IV-Osnovne algoritamske strukture

- Algoritam predstavlja uređen niz precizno formulisanih pravila koja su primenjena na unapred definisani skup polaznih podataka i kojima se u konačnom vremenskom periodu rešava jedan ili čitava klasa problema
- Algoritam uvek tačno definiše pravilan redosled obavljanja akcija nad podacima i on uvek treba da dovode do traženih rezultata
- U procesu programiranja, skup akcija definisan je mogućnostima računara, odnosno naredbama korišćenog prog. jezika, dok se redosled izvršavanja akcija zadaje pomoću algoritamskih/programskih struktura
- Svaki algoritam se može opisati na prirodnom jeziku ljudi tekstualnim i grafičkim simbolima u obliku **dijagrama toka** ili **blok šeme**.

Primer: ***Napisati algoritam za izračunavanje cene taksi usluge***.

1. Definisati ulazne veličine: cenu "starta" **S**, cenu po pređenom kilometru **C**, broj pređenih km **L** i popust u procentima **P**.
2. Izračunati cenu bez popusta T1 kao **T1=S+C*L**
3. Izračunati cenu sa popustom T2 kao **T2=T1-T1*P/100**
4. Prikazati izlazne veličine: cenu bez popusta **T1** i cenu sa popustom **T2**

IV – Osobine dobrog algoritma

1. Diskretnost

- Svaki algoritam se sastoji od konačnog broja posebnih koraka (algoritamski koraci)
- Svaki korak može zahtevati obavljanje jedne ili više operacija
- U zavisnosti od toga koje operacije računar može da obavi, uvode se ograničenja za tip operacija koje se u algoritmu mogu koristiti (najčešće +, -, *, /)

2. Determinisanost

- Svaki algoritamski korak mora biti strogo definisan i potpuno jasan (izraz izračunaj **5/0** ili "dodaj **6** ili **7 x**" nisu dozvoljeni)
- Posle izvršenja tekućeg algoritamskog koraka mora biti jednoznačno definisano koji je sledeći korak

3. Efektivnost (konačnost)

- Sve operacije koje se javljaju u jednom algoritamskom koraku moraju se izvršiti za **konačno** (razumno kratko) **vreme**
- Vreme izvršenja celog algoritma mora biti konačno, tj. razumno dugo

IV-Osobine dobrog algoritma

4. Rezultativnost

- Svaki algoritam mora posle konačnog broja koraka generisati traženi rezultat koji uvek pod istim uslovima mora de je isti
- Algoritam može imati nijedan ili više ulaznih podataka i može generisti jedan ili više izlaznih rezultata

5. Masovnost

- Svaki algoritam definiše postupak za rešavanje klase problema, a ne pojedinačnog slučaja
- Treba omogućiti rešavanje sistema jednačina bilo kog reda, množenje matrica proizvoljnog reda, ...

6. Optimalnost

- Svaki algoritam bi trebalo da teži optimalnom rešavanju problema

IV-Načini predstavljanja algoritama

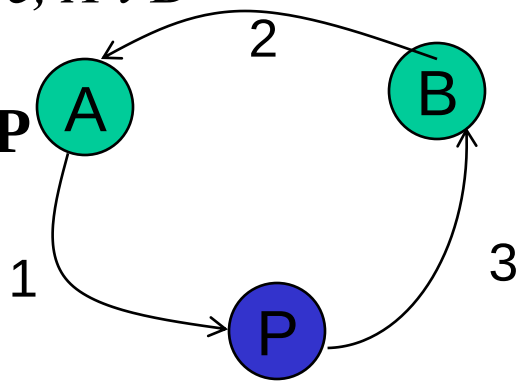
1. Tekstualni

- ✓ Koriste se precizne rečenice govornog jezika
- ✓ Koristi se za lica koja se prvi put sreću sa pojmom algoritma
- ✓ **Dobra osobina**: razumljivost za širi krug ljudi
- ✓ **Loše**: nepreciznost koja proističe iz nepreciznosti samog jezika

Primer 1: *Zameniti sadržaje dve memorijske lokacije, A i B*

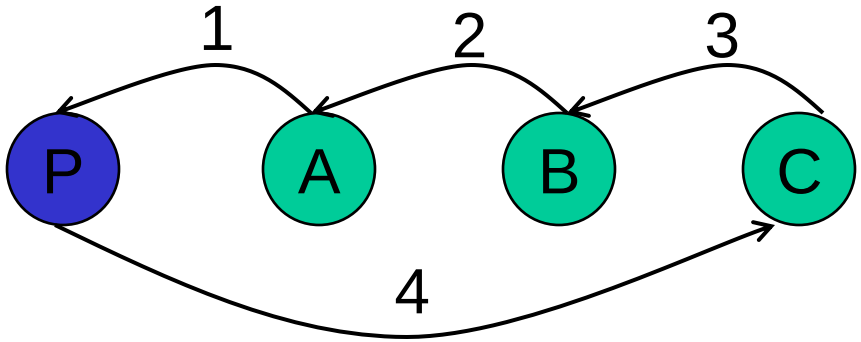
Rešenje – potrebna je treća, pomoćna, lokacija

1. sadržaj lokacije **A** zapamtiti u pomoćnu lokaciju, **P**
2. sadržaj lokacije **B** zapamtiti u lokaciju **A**
3. sadržaj lokacije **P** zapamtiti u lokaciju **B**
4. Kraj



Primer 2: *Ciklički pomeriti u levo sadržaje lokacija A, B i C*

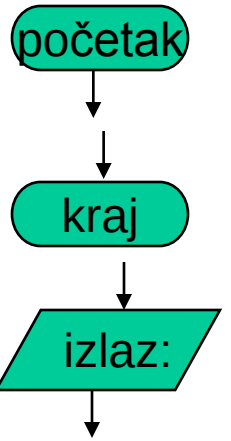
1. iz A u P
2. iz B u A
3. iz C u B
4. iz P u C
5. kraj

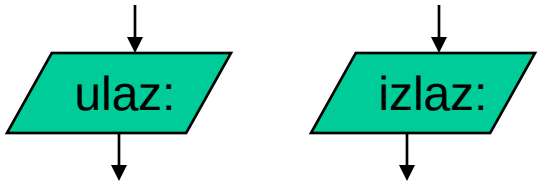


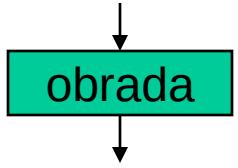
IV-Načini predstavljanja algoritama

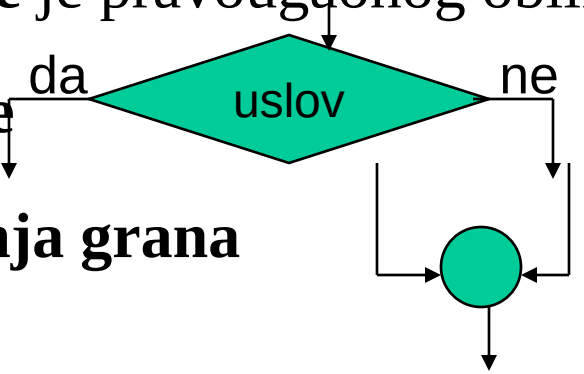
2. Grafički (pomoću blok šema)

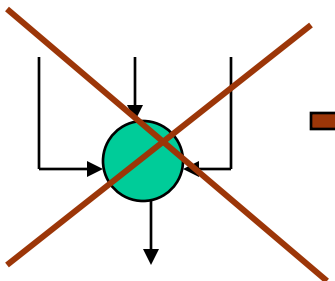
- Određeni grafički simboli predstavljaju pojedine aktivnosti u algoritmu
- Algoritam se predstavlja **usmerenim grafom**
- Čvorovi grafa predstavljaju aktivnosti koje se obavljaju u algoritmu a potezi ukazuju na sledeću aktivnost koja treba da se obavi
- **Polazni čvor** u usmerenom grafu koji predstavlja algoritam nema dolaznih grana, a ima samo jednu izlaznu granu
- **Krajnji čvor** u grafu koji predstavlja algoritam nema izlaznih grana, a ima samo jednu dolaznu granu
- **Blok oblika romboida** koristi se za označavanje ulaznih i izlaznih aktivnosti
- **Blok obrade** je pravougaonog oblika
- **Blok odluke**
- **Blok spajanja grana**

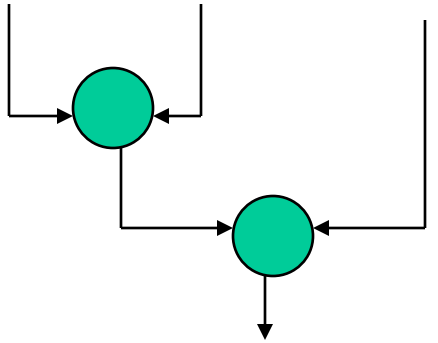












IV-Načini predstavljanja algoritama

3. Pseudo kodom

- Koristi se tekstualni način dopunjen formalizmom
- Svaka od osnovnih algoritamskih struktura se predstavlja na tačno definisani način:

1. **Sekvenca** se predstavlja kao niz naredbi dodeljivanja odvojenih simbolom **;** (**a:=5; b:=a*b; c:=b-a;**)
2. **Alternacija**

```
if (uslov) then
   niz_naredbi
else
   niz_naredbi
endif;
```

```
if (uslov) then
   niz_naredbi
endif;
```

3. **Petlje**

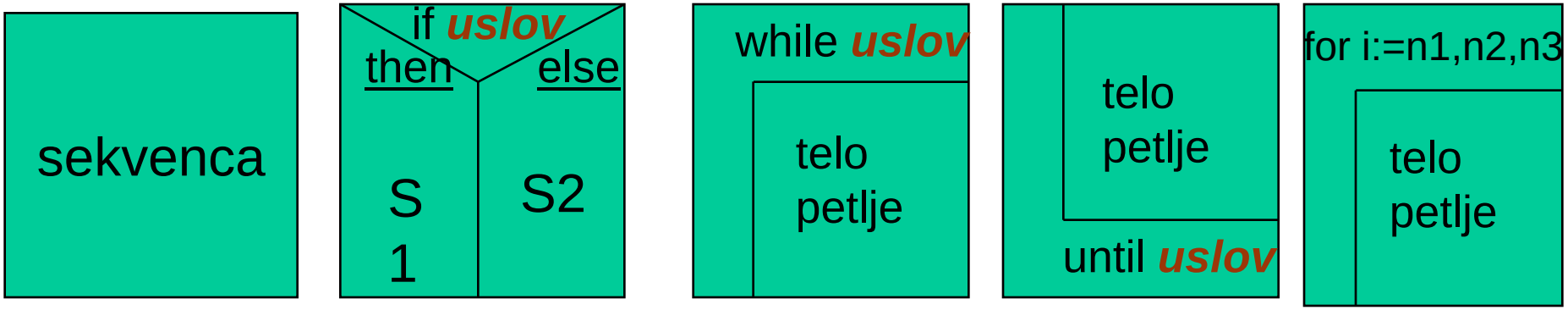
```
while (uslov) do
   niz_ naredbi
enddo;
```

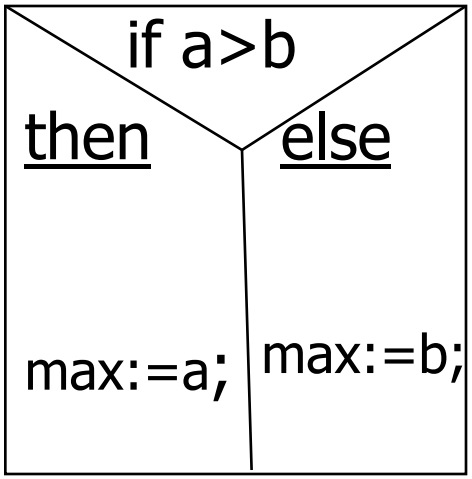
```
repeat
   niz_naredbi
until (uslov);
```

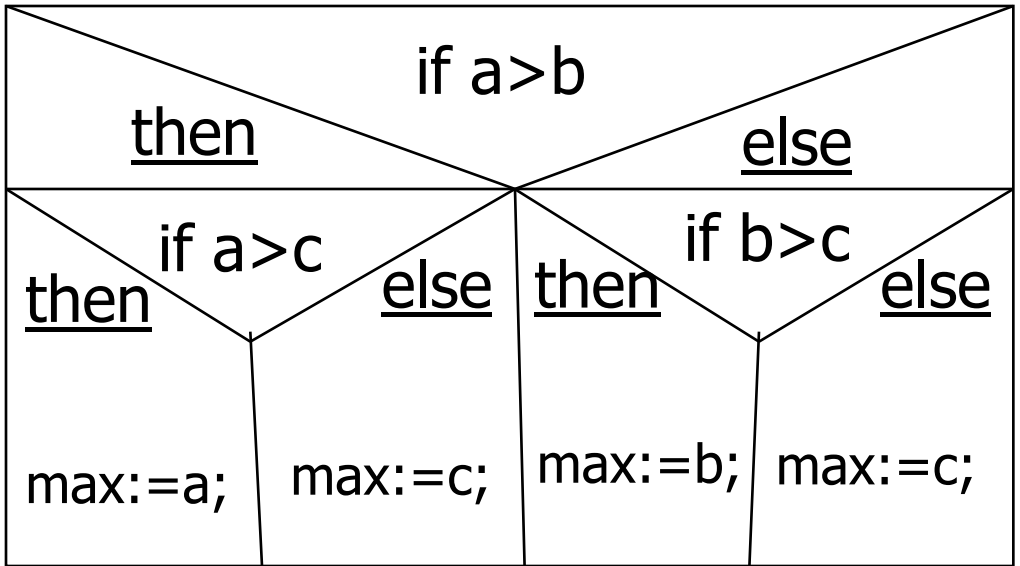
IV-Načini predstavljanja algoritama

4. Strukturogram

- Kombinacija grafičkog i pseudo koda
- Koriste se kao prikladna dokumentacija za već završene programe.
- Program se piše tako da se popunjavaju određene geometrijske slike

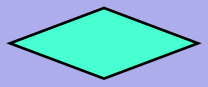
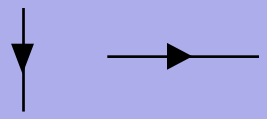






IV – Osnovni algoritamski simboli

- S obzirom da je ovakav način opisivanja algoritma teško čitljiv u slučajevima složenijih problema, znatno češće se koristi grafički prikaz pomoću takozvane **algoritamske šeme**.
- Algoritamska šema se može sastojati od sledećih grafičkih simbola:

Grafički simbol	*Značenje*
	Terminator (početak ili kraj programa)
	Unošenje podataka (ulazni podaci)
	Izdavanje podataka (izlazni podaci)
	Obrada podataka (izračunavanje)
	Uslovno grananje (odluka da ili ne)
	Povezivanje algoritamskih koraka

IV-Osnovne algoritamske strukture

- Kombinovanjem gradivnih blokova dobijaju se tri osnovne algoritamske strukture:

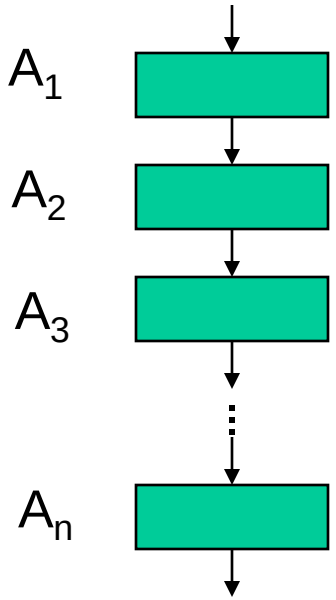
1. **Linijska** (Sekvenca) - sve akcije se izvršavaju tačno jednom u redosledu u kome su navedene.
2. **Razgranata** (Selekcija ili Alternacija) - struktura algoritma u kojoj tok operacija zavisi od ispunjenosti nekih uslova. Ona omogućuje da se od više grupa akcija, koje se nalaze u različitim granama razgranate strukture, izabere ona koja će se izvršiti jednom, dok se sve ostale grupe akcija neće izvršitini nijednom
3. **Ciklična** (Iteracija ili Petlja) - algoritam kod kog se određeni broj algoritamskih koraka ponavlja više puta. Ako je broj ponavljanja dela algoritma poznat unapred struktura je konstantna (brojački ciklus). Ako broj ponavljanja nije poznat unapred, nego zavisi od ispunjenosti nekog uslova struktura je promenljiva (uslovni ciklus).

Algoritamsko rešenje bilo kog problema može se uvek zapisati korišćenjem samo ove tri strukture

IV – Linijska struktura (Selekcija)

- Linijska struktura koja se dobija kaskadnim povezivanjem blokova obrade

- Algoritamski koraci se izvršavaju redom, jedan za drugim
- Algoritamski korak A_i, i=2,...,n ne može da otpočne sa izvršenjem dok se ne završi korak A_{i-1}
- sekvenca predstavlja niz naredbi dodeljivanja (:=)
- oblik naredbe:



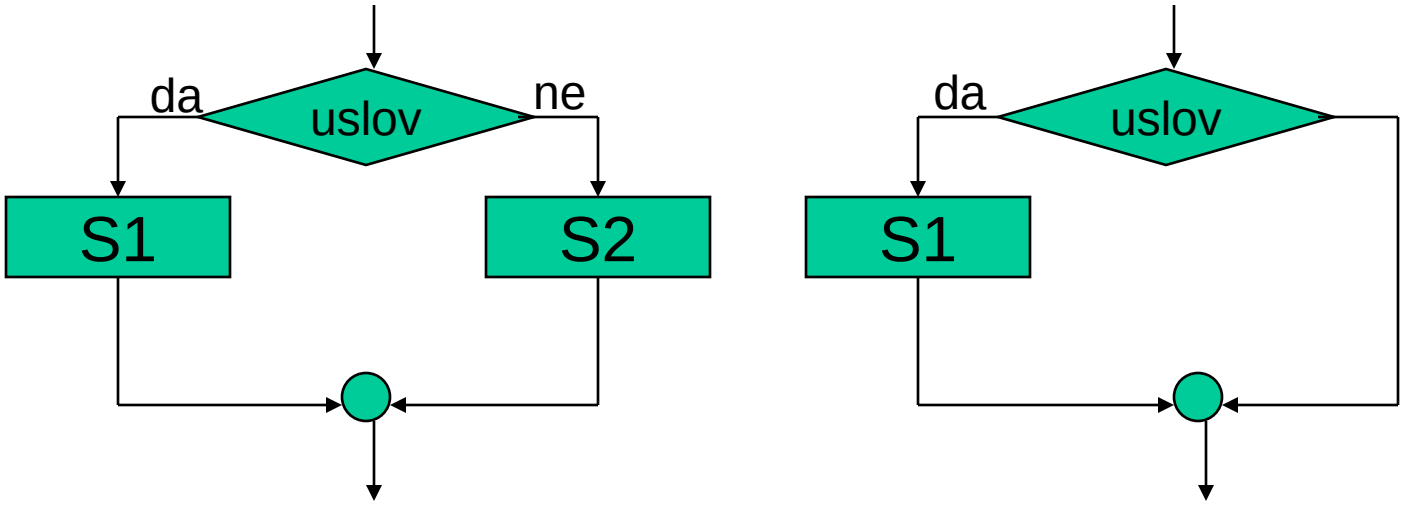
promenljiva:=vrednost

a:=b

n:=n+1

IV-Razgranata struktura (Sekvenca)

- Omogućava uslovno izvršenje niza algoritamskih koraka

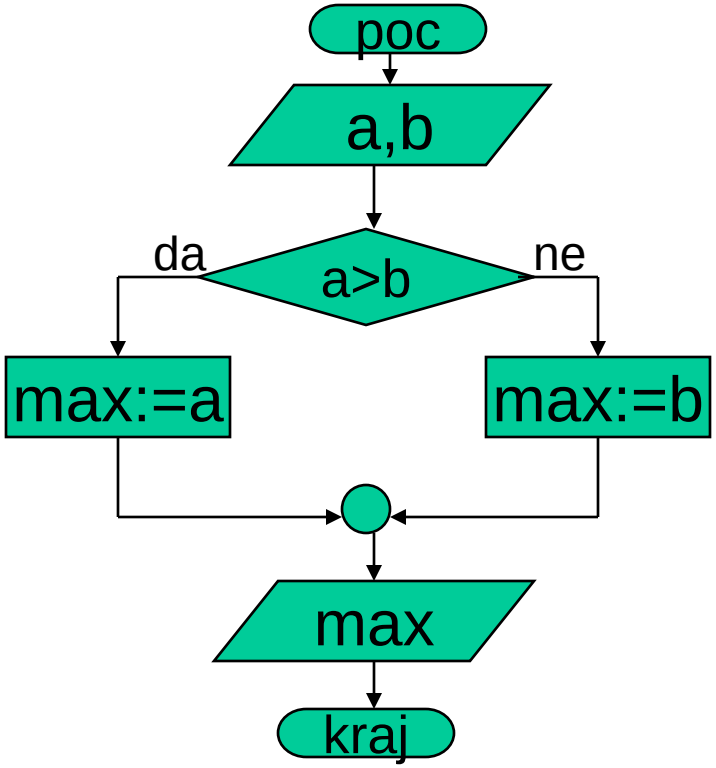


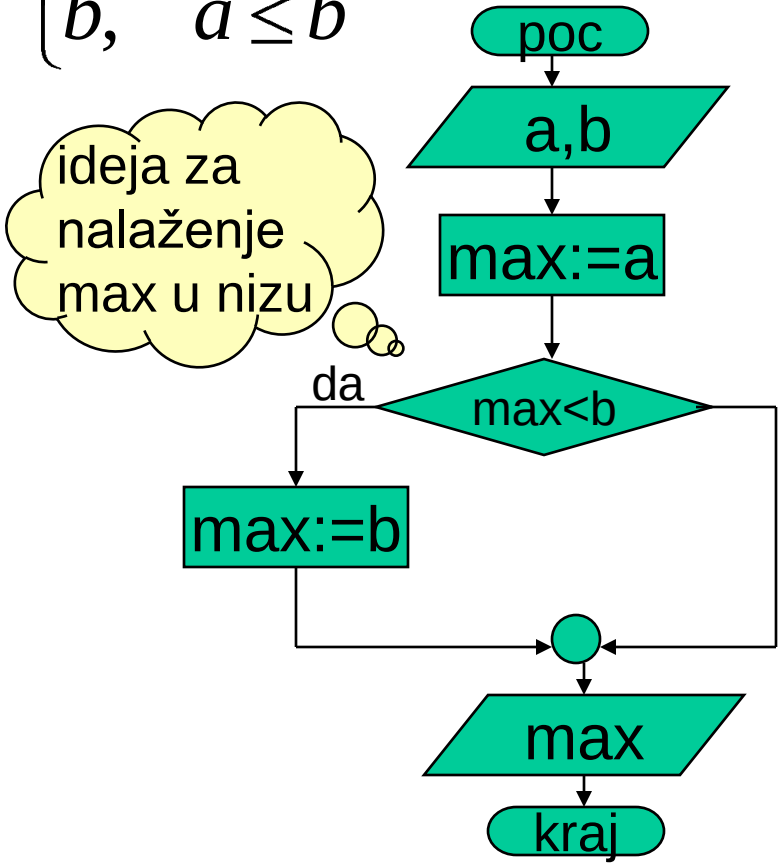
- Blokovi označeni sa S1 i S2 mogu sadržati bilo koju kombinaciju osnovnih algoritamskih struktura.

IV-Razgranata struktura (Sekvenca)

Primer 1: *Nacrtati dijagram toka algoritama kojim se određuje veći od dva zadata broja korišćenjem formule*

$$\max\{a,b\} = \begin{cases} a, & a > b \\ b, & a \le b \end{cases}$$

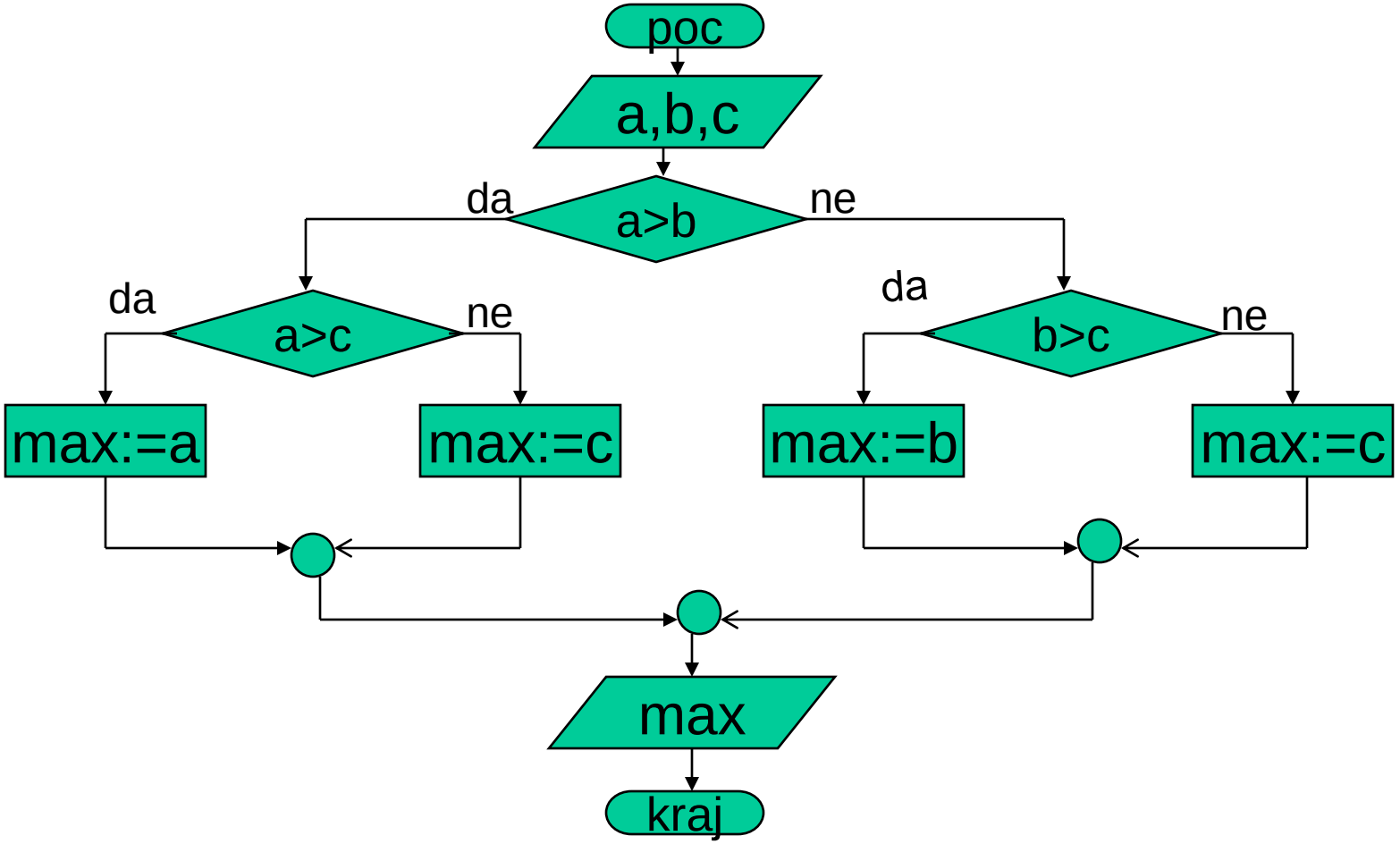




IV-Razgranata struktura (Sekvenca)

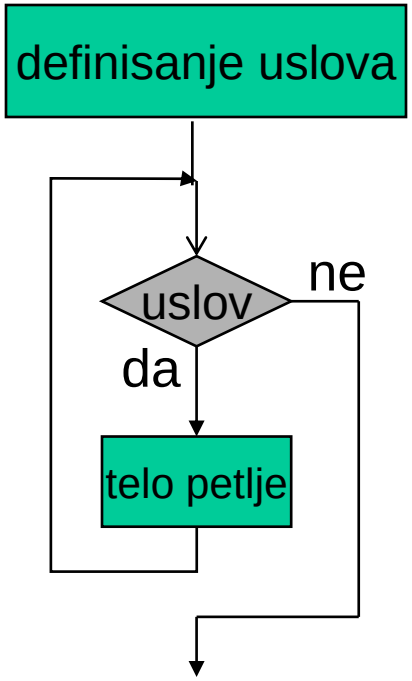
Primer 2: *Naći maksimum od tri zadata broja a, b i c*

$$\max\{a,b,c\} = \max\{\max\{a,b\},c\}$$

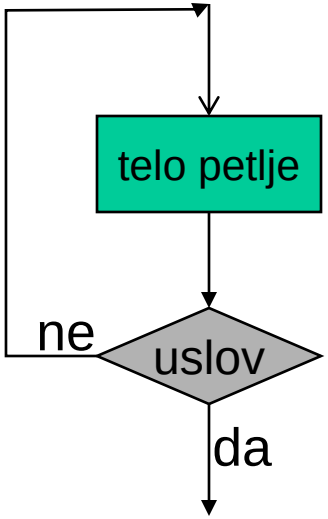


IV – Ciklična struktura (Iteracija)

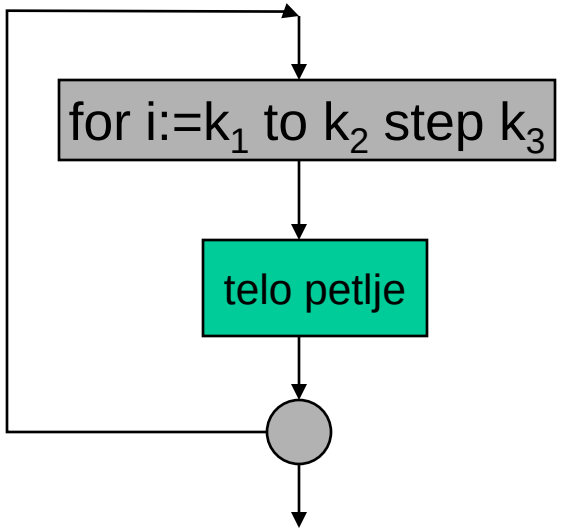
- Omogućava da se algoritamski koraci ponavljaju više puta.



while-do



repeat-until



brojačka petlja

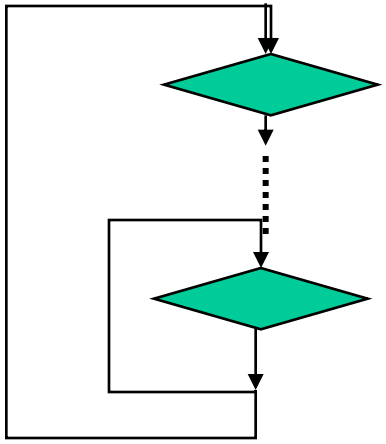
broj prolaza $n = \left[\frac{k_2 - k_1}{k_3} \right] + 1$

IV – Ciklična struktura (Iteracija)

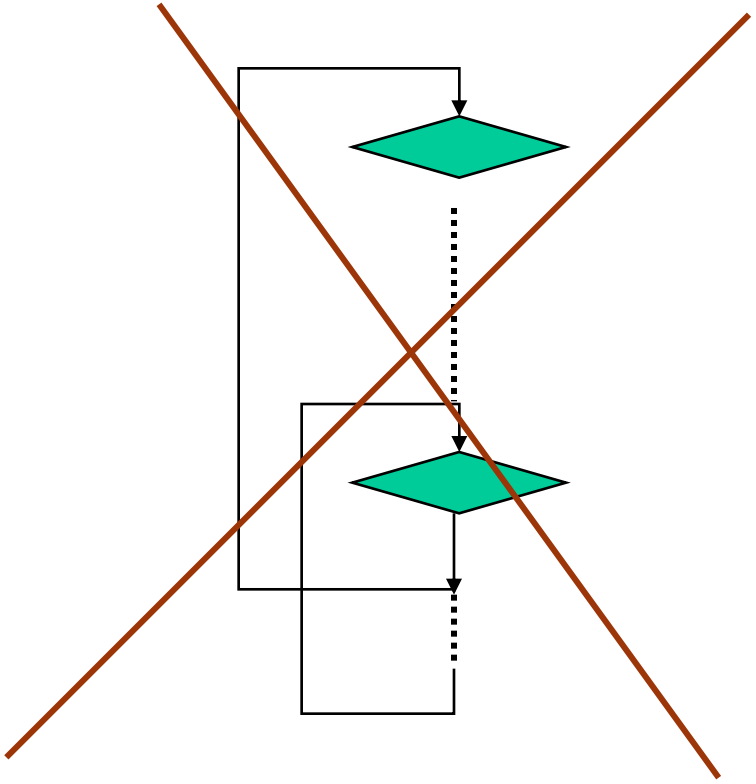
- Kod petlje tipa ***while-do*** telo petlje se izvršava sve dok je uslov ispunjen
- Pre ulaska u petlju potebno je definisati početni uslov.
- Telo petlje ne mora da se izvrši ni jednom (ako uslov nije zadovoljen).
- Kod petlje tipa ***repeat-until*** telo petlje se izvršava sve dok uslov nije zadovoljen tj. telo petlje će se izvršiti bar jednom.
- Kod obe petlje u okviru tela petlje mora postojati naredba kojom se menja uslov da bi se omogućio izlazak iz petlje (u protivnom bi se petlja beskonačno izvršavala).
- Broj prolaza kroz petlju u oba slučaja nije unapred poznat (zavisi kada će se ispuniti uslov za izlazak iz petlje)
- Kod petlje brojačkog tipa, broj prolaza kroz telo petlje je definisan u trenutku kada otpočne njeno izvršenje.
- Parametri petlje, indeks ***i***, donja granica **k1**, gornja granica **k2**, i korak promene indeksa **k3**, se ne smeju menjati u okviru petlje!

IV – Pravila korišćenja algoritama

- Ako petlja počne unutar **then** bloka ili **else** bloka, u tom bloku se mora i završiti!
- Dozvoljene su paralelne (ugnježdene) petlje.
- Nisu dozvoljene petlje koje se seku!



Paralelne petlje



petlje koje se seku

IV – Kontrola toka programa

- U teoriji programiranja definisana su tri osnovna tipa programskih struktura: **sekvenca**, **grananja** i **programske petlje**.
- Svaka od ovih struktura se realizuje sa posebnim strukturnim naredbama C jezika:
  1. **sekvenca** se implementira korišćenjem **bloka**,
  2. **grananja** - naredbama uslovnog skoka (**if** i **switch**) i naredbama bezuslovnog skoka (**goto**, **break** i **continue**),
  3. **programske petlje** naredbama **for**, **while** i **do-while**.

1. Blok naredbi

- Skup programskih naredbi koje se izvršavaju onim redosledom kako su zapisane.
- Blok uvek počinje simbolom **{** i završava se simbolom **}**
- Na početku bloka mogu biti navedeni opisi promenljivih i konstanti koje su lokalne za blok.

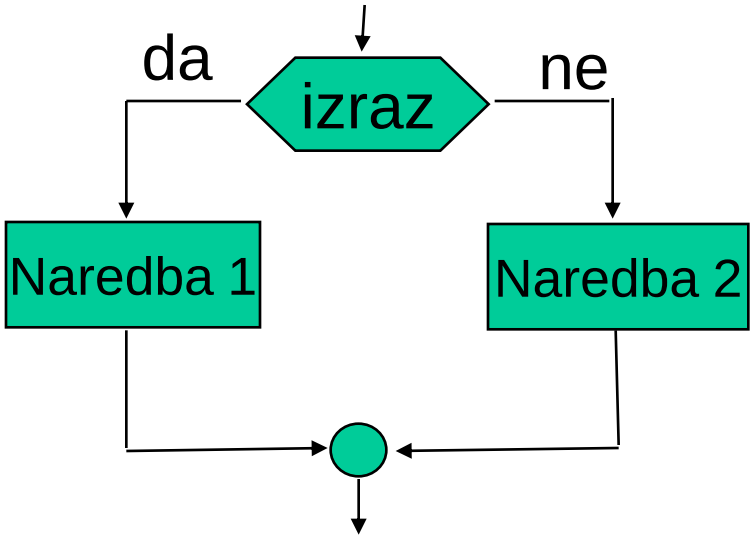
```
{
   <naredba1>
   <naredba2>
   …
   <naredbaN>
}
```

IV – Kontrola toka programa

2. Naredbe uslovnog grananja

❑ If

- *If*-naredbom se implementira osnovi tip selekcije (grananja) kojom se vrši uslovno izvršenje jedne od dve moguće naredbe.
- Standardni dijagram toka strukture grananja



Ekvivalentan kod u C-u je:

```
if (<izraz>)
   <naredba1>
else
   <naredba2>
```

- **Dejstvo:** kada je vrednost izraza:
  - različita od nule (**true**), izvršava se naredba <naredba1>,
  - nula (**false**), izvršava se naredba <naredba2>.

IV - Kontrola toka programa

Zadatak: ***Izračunavanje vrednosti funkcije***

Napisati program na C-u za izračunavanje vrednosti funkcije ***y = 1 / x***

Rešenje:

Funkcija **y = 1/x** nije definisana za **x=0**. Zbog toga se u rešenju koristi struktura grananja kojom se odlučuje da li se vrednost funkcije računa ili ne.

```
#include <stdio.h>
main()
{
   float x;
   printf("Unesite vrednost argumenta x \n");
   scanf("%f",&x);
   if ( x == 0 )
      printf("Za x=0 funkcija y=1/x nije definisana\n");
   else
      printf( "y=%f\n", 1/x );
}
```

IV – Kontrola toka programa

Zadatak: *Napisati program na C-u za izračunavanje vrednosti funkcije:*

$$y = \begin{cases} x, & \text{za } x < 2 \\ 2, & \text{za } 2 \le x < 3 \\ x - 1 & \text{za } x \ge 3 \end{cases}$$

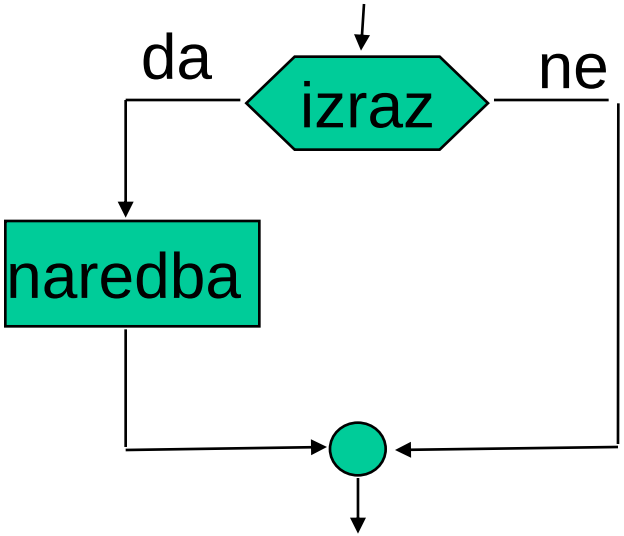
Rešenje:

```
#include <stdio.h>
main()
{
        float x,y;
        printf("Unesite vrednost argumenta x \n");
        scanf("%f",&x);
        if ( x < 2 )
                y = x;
        else if ( x < 3 )
                y = 2;
        else
                y = x – 1;
        printf( "y=%f\n", y );
}
```

IV – Kontrola toka programa

❑ Skraćeni oblik If naredbe

- Selekcijom se može implementirati i odluka da li će se jedna naredba u programu izvršiti ili ne.
- Dijagram toka strukture skraćenog grananja



- Ekvivalentan kod u C-u je:

```
if (<izraz>)
    <naredba>
```

Dejstvo:

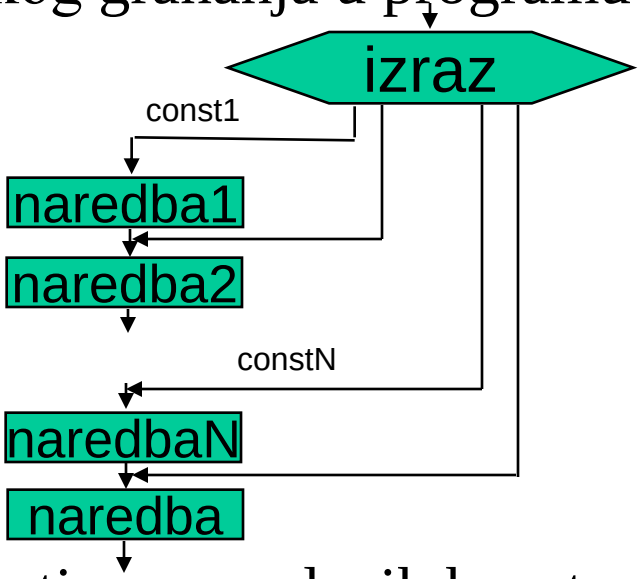
- Kada je vrednost izraza:
  - različita od nule (**true**), izvršava se navedena <**naredba**>
  - jednaka nuli (**false**), ne izvršava se ni jedna naredba.

IV – Kontrola toka programa

❑ Switch

➢ Jedan od načina za realizaciju višestrukog grananja u programu

```
switch (<izraz>)
{
   case <const1>: <naredba1>
   case <const2>: <naredba2>
   ...
   case <constN>: <naredbaN>
   [default: <naredba>]
}
```



➢ Vrednost izraza upoređuje se sa vrednostima navedenih konstanti

➢ Ukoliko se vrednost izraza poklopi sa nekom od konstanti izvršiće se niz naredbi iz *switch*-strukture od naredbe obeležene tom konstantom do kraja strukture.

➢ Ukoliko se vrednost izraza ne poklapa ni sa jednom od ponuđenih vrednosti izvršiće se samo naredba navedena u *default*-delu (ukoliko *default*-deo postoji).

➢ Izraz (kao i konstante **const1**,…,**constN**) mogu biti tipa int ili char.

IV - Kontrola toka programa

Zadatak: *Napisati program na C-u za štampanje imena meseci u godini počev od tekućeg.*

Rešenje:

```
#include <stdio.h>
main()
{
        int tekuci;
        printf("unesite redni br. tekuceg meseca\n");
        scanf("%d",&tekuci);
        switch( tekuci )
        {
                case 1: printf("januar\n");
                case 2: printf("februar\n");
                case 3: printf("mart\n");
                case 4: printf{"april\n");
                case 5: printf("maj\n");
                case 6: printf("jun\n");
                case 7: printf{"jul\n");
                case 8: printf("avgust\n");
                case 9: printf("septembar\n");
                case 10: printf("oktobar\n");
                case 11: printf("novembar\n");
                case 12: printf("decembar");
        }
}
```

IV – Kontrola toka programa

2. Naredbe bezuslovnog skoka

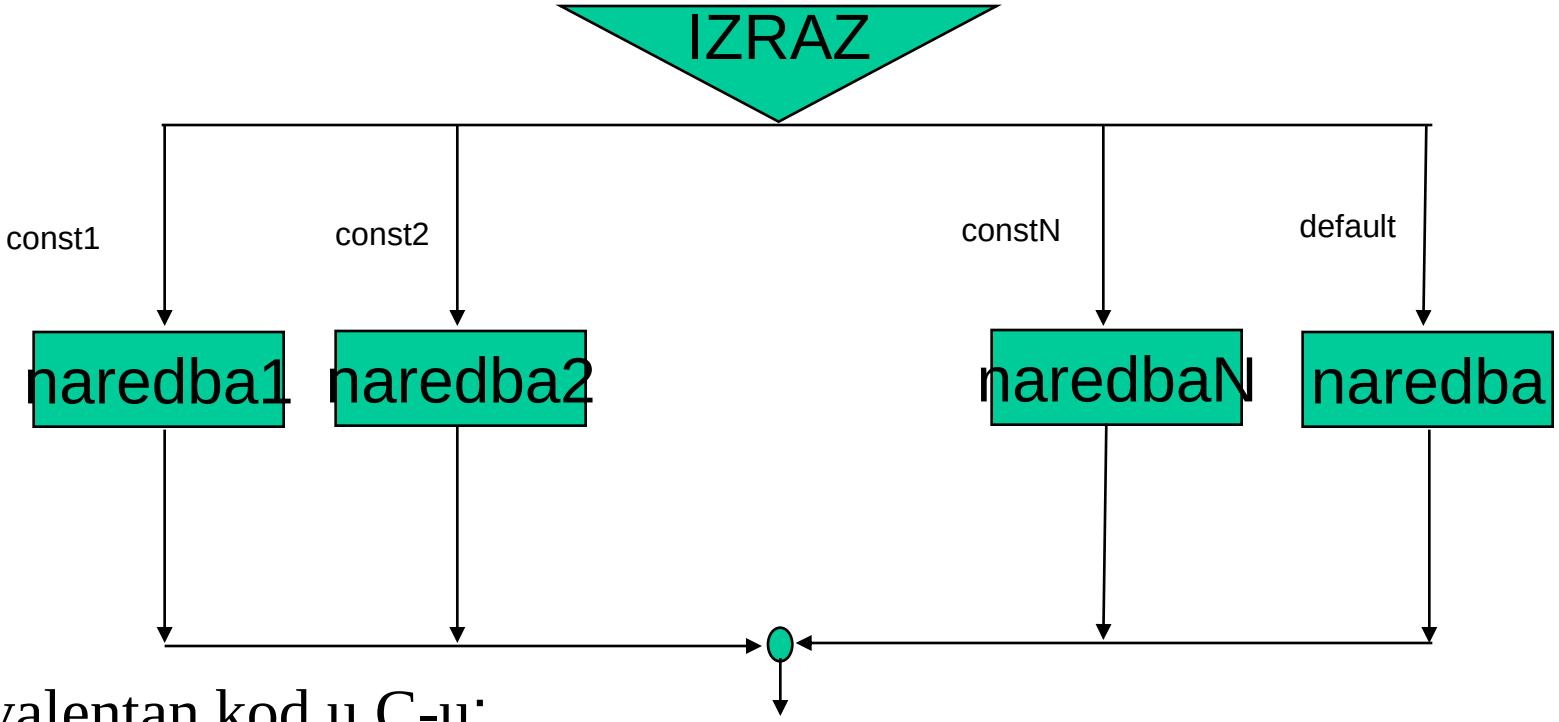
❑ break

- Break prekida izvršenje strukture u kojoj se naredba nalazi.
- Format naredbe je:

  break;

- Ovom naredbom se prekida izvršenje programske petlje
- Izvršenje programa će se nastaviti od prve naredbe iza programske petlje
- Najčešće se koristi unutar *switch*-strukture kako bi se omogućilo izvršenje samo naredbe obeležene odgovarajućom konstantom.
- Struktura koja se realizuje kombinacijom **switch**-strukture i ***break***-naredbe, poznata kao struktura «**češlja**»

IV – Kontrola toka programa



Ekvivalentan kod u C-u:

```
switch (<izraz>)
{
        case <const1>: <naredba1>  break;
        case <const2>: <naredba2>  break;
        ...
        case <constN>: <naredbaN>  break;
        [default: <naredba>]
}
```

IV – Kontrola toka programa

Zadatak: *Napisati program na C-u za štampanje imena tekućeg meseca.*

Rešenje:

```
#include <stdio.h>
main()
{
        int tekuci;
        printf("unesite redni br. tekuceg meseca\n");
        scanf("%d",&tekuci);
        switch( tekuci )
        {
            case 1: printf("januar\n"); break;
            case 2: printf("februar\n"); break;
            case 3: printf("mart\n"); break;
            case 4: printf("april\n"); break;
            case 5: printf("maj\n"); break;
            case 6: printf("jun\n"); break;
            case 7: printf("jul\n"); break;
            case 8: printf("avgust\n"); break;
            case 9: printf("septembar\n"); break;
            case 10: printf("oktobar\n"); break;
            case 11: printf("novembar\n"); break;
            case 12: printf("decembar\n");
        }
}
```

IV – Kontrola toka programa

Zadatak: *Napisati program za štampanje broja dana tekućeg meseca.*

Rešenje:

```
#include <stdio.h>
main()
{
        int tekuci;
        printf("unesite redni br. tekuceg meseca\n");
        scanf("%d",&tekuci);
        switch( tekuci )
        {
          case 1: case 3: case 5: case 7: case 8:
          case 10: case 12: printf("31\n"); break;
          case 2: printf("28\n"); break;
          case 4: case 6: case 9:
          case 11: printf("30\n"); break;
          default: printf("r.br. meseca nije korektan\n" );
        }
}
```

IV – Kontrola toka programa

❑ continue

- **continue** naredba se može naći samo u telu neke programske petlje i njom se prekida izvršenje tekuće iteracije petlje.

Format naredbe:

```
continue;
```

❑ goto

Format naredbe:

```
goto <labela>
```

- **Dejstvo:** Izvršenje programa se nastavlja od naredbe obeležene navedenom labelom.

- Primer labele:

```
lab:
```

IV – Kontrola toka programa

3. Programske petlje

- Programske petlje omogućavaju višestruko ponavljanje određenog dela programa u toku njegovog izvršavanja.
- Vrste petlji:

  a) sa konstantnim brojem prolaza (brojačka petlja),

  b) sa promenljivim brojem prolaza:

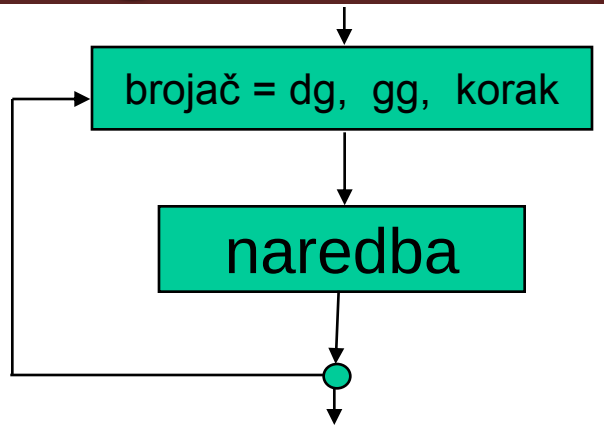
  - do while tip
  - repeat until tip
- U programskom jeziku C postoje tri vrste programskih petlji:

  1) *for* - petlja
  2) *while* - petlja
  3) *do-while* - petlja

❑ FOR petlja

- Koristi se kada u progamu treba realizovati takozvanu brojačku petlju.
- Brojačka petlja podrazumeva da je unapred (pre ulaska u petlju) poznat (može da se izračuna) broj ponavljanja tela petlje.

IV – Kontrola toka programa



<brojač> - ime promenljive koja predstavlja brojač petlje,
<dg> - početna vrednost brojača petlje,
<gg> - konačna vrednost brojača petlje,
<korak> - vrednost koja se dodaje brojaču petlje na kraju svake iteracije.

- Format **for**-naredbe u C-u je:

```
for (<izraz1>; <izraz2>; <izraz3>)
    <naredba>
```

- <izraz1> - vrši inicijalizaciju promenljivih koje se koriste u petlji (što može da bude postavljanje početne vrednosti brojača petlje),
- <izraz2> - predstavlja uslov na osnovu koga se odlučuje da li će se telo petlje još izvršavati ili se izvršenje petlje prekida - izvršava se dok je vrednost ovog izraza različita od nule
- <izraz3> - definiše promenu vrednosti promenljivih u petlji koja se vrši nakon svake iteracije (definiše se kako se menja vrednost brojača)
- Bilo koji od izraza može biti izostavljen, ali se znak ';' mora pisati.

IV – Kontrola toka programa

- **For** petlja u C-u ima širi smisao od navedene definicije brojačkih petlji
- Zamenjena while naredbom, **for** petlja ima oblik:

```
<izraz1>;
while (<izraz2>)
{
   <naredba>
   <izraz3>;
}
```

- Ako je <izraz2> **izostavljen**, smatra se da je uslov for petlje logički tačan, pa se petlja ponaša kao beskonačna petlja.

IV - Kontrola toka programa

Zadatak: *Sumiranje N brojeva čije se vrednosti unose sa tastature.*

Rešenje:

- Ovaj primer pokazuje način realizacije klasične brojačke petlje for-petljom u C-u.
- Brojačkom petljom u ovom slučaju definiše se deo koda koji će se izvršiti tačno N puta.

```
#include <stdio.h>
main()
{
    int i,n,k,S;
    printf("unesite koliko brojeva treba sumirati\n");
    scanf("%d",&n);
    for ( i=0, S=0; i<n; i++ )
    {
            printf("unesite sledeci broj\n");
            scanf("%d",&k);
            S+=k;
    }
    printf("suma unetih brojeva je %d\n",S);
}
```

IV – Kontrola toka programa

Zadatak: *Napisati program na C-u za izračunavanje i štampanje vrednosti funkcije ch(x) korišćenjem for-petlje. Izračunavanje prekinuti kada relativna vrednost priraštaja sume postane manja od zadate vrednosti ε.* ***Napomena:*** *Ovaj red važi za* $|x|<4$.

$$ch(x)=\sum_{k=0}^{\infty}\frac{x^{2k}}{(2k)!}$$

Rešenje:

- U ovom slučaju for petlju koristimo za realizaciju petlji koje se ponavljaju dok je određeni uslov zadovoljen. Praktično, ***for***-petlja sada ima ulogu ***while***-petlje.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        int k;
        float x,a,S,c,eps;
        printf("unesite argument x i tacnost epsilon\n");
        scanf("%f,%e",&x,&eps);
        for ( a=1, S=1, k=0; abs(a/S)>eps;
            a*=x*x/((2*k+2)*(2*k+1)), S+=a, k++ );
        printf("ch(%f)=%e\n",x,S);
}
```

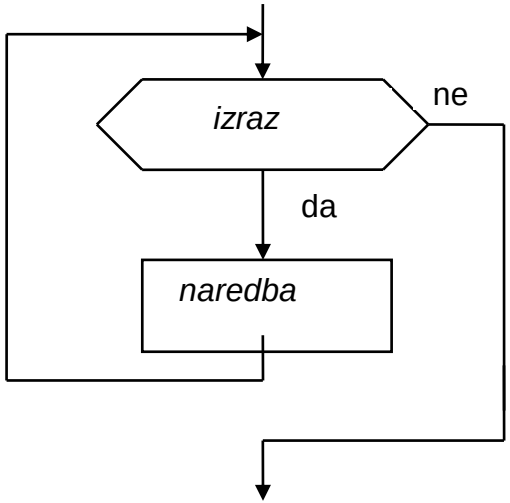
IV – Kontrola toka programa

❑ WHILE petlja

- While petlja omogućava ponavljanje naredbe programa dok je definisan uslov zadovoljen (tj. dok je vrednost izraza različita od nule).
- Ekvivalentan kod u C-u je:

```
while (<izraz>)

    <naredba>
```



- *While*-naredba služi za realizaciju iterativnih procesa (sumiranje beskonačnih redova i sl.)

IV - Kontrola toka programa

Zadatak: *Napisati program na C-u za izračunavanje i štampanje vrednosti funkcije* $ch(x)=\sum_{k=0}^{\infty}\frac{x^{2k}}{(2k)!}$ *primenom sledećeg razvoja u red:*

$$|S_{i+1}-S_i|=|a_{i+1}| \qquad |(S_{i+1}-S_i)/S|\approx|a_{i+1}/S_{i+1}|$$

- Izračunavanje prekinuti kada relativna vrednost priraštaja sume postane manja od zadate vrednosti ε. Napomena:ovaj red važi za $|x|<4$.

Rešenje:

- Vrednost beskonačne sume *S* se zamenjuje konačnom sumom
- Koliko je članova dovoljno sumirati?
- Kriterijum za prekid sumiranja:

- apsolutna vrednost priraštaja sume manja od neke zadate tačnosti
- relativna vrednost priraštaja sume manja od neke zadate tačnosti.

- Rekurentnu formulu za izračunavanje vrednosti proizvoljnog člana sume $a_k=\frac{x^{2k}}{(2k)!}$

$$a_{k+1}=\frac{x^{2(k+1)}}{(2(k+1))!}=\frac{x^{2k}x^2}{(2k+2)!}=\frac{x^{2k}x^2}{(2k+2)(2k+1)2k!}=\frac{x^2}{(2k+2)(2k+1)}a_k$$

- Potrebno je odrediti vrednost prvog člana. $a_0=1$
- Prvi član sume je:

IV - Kontrola toka programa

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        int k;
        float x,a,S,c,eps;
        printf("unesite argument x i tacnost eps\n");
        scanf("%f,%e",&x,&eps);
        a=1;
        S=1;
        k=0;
        while ( abs(a/S)>eps )
        {
                a*=x*x/((2*k+2)*(2*k+1));
                S+=a;
                k++;
        }
        printf("ch(%f)=%e\n",x,S);
}
```

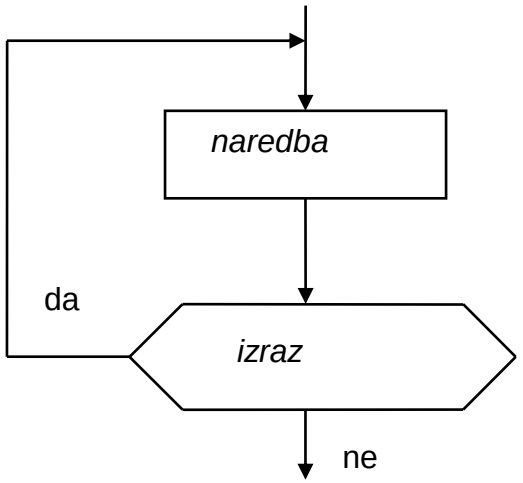
IV – Kontrola toka programa

❑ DO WHILE petlja

- Do-while petlja ima slično dejstvo kao i while-petlja.
- Jedina razlika je u tome što se uslov za ponavljanje petlje u kodu nalazi iza njenog tela tako da se naredba, koja predstavlja telo petlje, mora izvršiti bar jednom.

- Format odgovarajuće naredbe u programskom jeziku C je:

```
do
    <naredba>
while (<izraz>);
```



IV - Kontrola toka programa

Zadatak: *Napisati program koji izračunava i štampa vrednost n-tog korena iz a (pri čemu je a>0), primenom sledećeg iterativnog postupka:*

$$x_0 = (a + n - 1)/n \qquad x_{i+1} = ((n-1)x_i + a/x_i^{n-1})/n$$

Izračunavanje prekinuti kada je $|x_{i+1} - x_i| \le \varepsilon$ *gde je* ε *zadata tačnost.*

Rešenje:

```
#include <stdio.h>
#include <math.h>
main()
{
        int n;
        float x1,x2,a,eps,absvr;
        printf("unesite n, a i tacnost epsilon\n");
        scanf("%d%f%e",&n,&x,&eps);
        x2=(a+n-1)/n;
        do {
                x1=x2;
                x2=((n-1)*x1+a/pow(x1,n-1))/n;
                absvr = x2-x1>0 ? x2-x1 : x1-x2;
        } while( absvr > eps );
        printf("x=%g",x);
}
```

Hvala na pažnji !!!

Pitanja

? ? ?