Programski jezici i strukture podataka

14

STRUKTURIRANO PROGRAMIRANJE

Pojam strukturiranog programiranja

- Pod pojmom strukturiranog programiranja podrazumevaju se tehnike sistematskog i disciplinovanog razvoja programskih sistema i njihove dokumentacije, pri čemu se razvoj odvija:
  - u koracima preciziranja,
  - uz korišćenje konačnog broja kontrolnih struktura
  - i uz stalnu kontrolu korektnosti programskih rešenja.
- U tehnici strukturiranog programiranja programi se realizuju koristeći postupak **hijerarhijskog ugneždivanja** struktura programa i podataka.

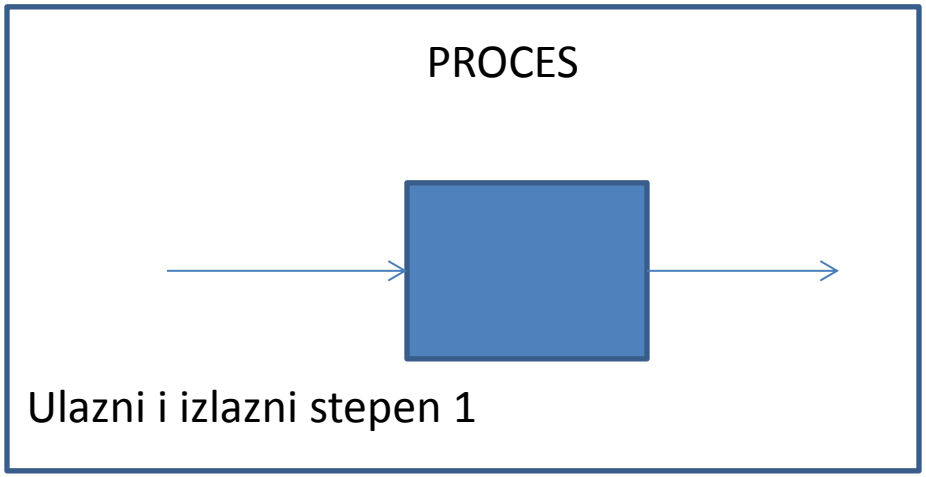
Pravilni programi

- Svaki dijagram toka predstavlja usmereni graf čiji čvorovi mogu biti:

1. PROCES
2. PREDIKAT
3. KOLEKTOR

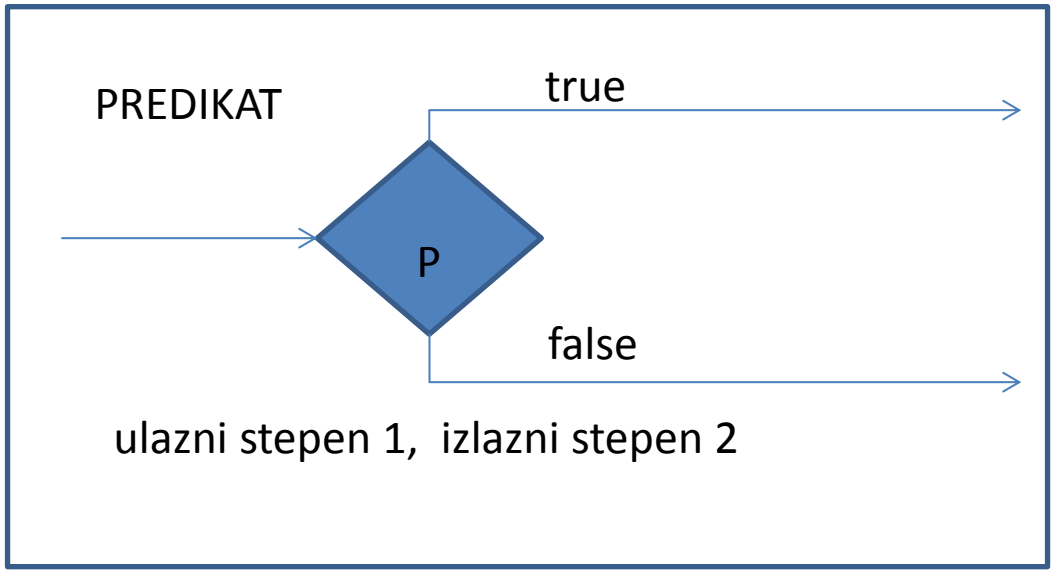
Funkcionalni čvor - *proces*

- **DEFINICIJA 1.** Element dijagrama toka sa jednom ulaznom i *jednom izlaznom* linijom u kome se obavlja obrada ili prenos podataka naziva se **proces.** Alternativni nazivi su:
  - **funkcijski** čvor,
  - ili **obrada**,
- Najčešća realizacija procesa je u vidu instrukcije **dodele vrednosti**).



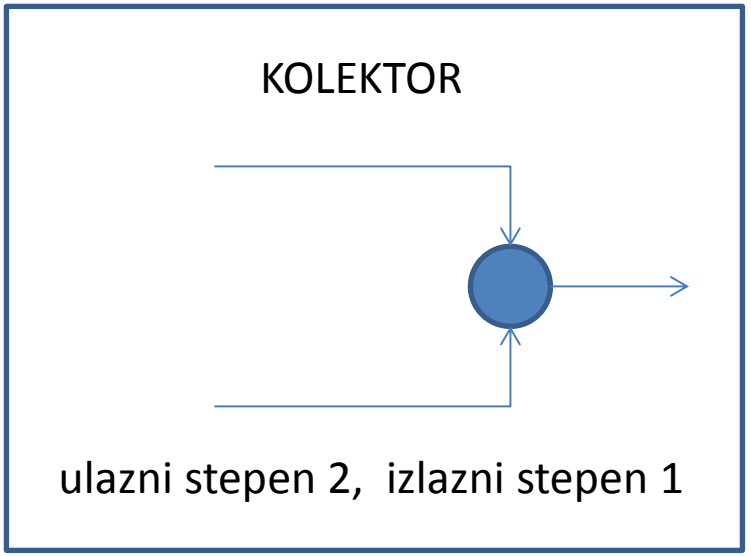
Predikat

- **DEFINICIJA 2. Predikat** je element dijagrama toka sa jednim ulazom i dva izlaza koji obavlja isključivo kontrolnu funkciju:
  1. **izračunava** vrednost pridruženog logičkog izraza
  2. zatim **usmerava** dalji tok obrade u pravcu jedne od izlaznih linija ako je izračunata vrednost ***true*,** a u pravcu druge od izlaznih linija ako je izračunata vrednost ***false*.**



Kolektor

- **DEFINICIJA 3**. **Kolektor** je element dijagrama toka sa dve ulazne linije i jednom izlaznom linijom koji ne obavlja obradu i koji se primenjuje onda kada dva nezavisna toka obrade treba usmeriti u nekom zajedničkom pravcu.



Pravilan program

- **DEFINICIJA 4**. Pod pojmom **pravilan program** (engl. proper program) podrazumeva se program čiji dijagram toka zadovoljava sledeća tri uslova:

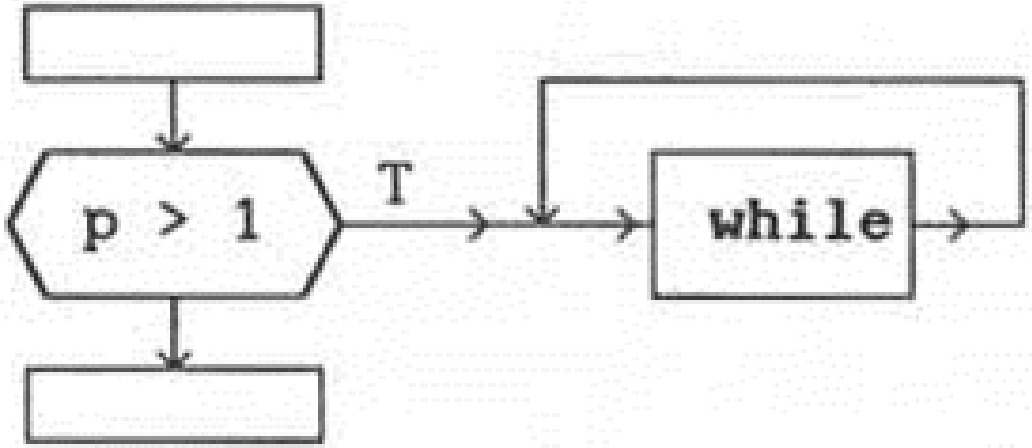
1. Program ima samo jednu ulaznu liniju.
2. Program ima samo jednu izlaznu liniju.
3. Za svaki čvor (proces, kolektor, ili predikat) postoji putanja koja ide od ulazne linije kroz taj čvor do izlazne linije.

Pravilni programi

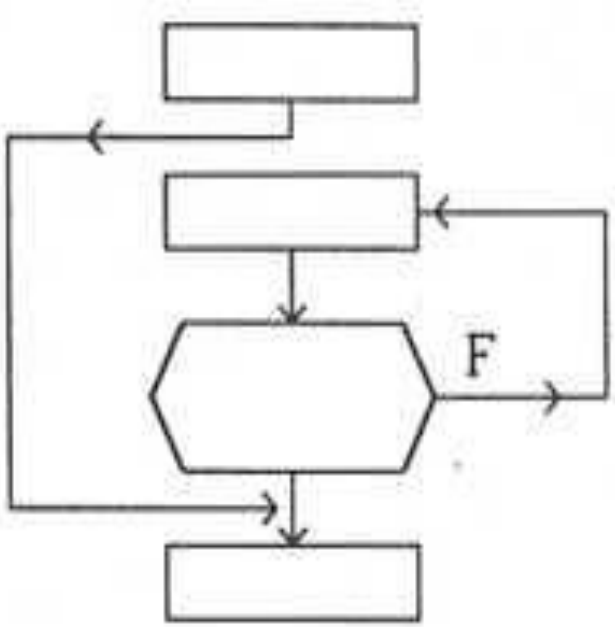
- Poslednji od navedenih uslova ukazuje da pravilni programi ne smeju imati:
  - **beskonačne petlje**
  - **izolovane (nedostižne) programske segmente**.

Primer programa koji sadrži beskonačnu petlju

- Ako je učitana vrednost za p veća od 1.
- Ulazi se u beskonačnu petlju iz koje nema izlaza.
- Ovaj program nije pravilan.



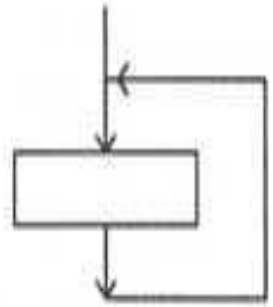
Primer programa sadrži izolovani programski segment



- DO-WHILE petlja nikad ne može izvršiti.
- To znači da navedeni program nije pravilan.

//Program:

```
while(1)
{
        read x, y, z ; write f(x, y, z);
        ....
}       //teorijski nije pravilan.
```



- Programi ovog tipa se u praksi pojavljuju.
- Oni se koriste kad je potrebno analizirati funkciju f(x,y,z) za razne vrednosti argumenata x, y, i z.
- Podrazumeva se da se ovakvi programi završavaju prekidom programa što korisnik postiže primenom nekog od signala prekida.

Pravilan potprogram

DEFINICIJA 5. Svaki sastavni deo pravilnog programa (tj. neki podgraf u dijagramu toka) koji je takođe pravilan program nazlva se pravilan potprogram.

- Svi **procesi** u sastavu pravilnog programa su pravilan potprogram.
- **Predikati nisu** pravilni potprogrami zato jer imaju po dva izlaza.
- **Kolektori nisu** pravilni potprogrami jer imaju po dva ili više ulaza.

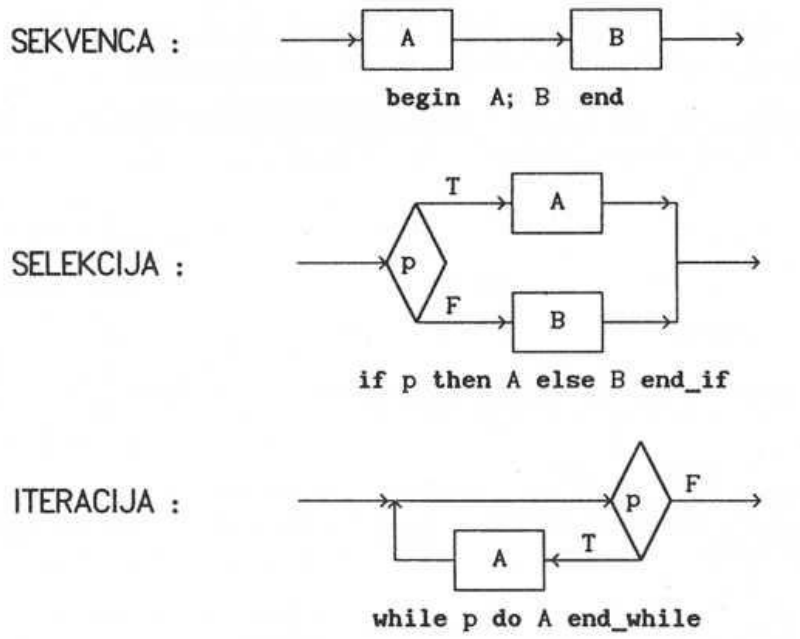
Prost program

DEFINICIJA 6. Pravilan program čiji svaki sastavni deo koji je pravilan potprogram sadrži samo jedan čvor, naziva se **prost program**.

- Prosti programi se ne mogu dalje dekomponovati u pravilne potprograme.
- Najviše primera prostih programa nalazimo među osnovnim kontrolnim strukturama.

Strukturna teorema (Teorema Giuseppe Jacopinija)

- Strukturna teorema se pojavila 1966.
- Posmatrane su sledeće tri osnovne kontrolne strukture:



- Postavljen je zadatak da se ustanovi pod kojim uslovima se programi mogu realizovati superpozicijom navedenih osnovnih kontrolnih struktura.
- **Za pravilne programe to je uvek moguće.**

Baza strukturiranih programa

DEFINICIJA 7. Skup prostih programa čijom superpozicijom se može realizovati proizvoljan pravilan program nazivamo baza strukturiranih programa.

- Baza strukturiranih programa može biti **dovoljna**, ako se izostavljanjem neke od kontrolnih struktura iz baze dobija neka jednostavnija baza.
- U protivnom baza je **potrebna**.

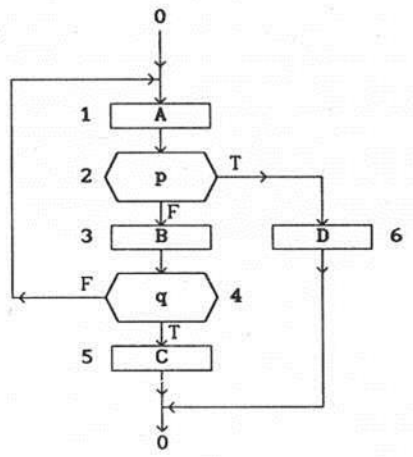
DEFINICIJA 8. Formalno strukturiran program je svaki pravilan program sačinjen korišćenjem jedino kontrolnih struktura iz određene baze.

DEFINICIJA 9. Dva programa su ekvivalentna onda kada realizuju istu obradu podataka, tj. kada za iste ulazne podatke generišu iste rezultate.

Funkcije veze

DEFINICIJA 10. Vrednost L(i,j) funkcije veze odreduje vezu od čvora **i** do čvora **j** u dijagramu toka.

- Ako je L(i,j)=T onda veza postoji.
- Ako je L(i,j)=F onda veze nema.
- Za svaku vrednost i postoji samo jedan j takav da L(i,j)=T.



POLAZNI ČVOR	ODREDIŠNI ČVOR 0	1	2	3	4	5	6
0	F	T	F	F	F	F	F
1	F	F	T	F	F	F	F
2	F	F	F	$\bar{p}$	F	F	p
3	F	F	F	F	T	F	F
4	F	$\bar{q}$	F	F	F	q	F
5	T	F	F	F	F	F	F
6	T	F	F	F	F	F	F

Tabela funkcije veze

Teorema o kanoničnoj formi

TEOREMA 1. Za svaki pravilan program postoji ekvivalentan program koji se sastoji od REPEAT-UNTIL (ili WHILE-D0) petlje unutar koje se nalazi CASE struktura sa onoliko paraleinih grana koliko ima čvorova u dijagramu toka.

REPEAT-CASE ili WHILE-CASE forma programa naziva se kanonična forma programa.

Osnovna strukturna teorema

TEOREMA 2. Svaki pravilan program može se transformisati u ekvivalentan formalno strukturiran program uz korišćenje baze od **tri osnovne kontrolne strukture** (sekvenca, selekcija i iteracija) primenjene na procese i predikate polaznog programa, i uz moguću primenu jedne dopunske selektorske promenljive.

Strukturna teorema sa dve kontrolne strukture

- Formulacija strukturne teoreme koju smo izložili u prethodnom tekstu je ona formulacija koja ima najviše upotrebne vrednosti u programerskoj praksi.
- Međutim, primetili smo da postoji više mogućih formulacija.
- Sa teorijske tačke gledišta interesantno je pokazati da je moguće strukturnu teoremu formulisati i primenom samo dve kontrolne strukture.

Strukturna teorema sa dve kontrolne strukture

TEOREMA 3. Svaki pravilan program može se transformisati u ekvivalentan formalno strukturiran program uz korišćenje baze od samo **dve osnovne** kontrolne strukture (sekvenca i iteracija) primenjene na procese i predikate polaznog programa, uz moguću primenu dopunskih selektorskih i logičkih promenljivih.

```
IF p THEN A ELSE B END_IF
```

je ekvivalentno sa:

```
alfa := p ; beta := not(p) ;
WHILE alfa DO
     A ;
     alfa := false
END_WHILE ;
WHILE beta DO
     B ;
     beta := false
END WHILE
```

Strukturirani programi

- Po definiciji strukturirano prograrairanje je razvoj programa u koracima preciziranja čime se dobijaju programi u formi hijerarhijski ugnežđenih struktura programskih iskaza i objekata koji se programom obrađuju.
- Ovom tehnikom se postižu sledeći ciljevi:
  - minimizacija broja grešaka tokom procesa razvoja softvera,
  - minimizacija rada potrebnog za razvoj korektnih programa, i
  - minimizacija troškova održavanja softvera.

ALGORITMI

Pitanja koja se postavljaju

- Kako da utvrdimo da li za neki problem postoji algoritam koji ga rešava?
- Da li je svaki problem algoritamski rešiv?
- Kako da prepoznamo problem za koji ne postoji algoritam koji ga rešava?

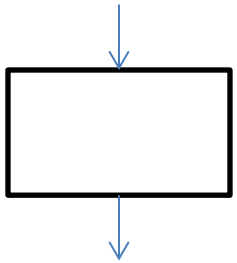
Formalizacija algoritama

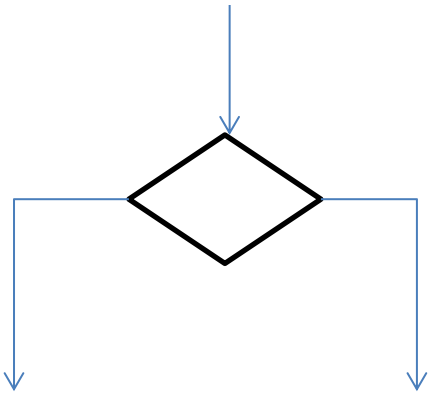
- Pitanjem formalizacije pojma algoritma se bavilo nekoliko istaknutih matematičara i uvedeno je nekoliko raznorodnih formalizama, tj. nekoliko raznorodnih sistema izračunavanja.
  - Tjuringove mašine
  - Rekurzivne funkcije
  - Lambda račun
  - Postove mašine
  - Markovljevi algoritmi
  - Neograničena registarska mašina

Normalni algoritmi Markova

- Ovo je algoritamski sistem, izveden 1954. iz Blok Dijagram Algoritma.
- Posmatraju algoritam sa stanovišta izvršavanja.
- Osnovni problem kojim se bave Normalni algoritmi Markova je kako prikazati alfabetski operator G na standardni način.

- Kod zadavanja alfabetskog operatora (transformacije ulazne reči) Markov je uočivo da postoje dve osnovne operacije:
  - operacija obrade i
  - operacija odluke.
- Tako da se Markovljevi algoritmi prikazuju primenom ove dve operacije:





- Postoji i treća operacija **smena** $\alpha \rightarrow \beta$, kada nam je data neka reč, i podreč α želimo da zamenimo sa β, to radimo po sledećim pravilima:
  - Posmatramo reč sleva i kada prvi put primetimo podreč α, na tom mestu je zamenimo.

  Primer:

  pre smene: 49238592349923

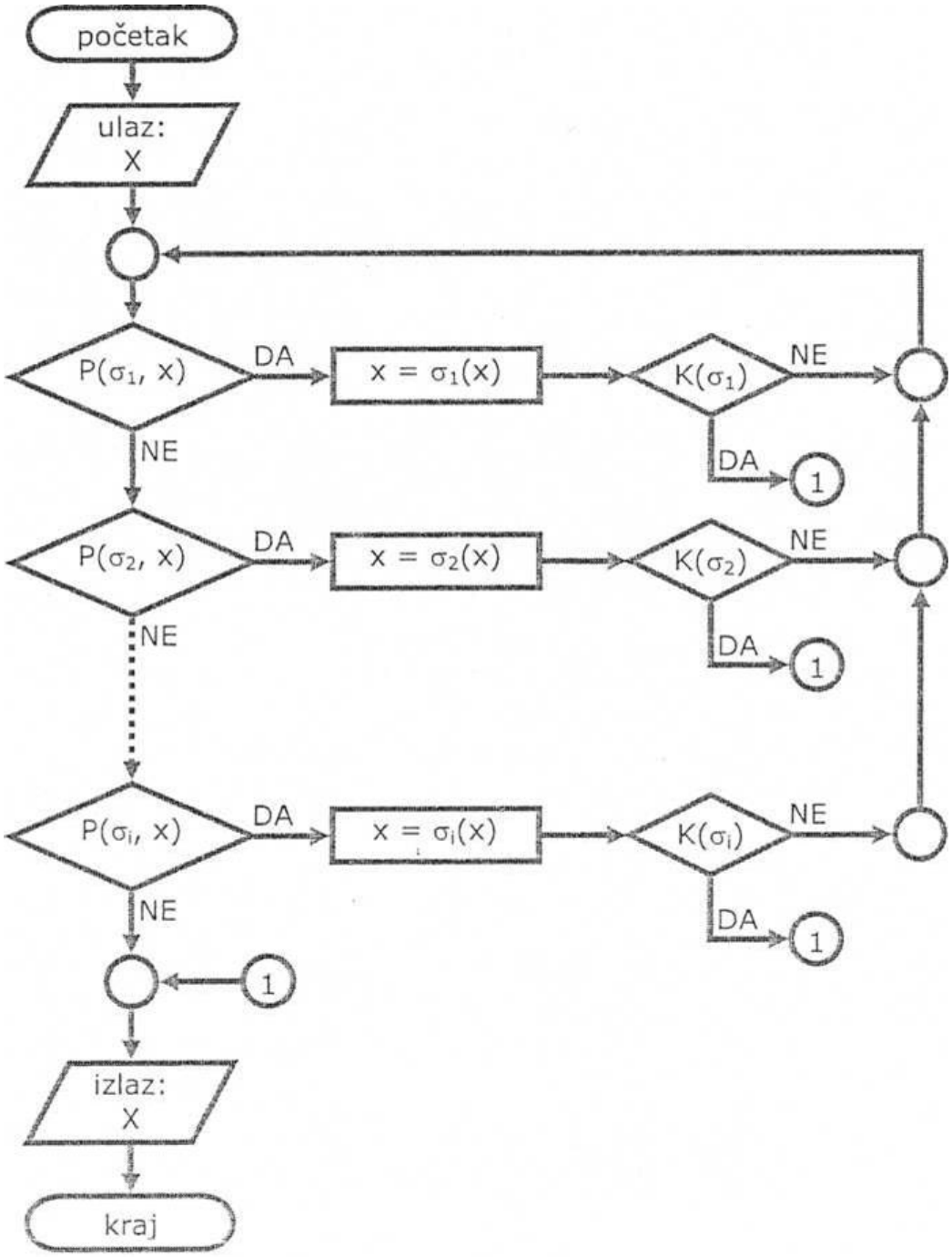
  smena: $923 \rightarrow 0$

  posle smene: 408592349923

- Ako smena ima više, mora se voditi računa o redosledu njihovog izvršavanja.
- Tako podrazumevamo uređen niz smena koje se primenjuju u tekućoj reči.
- Ako podreči α nema, ništa se neće dogoditi.
- U svakom slučaju, mora postojati posebna vrsta nula ili više završnih smena koje se izvršavaju ako ni jedna druga više ne može da se izvrši. Njih označavamo tako što stavimo tačku iza strelice $\alpha\rightarrow.\beta$
- Uobičajena oznaka za prazno siovo je Λ. Proširivanje reči radimo na sledeći način: $\Lambda \rightarrow\beta$, tj. ispred tekuće dodajemo β.

- Normalne algoritme Markova definišemo koristeći sledeće definicije:
  - σ_i je smena;
  - x je reč;
  - $P(\sigma_i, x)$ je predikat, tj. $x = \sigma_i(x)$
  - $K(\sigma_i)$ vraća tačno ako je σ_i završna smena odnosno netačno ako nije.

Opšti oblik Markovljevih normalnili algoritama prikazan je sledećom šemom:



Osobine algoritama

- **Diskretnost** (po definiciji algoritmi su diskretni, nema kontinualnih algoritama)
- **Rezultativnost** (posle konačnog broja koraka se generiše rezultat)
- **Determinisanost** (za iste ulazne podatke daje isti rezultat)
- **Elementarnost** (zakon dobijanja izlaznih veličina mora biti jasan i prost)
- **Masovnost** (algoritam treba da bude primenljiv na mnogo ulaznih podataka)

Kompleksnost algoritma

- Kompleksnost algoritma predstavlja vreme rada algoritma, odnosno broj koraka algoritma koji dovode do traženog rešenja.
- Kako su vreme rada algoritma i broj koraka direktno proporcionalne veličine, nebitno je koja će se od ovih veličina koristiti za definisanje kompleksnosti.
- Vreme rada zavisi i od ulaznih podataka i oni definišu dimenziju problema.
- Kompleksnost algoritma definisana je funkcijom f(n) koja određuje vreme rada algoritma u zavisnosti od dimenzije problema za najnepovoljniji ulazni podatak.
- Kompleksnost algoritma može da bude:
  - konstantna,
  - linearna,
  - polinomijalna,
  - eksponencijalna, ...

Najčešće funkcije kojima se izražava kompleksnost algoritma

- **1** Izračunavanje se izvršava u konstantnom vremenu, bez obzira na ulazni skup.
- **log n** Složenost vrlo sporo raste sa povećanjem dimenzije ulaza. To se dešava u slučajevima kada se veliki problem rešava svođenjem na manji tako što mu se dimenzija konstantnom brzinom smanjuje.
- **n** Za svaki element ulaznog skupa potrebno je neko (konstantno) vreme obrade. Kažemo da je kompleksnost linearna. Ako se ulazni skup udvostruči, i vreme izvršavanja se udvostruči.
- **n log n** Polazni zadatak se obično razbija na dva ili više manjih, sa približno istom dimenzijom. Manji zadaci se zatim na isti način rešavaju. Vreme izvršavanja raste nešto brže od dimenzije.
- n^2 Ovo je složen tip algoritma. Složenost raste mnogo brže od dimenzije. U ovom slučaju se obično obrađuje po par elemenata iz ulaznog skupa.
- n^3 U ovom slučaju se obično obrađuju sve trojke elemenata iz ulaznog skupa. U programu postoje tri koncentrične petlje. Broj izvršavanja svake petlje jednak je dimenziji problema.
- 2^n Eksponencijalna složenost, dešava se u postupku rešavanja, poznatom pod nazivom GRUBA SILA (brute-force). U većini slučajeva izvode se određene radnje nad većinom podskupova ulaznog skupa ili nad permutacijama (kombinacijama) elemenata ulaznog skupa.

Tjuringovu verziju Čerčove teze:

- Problem je **algoritamski rešiv** ako se može rešiti na Tjuringovoj mašini.
- Algoritmom se može smatrati svaki niz instrukcija koji se može izvršiti na Tjuringovoj mašini.
- Tjuringova mašina radi nad **konačnim skupom simbola**.
- Tjuringova mašina je **prebrojiv skup**.
- Skup svih algoritamski rešivih problema je prebrojiv.
- Skup svih problema odlučivanja je neprebrojiv
- **Postoje problemi za koje ne postoje algoritmi**.

Neformalni opis Tjuringove mašine

- *Tjuringova mašina* se sastoji od:
  - *beskonačne trake* koja je podeljena na ćelije; sadržaj svake ćelije može biti 0 ili 1
  - *glave* koja se nalazi nad tačno jednom ćelijom; glava može čitati sadržaj svake ćelije, upisivati 1 ili 0 u ćeliju, pomerati se jednu ćeliju levo ili desno
  - *indikatora stanja*

Neformalni opis Tjuringove mašine

- *Tjuringova mašina* se u svakom trenutku nalazi u jednom od konačno mnogo stanja
- Skup svih stanja se obeležava sa: $S=\{q_0, q_1, ...\}$
- Mašinom upravlja program koji je sačinjen od konačnog niza naredbi oblika: $q_i\ s\ o\ q_j$
  - s je znak nad kojim se nalazi glava
  - o je oznaka operacije

Neformalni opis Tjuringove mašine

- *Operacije* Tjuringove mašine:
  - o = 1, u ćeliju nad kojom se nalazi glava upisuje se 1
  - o = 0, briše se sadržaj ćelije nad kojom se nalazi glava
  - o = L, glava se pomera jednu ćeliju ulevo
  - o = R, glava se pomera jednu ćeliju udesno

Primer

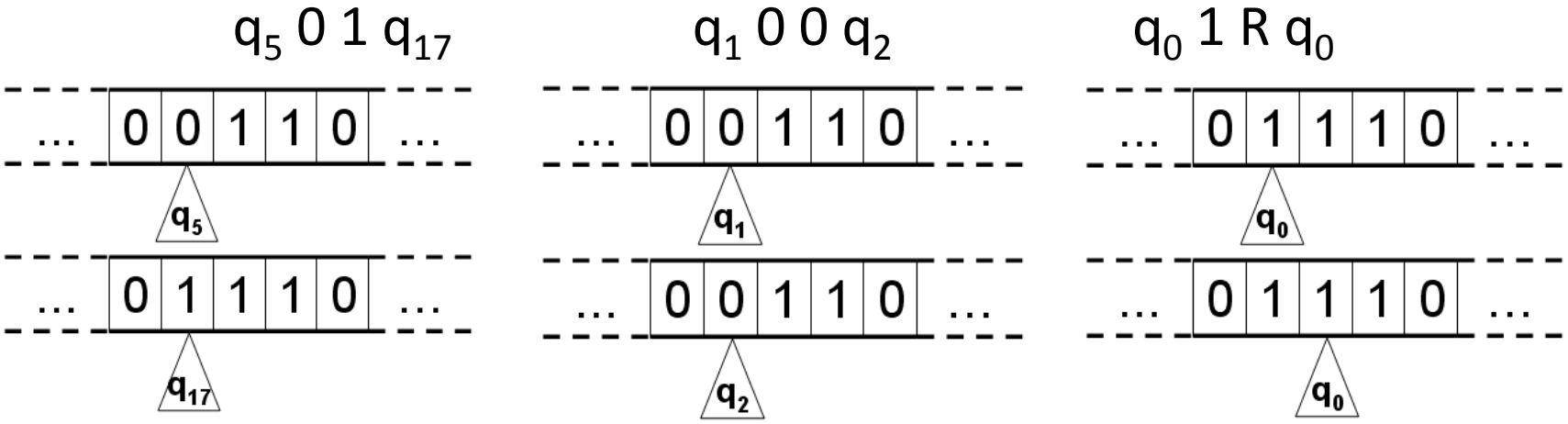
Naredba: q_5 0 1 q_{17} (Ako se mašina nalazi u stanju q_5, a glava nad znakom blanko, u ćeliju se upisuje znak 1 i prelazi u stanje q_{17}.)

Naredba: q_1 0 0 q_2 (Ako se mašina nalazi u stanju q_1, a glava nad znakom blanko, u ćeliju se upisuje blanko znak i prelazi u stanje q_2. Ovakva naredba služi samo za promenu stanja mašine.)

Naredba: q_0 1 L q_0 (Ako se mašina nalazi u stanju q_0, a glava nad znakom 1, glava se pomera ulevo, a mašina ostaje u istom stanju.)

Neformalni opis Tjuringove mašine

- *primer:*



Neformalni opis Tjuringove mašine

- Kada mašina radi **deterministički**, program sme sadržati samo jednu naredbu za svaku kombinaciju stanja q_i i sadržaja s ćelije nad kojom je glava.

Na primer, u jednom programu se ne smeju pojaviti sledeće naredbe:

$$q_4\ 1\ 1\ q_5$$

$$q_4\ 1\ L\ q_2$$

jer im se vrednosti parametara q_i i s poklapaju, a vrednosti parametara o i q_j razlikuju.

- U slučaju **nedeterminističkih** mašina ovaj zahtev ne postoji.

Neformalni opis Tjuringove mašine

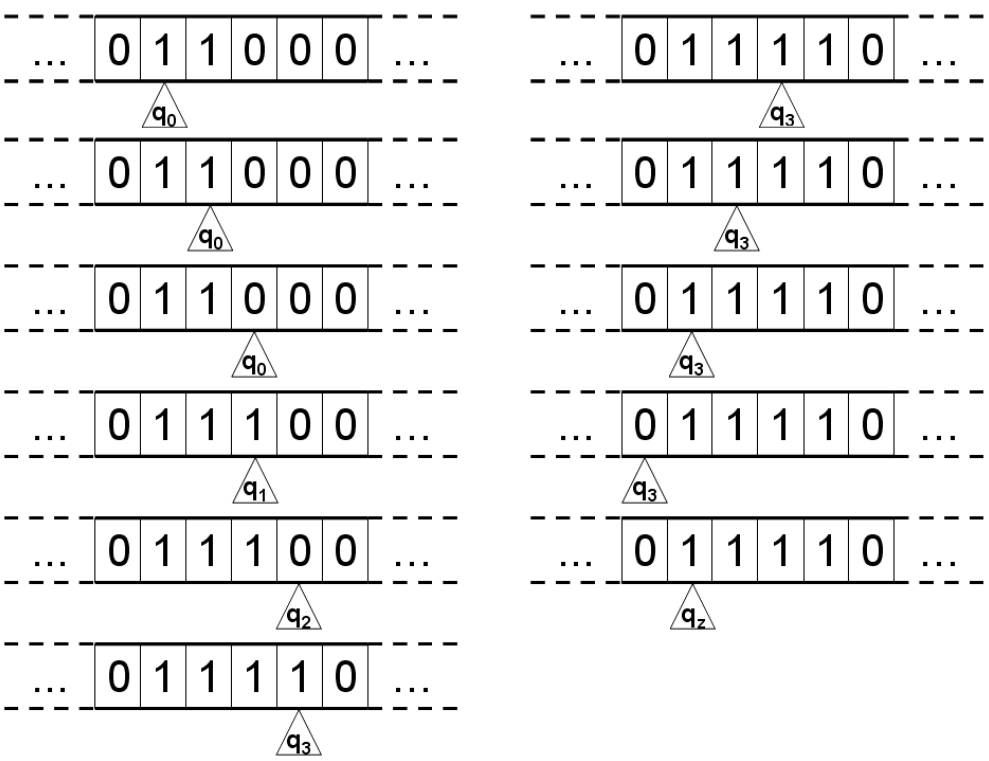
- *Konvencije* Tjuringove mašine:
  - stanje q_0 zovemo *početno stanje*; inicijalno se mašina *uvek* nalazi u početnom stanju
  - traka sadrži *konačno mnogo* ćelija u koje je upisan znak 1; ostale ćelije su prazne (sadrže znak 0)
  - *reč* se na traci prikazuje kao neprekidan niz ćelija koje sadrže znak 1; sa obe strane reči postoji bar po jedan blanko znak
  - na početku i na kraju programa, glava se nalazi iznad prve ćelije koja sadrži znak 1
  - *završno stanje* se obeležava sa q_z

Neformalni opis Tjuringove mašine

- *Konfiguracija* Tjuringove mašine:
  - opis sadržaja trake, položaja glave, stanja mašine
- *Standardna konfiguracija:*
  - traka je prazna ili sadrži konačno mnogo nepraznih reči razdvojenih sa po jednim blanko simbolom
  - glava je iznad prve ćelije trake koja sadrži znak 1
  - na početku izvršavanja mašina se nalazi u stanju q_0
  - mašina prestaje sa radom kada dođe u stanje q_z

Primer

- $q_0\ 1\ R\ q_0$
- $q_0\ 0\ 1\ q_1$
- $q_1\ 1\ R\ q_2$
- $q_2\ 0\ 1\ q_3$
- $q_3\ 1\ L\ q_3$
- $q_3\ 0\ R\ q_z$



Neformalni opis Tjuringove mašine

- Šta se dešava ukoliko se glava nađe iznad ćelije za čiji sadržaj ne postoji naredba? (analogno zaglavljivanju programa pisanog u konvencijalnom programskom jeziku)
- Moguće je program dopuniti naredbama koje ne menjaju ni stanje, ni poziciju glave, ni sadržaj ćelije
- *primer:* $q_0\ 0\ 0\ q_0$
- Ova naredba predstavlja *beskonačnu petlju*

Rekurzivni algoritmi

- Algoritam je ***rekurzivni*** ako se rešavanje problema svodi na predhodni korak, jednostavnijeg ulaza.
- Da bi se algoritam koji koristi rekurziju završio mora se predvideti uslov izlaska, odnosno uslov završetka.
- Rekurzivni algoritam zahteva jednu ili više ulaznih veličina, a vraća jednu sračunatu. Ta vrednost je iz koraka u korak sve bliža željenoj, iskazanoj u uslovu izlaska.
- Algoritam u sebi sadrži naredbu ***if*** koja testira uslov izlaska i naredbu ***else*** kojom se rekurzivno poziva sama funkcija, odnosno algoritam.

Primer:

- Rekurzivni algoritam za izračunavanje stepena

$$procedura: stepen(a \ \ je\ realan\ broj, n \ \ je\ nenegativan\ broj)$$
$$if \ \ n = 0\ then\ stepen(a,n) = 1$$
$$else\ stepen(a,n) = a \cdot stepen(a,n-1)$$

Algoritamske šeme

- Algoritamske šeme mogu se podeliti u dve kategorije:
  - ***Linijske algoritamske šeme,***
  - ***Ciklične algoritamske šeme***

- ***Linijske algoritamske šeme*** su one šeme kod kojih se svaki algoritamski korak izvršava najviše jedanput u toku izvršavanja algoritma.

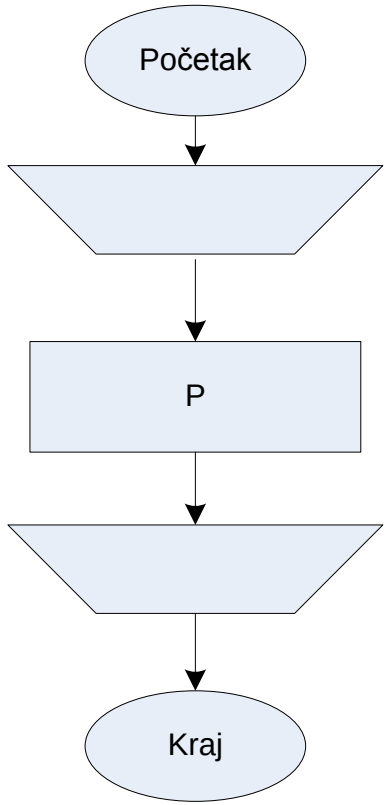
  Mogu biti ***proste*** i ***razgranate***.

- ***Proste linijske algoritamske šeme***, su one šeme kod kojih se svaki algoritamski korak izvršava tačno jednput u toku izvršavanja algoritma.

- ***Razgranate linijske algoritamske šeme***, su one šeme kod kojih se svaki korak izvršava tačno jedanput i obavezno sadrži bar jedan uslovni algoritamski korak. Ako je uslov ispunjen, izlaz iz algoritamskog koraka biće označen sa *da,* a ako uslov nije ispunjen izlaz će biti označen sa *ne***.**

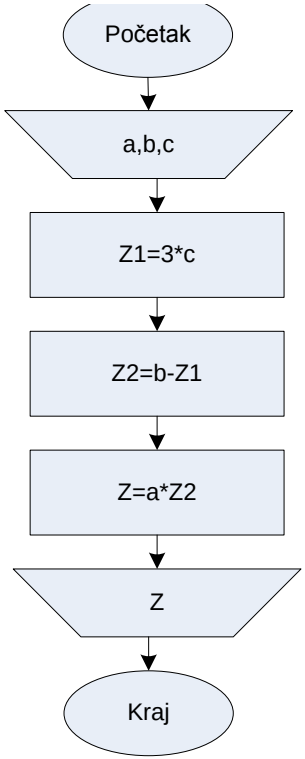
Linijska šema

- Grafički prikaz proste linijske šeme dat je na sledećoj slici.



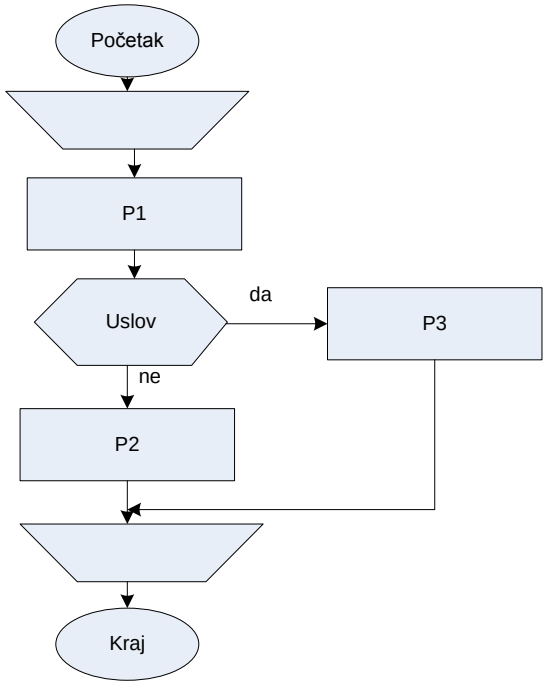
Primer

- **Primer**: Sastaviti algoritamsku šemu za izračunavanje izraza Z=a(b-3c)



Razgranata linijska šema

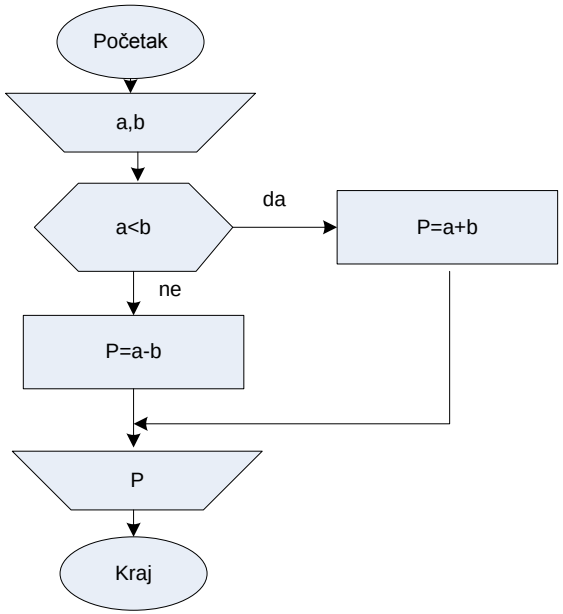
- Grafički prikaz razgranate linijske šeme dat je na sledećoj slici.



Primer

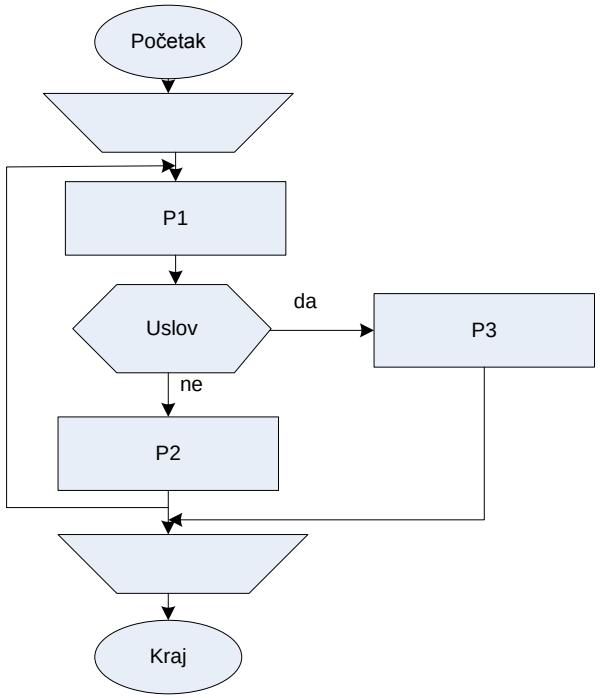
- **Primer**:
  Sastaviti algoritam za računanje vrednosti

$$Z = \begin{cases} a+b, a<b \\ a-b, a \geq b \end{cases}$$



Ciklične algoritamske šeme

- ***Ciklične algoritamske šeme*** su one šeme u kojima se jedan ili više algoritamskih koraka može izvršavati više od jedanput u toku izvršavanja algoritma. Ovi koraci čine ***ciklus.*** Ukoliko je uslov ispunjen izlazi se iz ciklusa, u suprotnom ciklus se ponavlja.
- Uslov za izlazak iz ciklusa zove se ***izlazni kriterijum ciklusa.***
- Ciklične algoritamske šeme mogu biti ***konstantne*** i ***promenljive*****.**
- ***Konstantne ciklične šeme*** su šeme kod kojih se zakon obrade tokom ciklusa ne menja, dok se kod ***promenljivih*** menja.

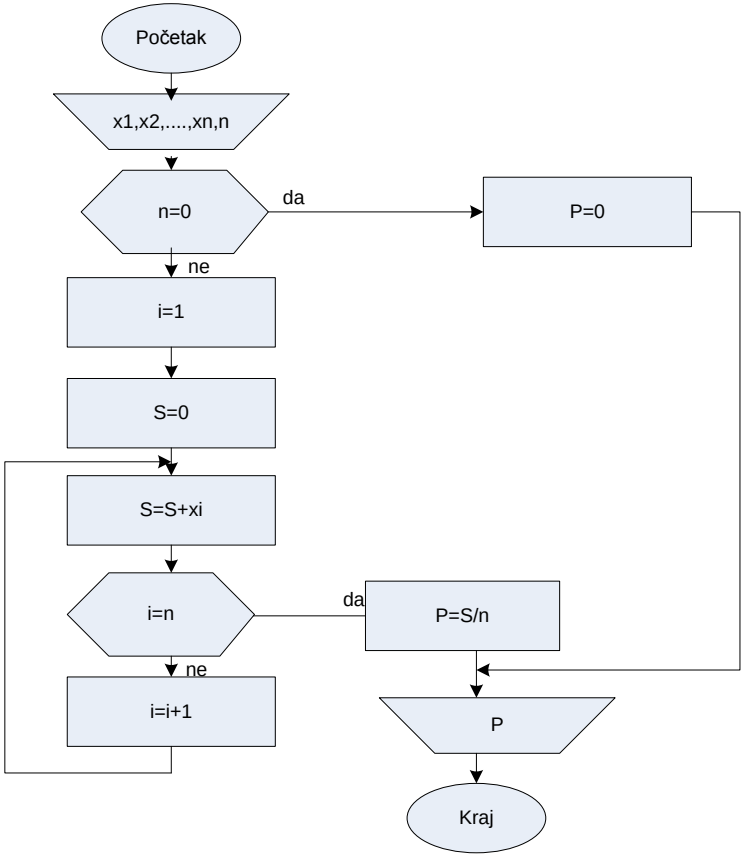


Primer

- **Primer**:

  Sastaviti algoritam koji za poznato *n* izračunava aritmetičku sredinu zadatih brojeva

$x_1, x_1, \ldots, x_n$ $\qquad P = \frac{x_1 + x_1 + \ldots + x_n}{n}$



Ekvivalentni algoritmi

- Za rešavanje jednog istog zadatka može se sastaviti više algoritama različitih struktura.
- Za ovakve algoritme kaže se da su ***ekvivalentni.***
- Među ekvivalentnim algoritmima treba izabrati onaj koji najefikasnije dovodi do rezultata.
- Kriterijumi za izbor najefikasnijeg algoritma su različiti:

1. najveća brzina izvršavanja algoritma,
2. minimalno angažovanje memorijskog prostora,
3. minimalno vreme koje je potrebno za izvršavanje algoritma,
4. što jednostavnija struktura i td,

Apstraktni tipovi podataka

- Struktura podataka i skup operacija koji se na njoj mogu izvrsiti.
- Klasa u objektno-orijentisano programiranju je apstraktni tip podataka.
- Klase imaju dodatna svojstva (nasledjivanje i polimorfizam) koje se ne razmatraju kod apstraktnih tipova podataka.

Skupovi objekata

- Programi cesto barataju sa skupovima objekata.
- Ovi skupovi mogu biti organizovani na mnogo nacina i mogu se koristiti razlicite programske strukture za njihovo predstavljanje, ali ipak, sa apstraktne tacke gledista, postoji nekoliko zajednickih operacija za bilo koji skup.

Primer zajedničkih operacija za bilo koji skup

- **create** Kreiranje novog skupa
- **add** Dodavanja objekta skupu
- **delete** Brisanje objekta iz skupa
- **find** Trazenje objekta koji zadovoljava odredjeni uslov u skupu
- **destroy** Brisanje skupa

Konstruktori i destruktori

- Metodi create i destroy - cesto zvani konstruktori i destruktori - obicno se implementiraju za sve apstraktne tipove podataka.
- S vremena na vreme, uslovi koriscenja ili semantika tipa podataka su takvi da postoji samo jedan objekat tog tipa u programu.
- U tom slucaju, moguce je sakriti od korisnika čak i referencu na objekat.
- Medjutim, čak i u ovim slucajevima, konstruktor i destruktor metodi se često navode.