su bile vezane i s razvojem UNIX kernela). Detaljniji vremenski prikaz razvoja Perla možete pogledati na **http://history.perl.org/PerlTimeline.html**.

Do danas, Perl je dogurao do verzije 5.18.x i postao jedan od opće prihvaćenih skriptih programskih jezika opće namjene. Nije ograničen samo na UNIX platforme, već postoji podrška i za druge operativne sustave (Windows, Mac, ...). Njegova posebnost je u tome što u sebi objedinjuje mogućnosti *C*, *sed* i *awk* programskih jezika, te UNIX-ovih ljuski.

2. NAMJENA PERLA

Kako se s vremenom razvijao, Perl se nije zadržao samo na procesiranju teksta, što mu je bila prvotna namjena. Danas je Perl naširoko primjenjivan za zadaće kao što su: administracija sustava, razvoj iteraktivnih web stranica, mrežno programiranje, grafičko programiranje, izrada sistemskih i običnih aplikacija.

U načelu, Perl se primjenjuje kada je potrebno brzo, jednostavno i efikasno razvijanje ispitnih okruženja programa ili njihovih prototipova te kada se zahtijeva široka prenosivost.

Perl podržava proceduralno i objektno orjentirano programiranje. Poput C i C++, Perl nema striktnu granicu između procedura ili objekata.

Perl je izveden kao modularan, tj. njegov interpreter se može dopunjavati bibliotekama koje mu proširuju funkcionalnost. U tom pogledu, Perl se može pohvaliti vrlo velikom bazom modula koje su razvili poklonici Perla. Baza modula je okupljena oko **CPAN** (eng. Comprehensive Perl Archive Network) mreže (**www.cpan.org**). Tako Perl interpreter nije ograničen samo za određenu primjenu, kao što je to npr. Matlab. To je razlog zašto se u žargonu kaže da je Perl "otvoren prema svima, ali nije poseban prema nekome".

Perl je od početka besplatan i open-source projekt. Danas je oko tog projekta okupljena široka zajednica programera koji dobrovoljno sudjeluju u njegovu razvoju. Voditelj projekta je Perlov tvorac Larry Wall. Svakako da je ta strategija razvoja pridonijela njegovoj popularizaciji.

Zanimljivost kod Perla je da ne podliježe GNU (**www.gnu.org**) licenci, iako je uključen u njihove distribucije. Sam po sebi Perl je besplatan, no Perl program nekog autora može se naplaćivati ovisno o njegovoj želji.

3. INSTALACIJA PERLA

3.1 UNIX / Linux OS

Perl je od samog početka razvijen kao jezik na UNIX operativnim sustavima jer koristi njegove osnovne pozive kernela. Kako je njegova raširenost rasla, tako je s vremenom postao sastavni dio distibucija UNIX i Linux operativnih sustava. Ipak, distribucije obično ne sadrže najnovije verzije Perla pa ga je potrebno skinuti s Interneta.

S Interneta se može skinuti kompajlirana verzija Perla (eng. ports, binary distributions), no to nije preporučljivo zbog već podešenih postavki prije kompajliranja koje ne moraju odgovarati onima koje želimo.

Najbolje je skinuti izvorni kod s Interneta, te ga kompajlirati na vlastitiom računalu. Najnoviji izvorni kod moguće je dohvatiti sa **www.cpan.org/src/stable.tar.gz**.

Kao što je vidljivo iz imena datoteke, izvorni kod dolazi u obliku *.tar* arhive komprimirane *gzip* alatom. Otpakiravanje arhive izvodi se slijedećom naredbom u ljusci:

```
$ tar xvzf stable.tar.gz
```

Nakon toga slijedi proces kompajliranja koji se najčešće izvodi slijedećim naredbama u ljusci:

```
$ cd stable
$ ./configure
$ make
```

Gornjim naredbama postavlja se tekući direktorij *stable* (koji je nastao otpakiravanjem arhive *stable.tar.gz*), učitavaju se postavke za proces kompajliranja i na kraju se pokreće samo kompajliranje.

Preporučljivo je prije procesa kompajliranja pročitati datoteku *Readme* koja dolazi sa izvornim kodom, a u kojoj je detaljno opisana procedura kompajliranja.

3.2 Windows OS

Za Windows platforme razvijena je posebna verzija Perla naziva **ActivePerl** koju je izradila tvrtka *ActiveState*.

ActivePerl je također besplatan i njegova instalacija je dostupna sa stranice **www.ActiveState.com/ActivePerl**.

Valja primjetiti kako postoji i verzija ActivePerla za Linux i ostale platforme. U ovom slučaju biti će spomenuta samo instalacija za Windows platforme.

Instalacija se pokreće jednostavnim aktiviranjem skinute *.exe* arhive. Postupak je zdravorazumski i samo je potrebno slijediti upute.

Treba napomenuti da je za starije verzije Windows OS (9x/Me/NT/2000) potrebno skinuti i poseban modul (MSI Installer) koji treba instalirati prije instalacije ActivePerla.

Nakon instalacije dobro je provjeriti da li je podešena varijabla okoline PATH tako da uključuje i *C:\Perl\bin* direktorij.

4. STRUKTURA PERL PROGRAMA

Prije opisivanja strukture programa pisanih u Perlu, posebno je naglasiti par konvencija. Kada se kaže "Perl", misli se pod time na elemente, pravila i definicije koje objedinjuje taj programski jezik, dok "perl" predstavlja njegovu izvedbu (interpreter).

Druga zanimljivost je da u smislu Perl kodiranja možemo koristiti termine program i skripta s jednakim smislom. I jedno i drugo je ispravno.

Perl program je običan tekstualni dokument koji se može pisati klasičnim editorom teksta (npr. *joe* na Linux OS, ili *Notepad* na Windows OS).

Prvi redak Perl programa obično počinje posebnim komentarom koji počinje sa *#!*. On služi kao uputa UNIX ili Linux ljusci (npr. *bash*) da se radi o Perl skripti koju Perl interpreter treba izvesti. Npr., jednostavan Perl program može izgledati ovako:

```
#!/usr/bin/perl
print "Ovo je jednostavan Perl program.";
# ovo je komentar
```

Kao što se vidi iz primjera, uputa ljusci u prvom retku zapravo govori kojim se programom izvodi skripta i gdje se njen interpreter nalazi. U našem slučaju, Perl interpreter se nalazi u direktoriju /usr/bin.

Kod Windows OS prva linija ne mora uopće biti komentar. Važno je samo da skripta ima sufiks *.pl* i da za takav tip datoteke OS poziva Perl interpreter.

Drugi redak primjera je izraz koji sadrži poziv Perlove predefinirane funkcije - operatora. To je *print* operator koji proslijeđuje tekst unutar navodnika na standardni izlaz STDOUT. Drugim riječima, ispisuje ga na monitoru. Treba primjetiti da svaki Perl izraz mora završavati sa *;*.

U trećem retku napisan je komentar. Obični komentari u Perlu počinju znakom # iza kojeg slijedi tekst koji interpreter preskače pri izvođenju. Ti komentari služe za umetanje teksta koji programeru služe za jednostavnije snalaženje u kodu.

Nakon pokretanja programa, Perl interpreter u potpunosti provjerava kod pozvanog programa, te ga kompajlira. Tek nakon toga počinje njegovo izvršavanje izraz po izraz. Nakon uspješnog izvođenja programa, Perl proces završava vraćajući ljusci iz koje je pozvan vrijednost o uspješnom kraju programa.

Iz takvog slijeda izvršenja Perl skripte, može se zaključiti da Perl nije efikasan za male skripte i njihova višestruka pokretanja. Razlog je taj što samo kompajliranje uzima nešto vremena i resursa prije izvođenja, pa trajanje izvršenja samog programa može biti manje od trajanja kompajliranja. Vidljivo je da u tom slučaju korisnost (omjer između korisnog i ukupno utrošenog vremena) procesorskog vremena mala.

5. VARIJABLE I NJIHOVI OPERATORI

U ovom dijelu biti će predstavljena tri tipa varijabli koji postoje u Perlu. To su: skalari (eng. scalars), polja (eng. arrays) i združena polja (eng. associative arrays). Imena varijabli sastavljena su od početnog znaka koji definira tip varijable i imena varijable.

Za razliku od ostalih jezika, Perl ne zahtijeva prethodne deklaracije varijabli, već se one automatski definiraju prilikom postavljanja vrijednosti varijable.

Također, ovdje će ukratko biti spomenuti neki od najčešće korištenih operatora. Njihova namjena je promjena vrijednosti varijabli. Operatori se dijele s obzirom na tip verijable. Više o samim operatorima može se doznati iz **perlop** *man* stranica.

5.1 Skalarne varijable

Skalarne varijable služe za pohranu brojeva ili niza znakova, te se označavaju kao:

```
$ime_varijable
```

Pridjeljivanje vrijednosti skalarima dano je u slijedećih nekoliko primjera:

```
$broj = 4;
$dec_broj = 4.5;
$float_broj = 3.14e10;
$pozdrav = "Pozdrav iz Zagreba!";
$vrijeme = 'vedro i hladno';
```

Svi brojevi u Perlu interno se pohranjuju *double* deklaracijom u C-u. Tek se s verzijom 5 uvela mogućnost interne pohrane brojeva kao cjelobrojnih (*int*deklaracija u C-u) s ciljem optimizacije programa.

Najčešći operatori nad brojevima su:

```
+   plus
-   minus
*   množenje
/   dijeljenje
**  eksponent
%   modulo
==  jednako
!=  različito
<   manji od
>   veći od
<=  manji od ili jednako
>=  veći od ili jednako
+=  # $i=$i+$a je isto kao $i+=$a
-=  # $i=$i-$a je isto kao $i-=$a
*=  # $i=$i*$a je isto kao $i*=$a
```

Znakovni nizovi mogu biti definirani unutar dvostrukih i jednostrukih navodnika. Dvostruki navodnici označavaju da se unutar znakovnog niza prepoznaju nazivi varijabli, te da vrijednost toga niza ovisi o vrijednosti te varijable. Npr.:

```
$ime = "Ognjen"
$obrazac = "$ime pohađa FER."
# $obrazac postaje "Ognjen pohađa FER."
```

Jednostruki navodnici označavaju da znakovni niz ima fiksnu vrijednost.

Najznačajniji operator znakovnih nizova je *chop($ime_varijable)* koji briže zadnji znak niza.

5.2 Polja

Polja služe za pohranu niza skalara, te se označavaju kao:

```
@ime_polja
```

Primjeri pridjeljivanja vrijednosti poljima dano je kroz nekoliko primjera:

```
@trio = ("ognjen", "drazen", "marin", "4401");
@trio_gusti = @trio;
```

Pristup određenom elementu polja izvodi se na slijedeći način:

```
$trio[3] = "1044";
```

Gornjim primjerom promijenjen je zadnji element polja "4401" u "1044". Treba primjetiti da indeksiranje elemenata polja počinje od 0.

Najčešći operatori polja su dani kroz slijedeće primjere:

```
push (@trio, "ivan");
# na kraj polja @trio dodaje se "ivan"
$ime = pop (@trio);
# "ivan" s kraja polja sprema se u $ime
unshift (@trio, "ivan");
# "ivan" se dodaje na početak polja
$ime = shift (@trio);
# uzima se prvi element polja
@trio = reverse (@trio);
# obrće se redoslijed elemenata
@aList = sort (@aList);
# abecedno sortiranje elemenata
chop (@aList);
# svakom elementu polja briše se zadnji znak
@linija = ;
# jedna linija sa standardnog ulaza pohranjuje se u @linija
```

5.3 Združena polja

Združena polja zapravo predstavljaju dvodimenzionalno polje skalara. Pojedini element (skalar) takvog polja nije indeksiran cijelim brojem već proizvoljnim skalarom koji se naziva ključ (eng. key). Svaki element polja ima svoj jedinstveni ključ prema kojem se razlikuje od ostalih elemenata. Združena polja se označavaju kao:

```
%ime_zdruzenog_polja
```

Primjeri pridjeljivanja vrijednosti poljima dano je kroz nekoliko primjera:

```
$a{"marko"} = 23;
$kljuc = ivan;
$a{$kljuc} = 25;
%b = %a;
```

Neki primjeri operatora za združena polja su slijedeći:

```
@kljucevi = keys(%a);
# vraća polje s vrijednostima ključeva
@elementi = values (%a);
# vraća polje vrijednosti elemenata
delete $a{"marko"};
# briše element s ključem "marko"
```

6. KONTROLA TIJEKA PROGRAMA

Tijek izvođenja programa u Perlu je od prvog prema zadnjem izrazu. On se može promijeniti uvjetnim petljama ili grananjem.

Kao u svim programskim jezicima, i u Perlu postoje izrazi za kontrolu tijeka programa. U nastavku će biti navedena samo njihova osnovna sintaksa, dok se njihov detaljniji opis može naći u **perlsyn** *man* stranicama.

if izraz služi za ispitivanje uvjetnog izraza. Ukoliko je on točan (TRUE) izvode se izrazi unutar vitičastih zagrada odmah iza *if*. U suprotnom, izvode se izrazi unutar vitičastih zagrada odmah iza *else*.

Uvjetni izraz daje vrijednost FALSE ako je rezultat tog izraza NULL string "" ili znamenka "0".

```
if (uvjetni_izraz_A) {
     A_točan_izraz_1;
     A_točan_izraz_2;
```

```
    A_točan_izraz_3;
} elsif (uvjetni_izraz_B) {
    B_točan_izraz_1;
    B_točan_izraz_2;
    B_točan_izraz_3;
} else {
    netočan_izraz_1;
    netočan_izraz_1;
    netočan_izraz_1;
}
```

while petlja izvodi blok izraza u vitičastim zagradama sve dok je uvjetni izraz koji ispituje točan (TRUE).

```
LABELA:  while (uvjetni_izraz) {
    izraz_1;
    izraz_2;
    izraz_3;
}
```

for petlja izvodi blok izraza u vitičastim zagradama sve dok je uvjetni izraz koji ispituje točan (TRUE).

```
for (početni_izraz; uvjetni_izraz; inkremetalni_izraz) {
# npr. for ($i=1; $i<5; $i++)
    izraz_1;
    izraz_2;
    izraz_3;
}
```

foreach petlja izvodi blok izraza u vitičastim zagradama za svaki element polja @polje, pri čemu je vrijednost tekućeg elementa polja sadržana u varijabli $i.

```
LABELA:  foreach $i (@polje) {
    izraz_1;
    izraz_2;
    izraz_3;
}
```

7. FUNKCIJE (PODRUTINE)

Funkcije su osnovni element svakog programskog jezika. Najčešće se ponašaju poput običnih operatora čineći promjenu vrijednosti neke varijable ili vraćajući neku vrijednost koja se može pridjeliti nekoj drugoj varijabli. Osim toga, u određenom kontekstu funkcije izvode i kontrolu tijeka izvođenja programa (mijenjajući tijek programa od poziva fukcije, preko izraza kojima je definirana, pa nazad do točke od kuda je pozvana).

Perl podržava pisanje funkcija, no u vlastitoj terminologiji one se nazivaju podrutine. Razlog je taj što se pod fukcijama misli na predefinirane funkcije - operatore.

Detaljnije o samim ppodrutinama može se naći u **perlsub** *man* stranicama.

7.1 Definicija podrutine

Definicija podrutine označava se s oznakom *sub* nakon čega slijedi ime funkcije i blok izraza unutar vitičastih zagrada. Pozivom funkcije izvest će se taj blok izraza. Na primjer:

```
sub ime_funkcije {
  izraz_1;
  izraz_2;
  izraz_3;
}
```

7.2 Poziv podrutine

Podrutina se može pozivati unutar nekog izraza ili zasebno. Njen poziv se sastoji od znaka *&* iza kojeg slijedi ime podrutine:

```
&ime_funkcije
```

Ukoliko se u podrutinu prenose argumenti, oni se unose unutar zagrada:

```
&ime_funkcije("ognjen", 23)
```

7.3 Vraćanje vrijednosti

Podrutina uvijek vraća vrijednost zadnjeg izraza koji je izveden unutar podrutine. Na primjer:

```
sub aFunction {
     izraz_1;
     izraz_2;
     $a = $b + $c;
}
```

U prethodnom primjeru podrutina vraća vrijednost varijable $a. Nakon povratka iz podrutine, vraćena vrijednost nalazi se u posebnoj varijabli *$_*.

7.4 Argumenti podrutine

Argumenti podrutine prenose se u podrutinu putem posebne varijable @_ koja predstavlja polje argumenata. Na primjer:

```
sub zbroji {
     $_[0]+$_[1]
}
```

Podrutina vraća vrijednost zbroja dvaju brojeva koji se u nju prenose putem argumenata. Poziv te podrutine bi na primjer izgledao ovako:

```
&zbroji(2, 5)
```

7.5 Lokalne varijable

Sve varijable definirane unutar Perl programa su globalne varijable.

Za podrutine je moguće definirati lokalne varijable koje su vidljive samo unutar podrutine. Na taj način ne može doći do promjene globalnih varijabli ako slučajno imaju jednako ime kao i neka od lokalnih varijabli. Lokalne varijable definiraju se operatorom *local()*:

```
sub neka_rutina {
     local ($A, $B);
     $A = $_[0];
     $B = $_[1];
}

&neka_rutina ($a, $b);
```

U prethodnom primjeru varijable unutar podrutine $A i $B će imati jednake vrijednosti kao i globalne varijable $a i $b. Bilo kakve promjene nad lokalnim varijablama neće utjecati na vrijednosti varijabli $a i $b.

8. REGULARNI IZRAZI

Regularni izrazi su Perlov pojam za znakove ili posebne znakovne (pod)nizove koji se žele pronaći unutar drugih znakovnih nizova. Regularni izrazi zapravo predstavljaju točno definirane uzorke znakovnih nizova (eng. patterns). Ti uzorci ne služe samo za pronalaženje tih uzoraka u drugim nizovima, već i za ostale operacije kao što je na primjer zamjena pronađenog uzorka u nizu nekim drugim znakovnim nizom. Tip operacije ovisi o primjenjenom operatoru.

Detaljnije o regularnim izrazima može se vidjeti u **perlretut** i **perlre** *man* stranicama.

Regularni izrazi u Perlu se definiraju na točno određen način. Najuobičajeni način definiranja je postavljanjem znakovnog niza (uzorka) unutar znakova /:

```
/neki_niz_znakova/
/stjepan_98/
```

Osim znaka / može se koristiti neki drugi ne-alfanumerički znak, samo je potrebno naglasiti znakom *m*:

```
m#uzorak#
m@stjepan_98@
```

Definirani regularni izraz se pretražuje u preodređenoj varijabli $_. Želi li se regularni izraz pretraživati unutar neke prethodno definirane varijable, tada se koristi operator =~:

```
$imena =~ /marko/;
```

Ukoliko se u varijabli $imena nalazi uzorak "marko", cijeli izraz vraća TRUE (u suprotnom FALSE). Budući da regularni izrazi vraćaju logički rezultat (TRUE/FALSE) vidljivo je da se oni većinom koriste kod kontrolnih struktura.

```
if (/ivan/)  {
  print "ivan postoji u $_\n";
}
```

8.1 Oblici uzoraka

Uzorci mogu biti znakovni nizovi koji se sastoje od jednog ili više znakova. Višeznakovni uzorci bili su prikazani na gornjim primjerima, dok jednoznakovni uzorci biti će prokazani kroz niz slijedećih primjera:

```
/.oki/
# odgovara nizu "coki," "moki", ali ne i "oki"
/[0123456789]/
# zamjenjuje bilo koji znak unutar uglatih zagrada
/[0-9]/
# isto kao i prethodni slučaj
/[^0-9]
# bilo koji znak osim brojeva
/[0-9a-zA-Z]/
# bilo koji alfanumerički znak
```

Predefinirane kategorije znakova su:

```
\d  # brojevi, [0-9]
\w  # slova, [a-zA-Z0-9_]
\s  # praznine, [ \r\t\n\f]
\D  # sve osim brojeva, [^0-9]
\W  # sve osim slova, [^a-zA-Z0-9_]
\S  # sve osim praznina, [^ \r\t\n\f]
```

Višeznakovni uzorci mogu u sebi sadržavati sekvencu znakova koji se ponavljaju određen broj puta. Zato u Perlu postoje specijalni znakovi kojima se definira točan broj ponavljanja u uzorku. Kroz slijedeće primjere biti će demonstrirani ti operatori:

```
/a*t/
# bilo koji broj znakova "a" iza kojeg slijedi "t"
```

```
/a+t/
# jedan ili više znaka "a" iza kojeg slijedi "t"
/a?t/
# nula ili jedan znak "a" iza kojeg slijedi "t"
/a{2,4}t/
# od 2 do 4 znaka "a" iza kojeg slijedi "t"
/a{2,}t/
# 2 ili više znaka "a" iza kojeg slijedi "t"
/a{2}t/
# točno 2 znaka "a" iza kojeg slijedi "t"
```

Za uzorke se može definirati i položaj na kojem mora biti unutar znakovnog niza koji se pretražuje. Na primjer:

```
/\bdr/
# \b == granica (prazninu, početak ili kraj niza)
# uzorak odgovara za "drvo" ali ne za "odrzavanje"
/ti\b/
# uzorak odgovara za "cuti" ali ne za "tisak"
/ti\B/
# \B == ne smije biti granica na tom mjestu
# uzorak odgovara za "tisak" ali ne za "cuti"
/^tr/
# ^ == početak niza
# uzorak odgovara za "trag" ali ne za "otrovan"
/ca$/
# $ == kraj niza
# uzorak odgovara za "kuca" ali ne za "cavao"
```

8.2 Operatori regularnih izraza

Najčešće korišten operator je zamjena koja je formulirana na slijedeći način:

```
$niz =~ s/uzorak/zamjena_za_uzorak/
```

Uzorak se pretražuje unutar varijable (skalara) $niz, te se svaki uzorak koji se podudara s njime zamjenjuje definiranim znakovnim nizom. Primjeri:

```
s/marko/ivan/
# zamjenjuje se "marko" s "ivan" (unutar $_)
s/marko/ivan/gi
# isto kao i prethodni samo operator sada
# nije osjetljiv na velika i mala slova
$a =~ s/marko/ivan/
# ista operacija se izvodi nad $a
```

Operator *split()* vraća polje skalara (znakovnih nizova) koji se nalaze između pronađenih uzoraka. Primjeri:

```
@var = split(/pattern/, $znak_niz);
@var = split(/pattern/);
```

Ukoliko nije definiran znakovni niz, uzima se sadržaj varijable $_.

Još jedan važan operator je *join()*. On uzima polje skalara (znakovnih nizova), te ih spaja dodajući između njih definirani znakovni niz:

```
@imena = ("ivan", "marko", "stjepan");
$a = join(" + ", @imena);
# kao rezultat skalar $a će sadržavati
# niz "ivan + marko + stjepan"
```

9. RUKOVANJE ULAZIMA / IZLAZIMA

Perl omogućuje osnovne ulazne/izlazne operacije koje obuhvaćaju prihvat podataka sa standardnog ulaza (tipkovnica), proslijeđivanje podataka na standardni izlaz (monitor), te pisanje i čitanje datoteka.

9.1 Standarni ulazi/izlazi

Standardni U/I su STDIN, STDOUT i STDERR. Pristup standardnom ulazu STDIN izvodi se preko <> operatora (eng. diamond operator). U skalarnom obliku, operator vraća jednu liniju s ulaza, dok u kontekstu polja operator vraća cijelu datoteku s time da svaki element polja predstavlja jednu liniju. Na primjer:

```
$a = ;  # vraća liniju
@a = ;  # vraća datoteku
```

Standardni izlaz STDOUT u najčešćem broju slučajeva nije potrebno posebno pozivati. Na primjer, operator *print* uvijek radi sa STDOUT proslijeđujući mu tekst koji mora ispisati.

Standardni izlaz za prijavljivanje pogreški STDERR je obično preusmjeren na monitor (STDOUT).

9.2 Rad s datotekama

Datotekama se pristupa putem posebnih pokazivača (eng. filehandles). Pokazivači su zapravo spona između datoteke i Perl procesa. Ti pokazivači se moraju povezati s datotekom koju će oni predstavljati u programu. Tome služi operator *open()*. Na primjer, pokazivač INPUT je povezan s datotekom *index.html* na slijedeći način:

```
open (INDEX, "index.html");
```

U primjeru je datoteka otvorena samo za čitanje. Ukoliko se u nju želi pisati to se radi na slijedeći način:

```
open (INDEX, ">index.html");
```

U ovom slučaju će se prebrisati stari sadržaj datoteke. Ukoliko se želi sadržaj datoteke nadopunjavati (eng. update), tj. dodavati podatke na njen kraj, potrebno je datoteku otvoriti na slijedeći način:

```
open (INDEX, ">>index.html");
```

Operator *open()* vraća vrijednost TRUE/FALSE s obzirom na to da li je operacija uspješno izvršena (TRUE za uspješno izvedenu operaciju). Provjera uspješno izvedene operacije *open()* može se ispitati na slijedeći način:

```
open (INDEX, ">index.html") || die "Poruka za SDTERR";
```

Funkcija *die()* proslijeđuje tekst unutar zagrada na STDERR i gasi Perl proces. U ovom primjeru ona će se izvoditi ukoliko je lijevi dio operatora || jednak FALSE (neuspješno otvorena datoteka).

Zatvaranje datoteke nije nužno izvršiti jer se obavlja automatski prilikom završenja Perl procesa. No, ukoliko ga je potrebno učiniti prije, za to služi operator *close()*. Ako u gornjem primjeru želimo zatvoriti datoteku *index.html* s pokazivačem INDEX, učiniti ćemo to ovako:

```
close(INDEX);
```

Čitanje datotete je isto kao i kod SDTIN. Operatorom <> pristupamo datoteci u skalarnom kontekstu ili u kontekstu polja.

```
$a = ;  # vraća liniju
@a = ;  # vraća cijelu datoteku
```

Pisanje u datoteku izvodi se operatorom *print*, samo što se mora prije definirati izlaz na koji se tekst proslijeđuje. Na primjer:

```
print INDEX "neki tekst $odlomak \n";
```

Ispitavanje statusa i atributa datoteka je u mnogo slučajeva korisno i poželjno. Time se mogu na vrijeme utvrditi nepravilnosti koje bi mogle nastati u programu.

Ispitivanje datoteka izvodi se operatorom - iza kojeg slijedi oznaka atributa koji se ospituje. Neki od ispitnih operatora su:

```
-r  # readable
-w  # datoteka za pisanje?
-x  # izvršna datoteka?
-e  # datoteka postoji?
-z  # prazna datoteka?
-d  # direktorij?
-l  # simbolički link?
-T  # tekstualna datoteka?
-B  # binarna datoteka?
```

Svaki od operatora vraća TRUE/FALSE vrijednost. Primjer:

```
if (-e INDEX) {
    print "Datoteka postoji!";
} else {
  die("Datoteka ne postoji!");
}
```

U primjeru se ispituje postojanje datoteke. Ukoliko ona ne postoji, uvjetni izraz vratiti će vrijednost FALSE čime će se pokrenuti *die()* operator koji će ugasiti Perl program.

10. SISTEMSKI POZIVI

Sistemski pozivi u Perlu su usko povezani s terminologijom UNIX operativnog sustava. Svi operatori omogućuju pozivanje određenih procesa iz ljuske.

Operatori se dijele na dvije glavne skupine: operatori datotečnog sustava i operatori procesa. U ovom dijelu će operatori biti samo ukratko navedeni, a njihova objašnjenja mogu se pronaći na **perlfaq8** i **perlfunc** *man* stranicama.

10.1 Operatori datotečnog sustava

Operatori za upravljanje datotekama i direktorijima pozivima su vrlo slični onima koji se pozivaju u ljusci. Najčešći operatori su:

```
chdir ("/put/ . . . /direktorij");
# promjena aktivnog sirektorija

opendir (POKAZIVAC, "/path/ . . . /directory");
# postavljanje pokazivača za direktorij

closedir(POKAZIVAC);
# zatvarenje direktorija

readdir(POKAZIVAC);
# vraća imena datoteka iz direktorija

symlink("put", "IME_VEZE");
# stvaranje simboličke veze

link("path/file", "IME_VEZE");
# stvaranje čvrste veze (eng. hard link)
```

```
unlink("IME_VEZE");
# brisanje veza na datoteku

mkdir("ime_direktorija", mod);
# stvaranje novog direktorija

rmdir("ime_direktorija");
# brisanje direktorija

chmod(mode, "ime_datoteke");
# promjena moda datoteka
```

10.2 Operatori procesa

Operatori procesa su vrlo moćni po svojim mogućnostima. Oni omogućuju izvedbe poput jednostavnih poziva drugih programa, te izvedbe vrlo složenih procesa kao što je klijent/server mrežni model.

U ovom dijelu samo će biti spomenuti neki od najčešćih operatora procesa. Detaljnije objašnjavanje primjera i operatora preraslo bi obim seminarskog rada.

Najjednostavniji sistemski operator je *system()*. On pokreće novi proces, dijete Perl procesa. Ime novog procesa je sadržano unutar zagrada:

```
system("pwd")
```

Vrijednost koju vraća proces dijete na STDOUT, moguće je dobiti ako se poziv procesa izvede pomoću unazadnih navodnika:

```
$a = `pwd`;
```

Pokretanje novog procesa može se učiniti operatorom *exec* samo što se u tom slučaju automatsk gasi Perl proces:

```
exec "pwd";
```

Najznačajniji operator kod baratanja procesima je zasigurno *fork*. On stvara klon Perl procesa koji ga je pozvao. Tako dva duplicirana procesa dijele jednake varijable (memoriju) i otvorene datoteke. Procesi se jedino razlikuju po vrijednosti varijable koju vraća *fork*. Proces vraća 0 za dijete i 1 za roditelja.