PROGRAMIRANJE II

P-01: Rekurzije

prof. dr **Dražen Brđanin**
2023/24

P-01: Rekurzije

- **Sadržaj predavanja**
  - definicija rekurzije
  - osnovne karakteristike rekurzije
  - proces izvršavanja rekurzije
  - dobre i loše strane rekurzije
  - eliminacija rekurzije
  - primjeri rekurzija

Definicija rekurzije

- U matematici i računarstvu, **rekurzija je pristup u kojem se neki pojam, objekat ili funkcija definiše na osnovu jednog ili više osnovnih (baznih) slučajeva i na osnovu pravila koja složene slučajeve svode na jednostavnije.**

- **Rekurzivna funkcija** = **funkcija koja poziva samu sebe**, svodeći rješavanje složenog problema na jednostavniji problem iste prirode, sve dok se problem ne pojednostavi do osnovnog (trivijalnog) slučaja.

- Nemaju svi programski jezici podršku za rekurzivne potprograme (npr. FORTRAN)

Primjer: **Rekurzivna (induktivna) definicija** x^n

$$x^n = \begin{cases} 1, & n = 0 \\ x \cdot x^{n-1}, & n > 0 \end{cases}$$

osnovni (bazni) slučaj

rekurzivni korak

Definicija rekurzije

- **Osnovni elementi rekurzije**:
  - **osnovni (bazni) slučaj** = jednostavan (trivijalan) problem koji može da se riješi bez rekurzivnog poziva i koji omogućava zaustavljanje rekurzije.
  - **rekurzivni korak** = mehanizam za pojednostavljenje složenog problema, tj. svođenje složenog problema na rješavanje jednostavnijeg problema iste prirode

    Rješenje problema u *n*-tom koraku bazira se na rješenju iz (*n*-1)-og koraka.

$$x^n = \begin{cases} 1, & n = 0 \\ x \cdot x^{n-1}, & n > 0 \end{cases}$$

osnovni (bazni) slučaj

rekurzivni korak

- **Izostavljanje osnovnog slučaja ili rekurzivnog koraka čini definiciju nekompletnom**.

Definicija rekurzije

- **Implementacija rekurzije**:

$$x^n = \begin{cases} 1, & n = 0 \\ x \cdot x^{n-1}, & n > 0 \end{cases}$$

osnovni (bazni) slučaj

rekurzivni korak

```
float stepenovanje(float x, int n)
{
    if (n==0)
        return 1;
    else
        return x * stepenovanje(x,n-1);
}
```

osnovni (bazni) slučaj

rekurzivni korak

Definicija rekurzije

- **Analiza izvršavanja rekurzivne funkcije**:

```
float stepenovanje(float x, int n)
{
    if (n==0) return 1;
    else return x * stepenovanje(x,n-1);
}
```

```
stepenovanje(3,2)
_________________
 x=3
 n=2 ⇒ return  3 * stepenovanje(3,1)                         3 * 3 = 9
                   _________________
                    x=3
                    n=1 ⇒ return  3 * stepenovanje(3,0)     3 * 1 = 3
                                      _________________
                                       x=3
                                       n=0 ⇒ return 1
```

Definicija rekurzije

Primjer:

```
#include <stdio.h>

float stepenovanje(float x, int n)
{
   if (n==0)
      return 1;
   else
      return x*stepenovanje(x,n-1);
}

int main()
{
   float x;
   printf("x=");
   scanf(“%f", &x);
   for (int n=0; n<5; n++)
      printf("%.2f^%d=%.4f\n", x,n,stepenovanje(x,n));
   return 0;
}
```

Primjer izvršavanja:

```
x=3
3.00^0=1.0000
3.00^1=3.0000
3.00^2=9.0000
3.00^3=27.0000
3.00^4=81.0000
```

Osnovne karakteristike rekurzije

- **Postojanje osnovnog slučaja**
  - mora da postoji jedan ili više osnovnih slučajeva čije je rješenje jednostavno (trivijalno) i ne zahtijeva rekurzivni poziv
  - postojanje osnovnog slučaja omogućava zaustavljanje rekurzije
- **Progres / konvergencija**
  - svaki (uzastopni) rekurzivni korak mora da vodi prema osnovnim slučajevima
  - rješavanje složenog problema mora da se svodi na rješavanje jednostavnijeg problema iste prirode, tako što funkcija poziva samu sebe ali sa drugim argumentima (koji reprezentuju jednostavniji problem)
- **Onemogućavanje/izbjegavanje ponavljanja koraka**
  - ne treba omogućiti da se ponavlja rješavanje istog problema u više uzastopnih koraka, jer to značajno troši resurse i usporava rad (vidjeti primjer sa Fibonačijevim nizom)

Proces izvršavanja rekurzije

- **Rekurzivna funkcija je funkcija!**
  - **Programski kod** rekurzivne funkcije (isto kao i za svaku drugu funkciju) tokom izvršavanja programa nalazi se **u CODE SEGMENTU**.
  - U CODE SEGMENTU postoji **samo jedan primjerak koda rekurzivne funkcije**.
  - **Prilikom poziva rekurzivne funkcije** (isto kao i za svaku drugu funkciju) na steku se formira odgovarajući **stek okvir** u kojem se nalaze:
    - argumenti koji se prosljeđuju u funkciju (stvarni↔formalni),
    - adresa povratka u pozivajućoj funkciji (kako bi se znalo odakle se nastavlja izvršavanje nakon povratka iz funkcije), ...
  - **Prilikom izvršavanja rekurzivnog koraka** (funkcija poziva samu sebe) na steku se formira **novi stek okvir** (koji pripada novoj instanci pozvane funkcije), u kojem se nalaze:
    - argumenti koji se u rekurzivnom koraku prosljeđuju u pozvanu funkciju,
    - adresa povratka u pozivaocu (kako bi se znalo odakle se nastavlja izvršavanje nakon povratka iz pozvane funkcije), ...

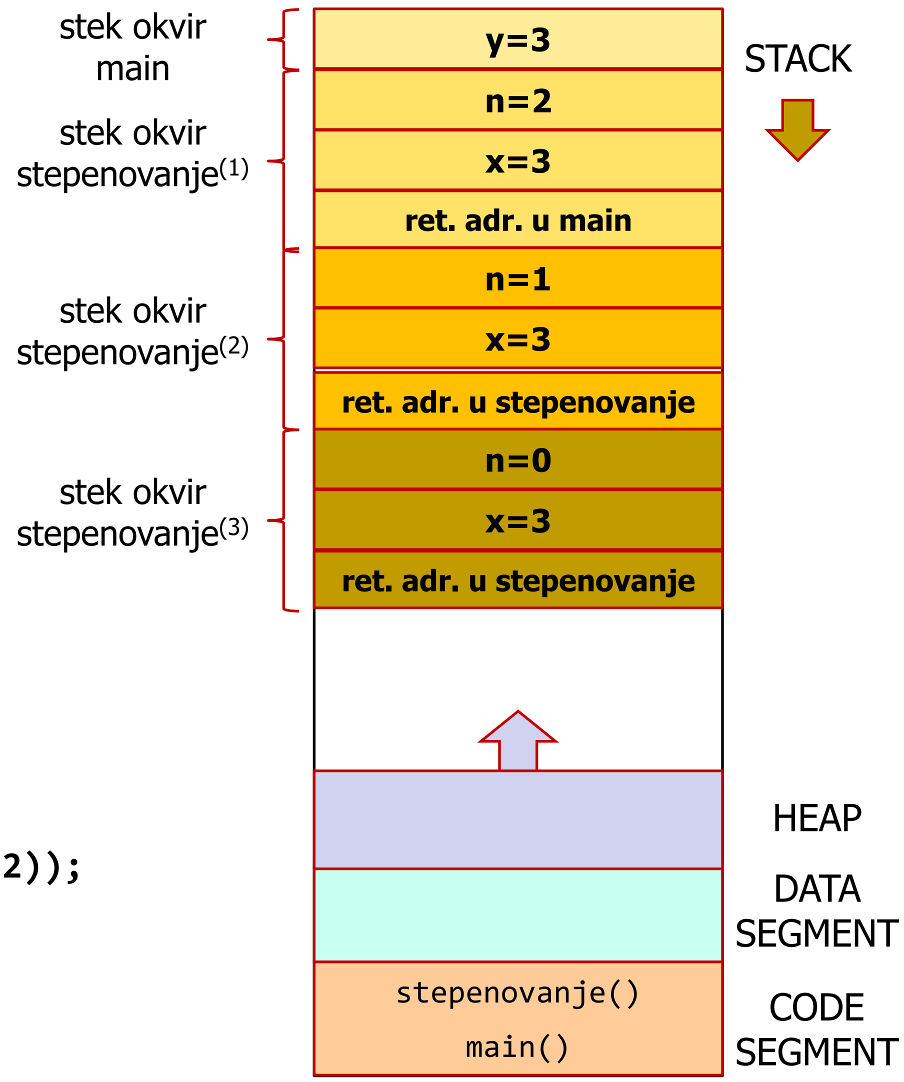
Proces izvršavanja rekurzije

Primjer:

```
#include <stdio.h>

float stepenovanje(float x, int n)
{
   if (n==0)
      return 1;
   else
      return x*stepenovanje(x,n-1);
}

int main()
{
   float y;
   printf("y=");
   scanf("%f", &y);
   printf("%.2f^2=%.4f\n", y,stepenovanje(y,2));
   return 0;
}
```



Proces izvršavanja rekurzije

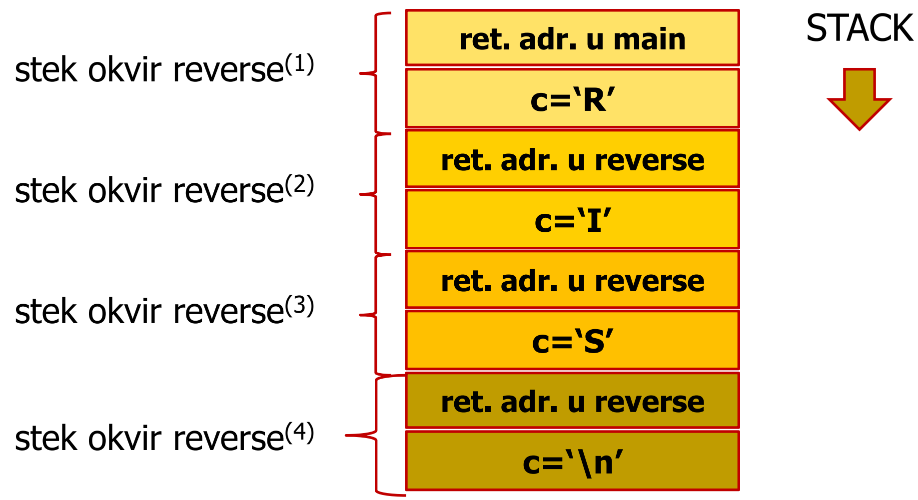
Po povratku iz pozvane funkcije (završen rekurzivni korak), nastavlja se izvršavanje od mjesta na kojem je prekinuto izvršavanje.

Primjer:

```
#include <stdio.h>

void reverse()
{
    char c;
    scanf("%c", &c);
    if (c != '\n')
    {
        reverse();
        printf("%c", c);
    }
    return;
}

int main()
{
    reverse();
    return 0;
}
```



Primjer izvršavanja:

```
RIS
SIR
```

Proces izvršavanja rekurzije

Primjer
(upotreba statičkih promjenljivih):

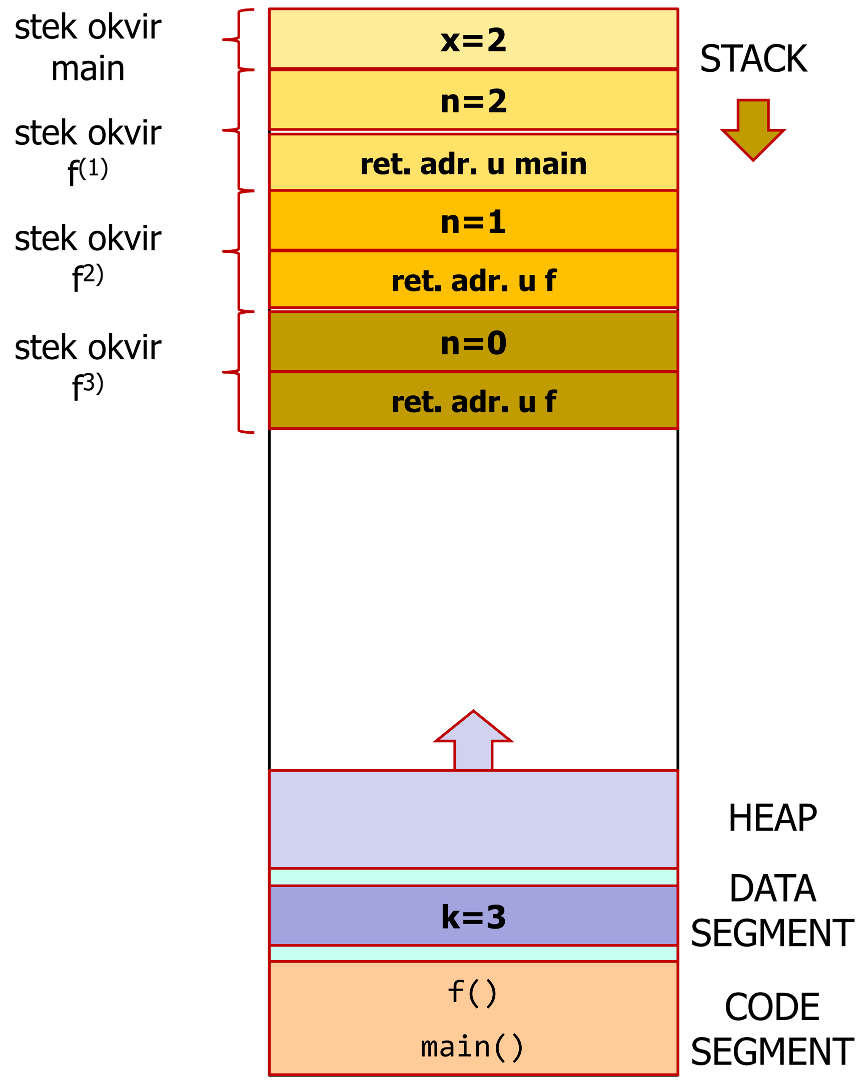
```
#include <stdio.h>

void f(int n)
{
   static int k=0;
   k++;
   if (n) f(n-1);
   printf("n=%d k=%d\n", n, k);
}

int main()
{
   int x=2;
   f(x);
   return 0;
}
```

Primjer izvršavanja:

```
n=0 k=3
n=1 k=3
n=2 k=3
```



Zašto rekurzija mora da konvergira?

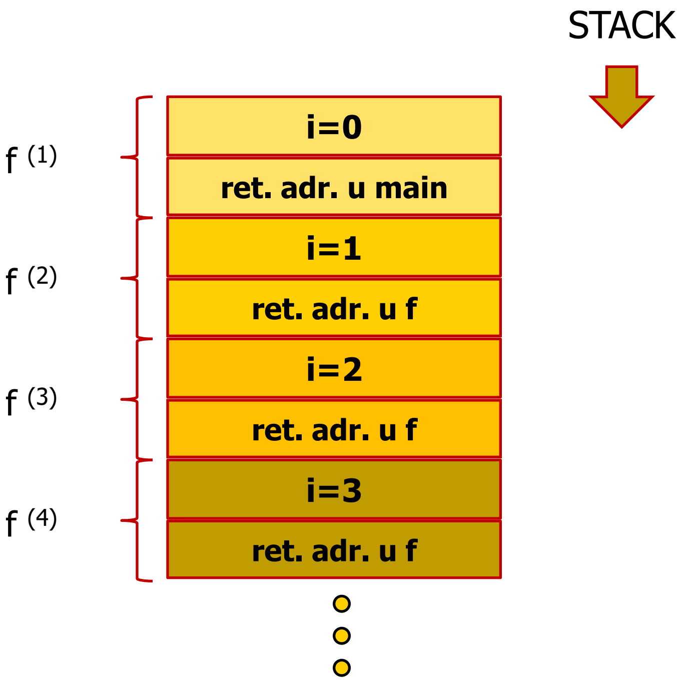
Svaka nova instanca pozvane funkcije ima svoj stek okvir, a veličina steka je ograničena!

Primjer 1:

```
#include <stdio.h>

void f(int i)
{
   f(i+1);
   return;
}

int main()
{
   f(0);
   return 0;
}
```



Primjer 2:

```
int bad(int n)
{
   if (n == 0) return 0;
   return bad(n/3 + 1);
}
```

bad(1)

n=1 ⇒ return bad(1)

n=1 ⇒ return bad(1) ???

Dobre i loše strane rekurzije

Dobre strane rekurzije

Kod je (obično):

- kratak, čitljiv i jednostavan za razumijevanje,
- jednostavan za održavanje i otklanjanje grešaka,
- pogodan za dokazivanje korektnosti,
- ...

Rekurziju treba koristiti:

- ako je rekurzivno rješenje "prirodno" i jednostavno za razumijevanje,
- ako rekurzivno rješenje ne zahtijeva suvišna izračunavanja koja je teško eliminisati,
- ako je ekvivalentno iterativno (nerekurzivno) rješenje previše kompleksno.

Loše strane rekurzije

- **Cijena poziva**:
  - svaki rekurzivni korak znači novi stek okvir i kopiranje argumenata na stek, što dalje znači novo **memorijsko zauzeće** i **usporavanje izvršavanja**
  - u slučaju "dubokih" rekurzija, **prostorna** (memorijska) i **vremenska složenost** mogu biti **kritične**
- **Suvišna izračunavanja**:
  - svođenje složenog problema na jednostavnije može da rezultuje suvišnim ponavljanjima istih izračunavanja

Dobre i loše strane rekurzije

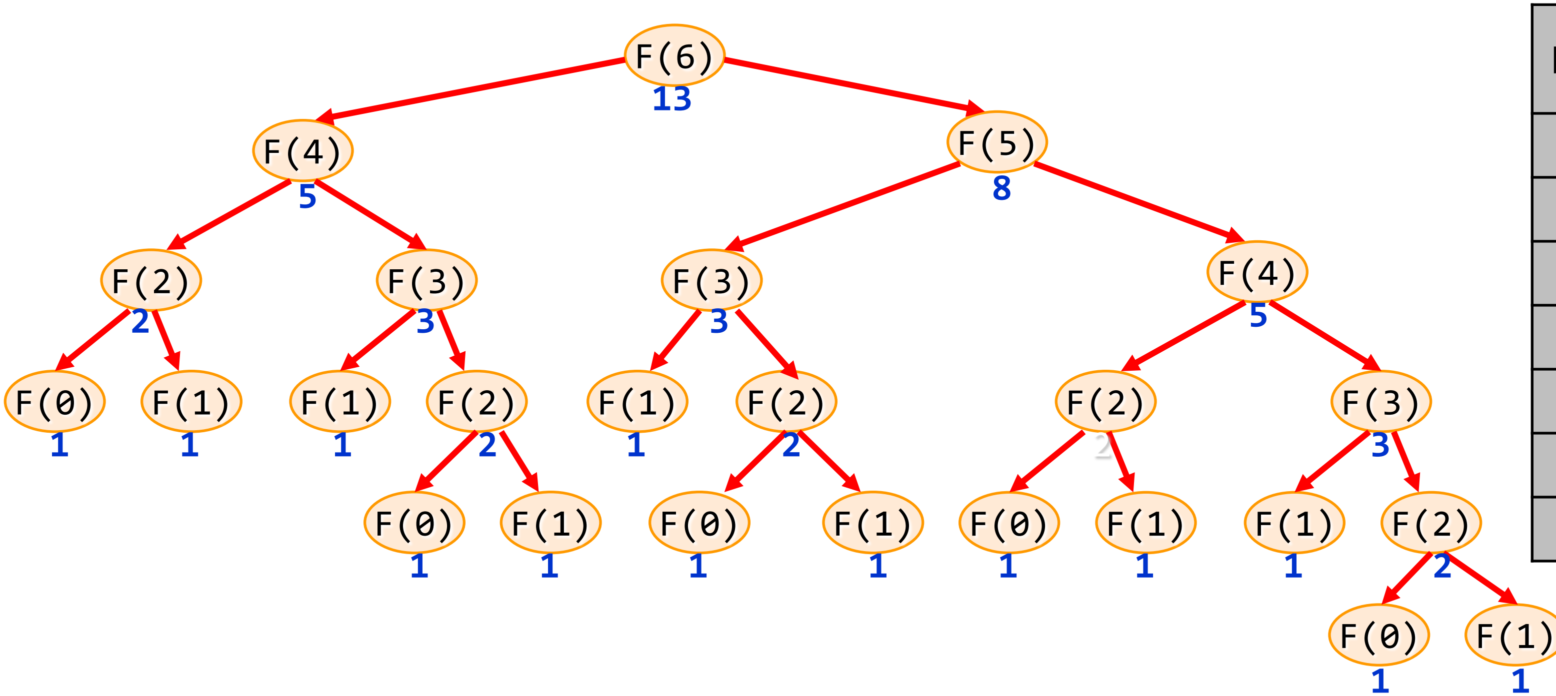
Primjer suvišnih izračunavanja (Fibonačijev niz):

1, 1, 2, 3, 5, 8, 13, 21, 34,... (?)

$F_0=F_1=1$

$F_i=F_{i-1}+F_{i-2};\ i>1$

```
int f(int i)
{
   if (i<=1) return 1;
   else return f(i-1)+f(i-2);
}
```



Poziv	Broj izvršavanja
F(6)	1
F(5)	1
F(4)	2
F(3)	3
F(2)	5
F(1)	8
F(0)	5

Dobre i loše strane rekurzije

Eliminacija suvišnih izračunavanja (memoizacija):

Memoizacija je tehnika koja podrazumijeva **pamćenje svih rezultata** ranijih rekurzivnih poziva u odgovarajućoj strukturi podataka.

Prilikom ulaska u funkciju provjerava se da li je već izračunata tražena vrijednost.

Ako postoji izračunata vrijednost, vraća se rezultat.

Inače se izračunava nova vrijednost, dodaje u strukturu i vraća rezultat.

```
int f(int i)
{
   if (i<=1) return 1;
   else return f(i-1)+f(i-2);
}
```

```
int f(int i)
{
   static int memo[MAX]={1,1};
   if (memo[i]) return memo[i];
   else return memo[i]=f(i-1)+f(i-2);
}
```

i	0	1	2	3	4	5	6
broj izvršavanja	1	1	3	5	9	15	25

i	0	1	2	3	4	5	6
min. broj izvršavanja	1	1	3	3	3	3	3
max. broj izvršavanja	1	1	3	5	7	9	11

Dobre i loše strane rekurzije

Eliminacija suvišnih izračunavanja (redefinicija rekurzivnog koraka):

$$x^n = \begin{cases} 1, & n = 0 \\ x \cdot x^{n-1}, & n > 0 \end{cases}$$

$$\Rightarrow$$

$$x^n = \begin{cases} 1, & n = 0 \\ (x \cdot x)^{n/2}, & n \text{ parno} \\ x \cdot x^{n-1}, & n \text{ neparno} \end{cases}$$

```
double stepenovanje(double x, int n)
{
   if (n==0)
      return 1;
   else
      return x*stepenovanje(x,n-1);
}
```

n	0	1	2	3	4	5	6	7	8
broj izvršavanja	1	2	3	4	5	6	7	8	9

```
double stepenovanje(double x, int n)
{
   if (n==0)
      return 1;
   else
      if (n%2==0)
         return stepenovanje(x*x,n/2);
      else
         return x*stepenovanje(x,n-1);
}
```

n	0	1	2	3	4	5	6	7	8
broj izvršavanja	1	2	3	4	4	5	5	6	5

Eliminacija rekurzije

- Svaku rekurzivnu funkciju moguće je transformisati u ekvivalentnu iterativnu (nerekurzivnu) funkciju.
- Ne postoji jedinstven i univerzalan pristup za transformaciju rekurzivne u nerekurzivnu funkciju.
- Univerzalan pristup za transformaciju rekurzivne u nerekurzivnu funkciju zahtijevao bi postojanje odgovarajuće strukture podataka koja bi sadržavala sve podatke/rezultate koji se smještaju na stek.
- Neke klase rekurzivnih funkcija mogu veoma jednostavno da se transformišu u nerekurzivne funkcije (npr. **repne rekurzije**).
- **Repni rekurzivni poziv** = **rekurzivni poziv čiji je rezultat ujedno i rezultat funkcije**, tj. **nakon rekurzivnog poziva (i vraćanja rezultata) nema dodatnih naredbi/izračunavanja**.

Eliminacija rekurzije

Repni rekurzivni poziv = **rekurzivni poziv čiji je rezultat ujedno i rezultat funkcije**, tj. **nakon rekurzivnog poziva (i vraćanja rezultata) nema dodatnih naredbi/izračunavanja**.

Primjer:

```
float stepenovanje(float x, int n)
{
   if (n==0)
      return 1;
   else
      if (n%2==0)
         return stepenovanje(x*x,n/2);
      else
         return x*stepenovanje(x,n-1);
}
```

repni rekurzivni poziv

nije repni rekurzivni poziv jer ima dodatno računanje nakon povratka iz pozvane funkcije

Eliminacija repne rekurzije

Repna rekurzija može da se eliminiše na sljedeći način:

- **prije rekurzivnog poziva treba promijeniti argument tako da ima vrijednost koju bi imao kad se izvrši rekurzivni poziv,**
- nakon što se promijeni vrijednost argumenta, nema više potrebe da se vrši rekurzivni poziv nego je dovoljno **kontrolu prebaciti na početak funkcije** (npr. pomoću `goto`) ,
- refaktorisati kod tako da se **goto zamijeni odgovarajućom petljom**.

Primjer:

```
int f(int n)
{
    if (n==0)
       return 1;
    else
       f(n-1);
}
```

⇒

```
int f(int n)
{
   start:
   if (n==0)
      return 1;
   else
   {
      n=n-1;
      goto start;
   }
}
```

⇒

```
int f(int n)
{
   while (n>0)
      n=n-1;
   return 1;
}
```

Eliminacija repne rekurzije

Primjer (Euklidov algoritam za određivanje mjere dva broja):

```
int mjera(int a, int b)
{
  if (b==0)
    return a;
  else
    return mjera(b,a%b);
}
```

```
int mjera(int a, int b)
{
  start:
  if (b==0)
    return a;
  else
  {
    int tmp=a%b;
    a=b;
    b=tmp;
    goto start;
  }
}
```

```
int mjera(int a, int b)
{
  while (b>0)
  {
    int tmp=a%b;
    a=b;
    b=tmp;
  }
  return a;
}
```

Primjeri rekurzija

Primjer (rekurentna relacija):

$$x_n = 5x_{n-1} - 4x_{n-2}, \quad x_1 = 1, \; x_2 = 2$$

```
unsigned clan(unsigned n)
{
  if (n<3) return n;
  return 5*clan(n-1)-4*clan(n-2);
}
```

Primjer (faktorijel):

$$n! = \begin{cases} 1, & n \le 1 \\ n \cdot (n-1)!, & n > 1 \end{cases}$$

```
unsigned faktor(unsigned n)
{
  if (n<=1) return 1;
  else return n*faktor(n-1);
}
```

```
unsigned faktor(unsigned n)
{
  return (n<=1) ? 1 : n*faktor(n-1);
}
```

Primjeri rekurzija

Primjer (sekvencijalno pretraživanje niza):

```
int search(tip niz[], tip x, int kapacitet, int i)
{
   if (i >= kapacitet) return -1;
   if (niz[i] == x) return i;
   return search(niz, x, kapacitet, i+1);
}
```

Inicijalni poziv funkcije za pretraživanje

```
search(niz,x,n,0)
```

Primjer (poboljšano sekvencijalno pretraživanje niza sa "stražom"):

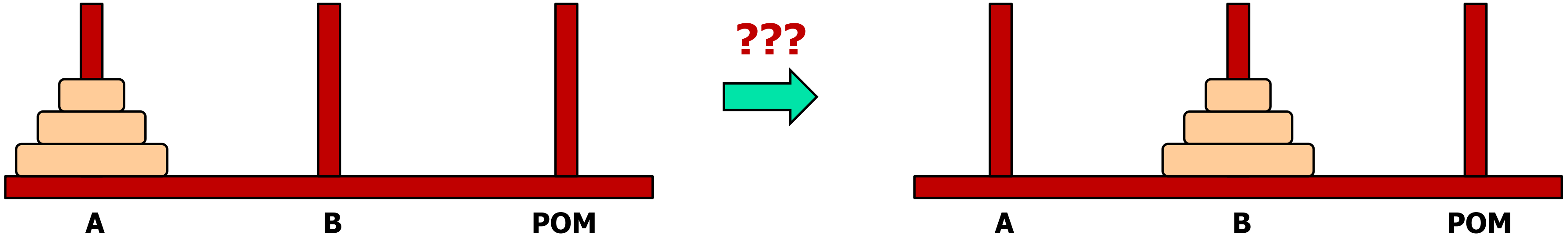
```
int search(tip niz[], tip x, int i)
{
   if (niz[i] == x) return i;
   return search(niz, x, i+1);
}
```

Potreban kod u pozivaocu (na kraj niza dodaje se "**stražar**" **– tražena vrijednost**)

```
niz[n]=x;
search(niz,x,0)
```

Primjeri rekurzija

Primjer (Hanojske kule – *Towers of Hanoi*):



Zadatak:

Prebaciti svih *n* (zlatnih) prstenova sa kule A na kulu B.

Pravila igre:

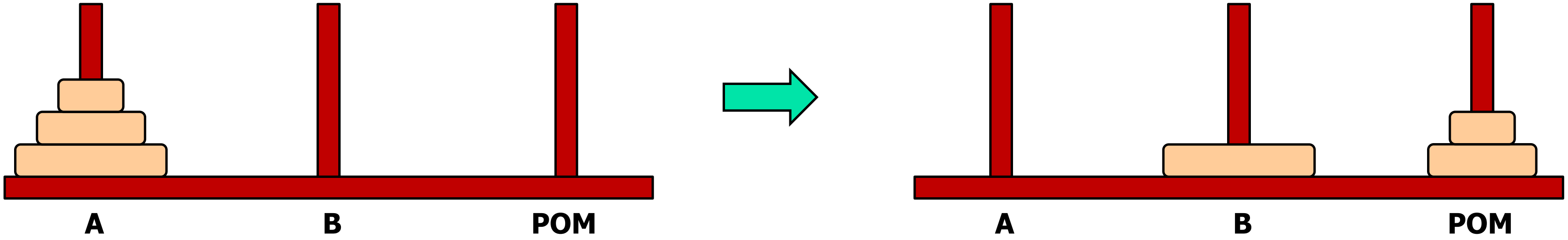
1. U jednom potezu može da se prebaci samo jedan prsten.
2. Manji prsten može da se stavi samo na veći prsten.
3. Za premještanje je dozvoljeno koristiti pomoćnu kulu POM.

Primjeri rekurzija

Primjer (Hanojske kule – *Towers of Hanoi*):

Ideja za rješavanje problema:

- **REKURZIJA**: problem prebacivanja *n* prstenova treba svesti na prebacivanje *n*-1 prstena.
- Ako prebacimo *n*-1 prstenova sa A na POM, tada ćemo moći preostali prsten prebaciti sa A na B.



- Sada je najveći prsten na odgovarajućoj kuli (B) i problem je sveden sa *n* na *n*-1 prsten.
- Prebacivanje *n*-1 prstenova sa POM na B je isti problem kao i prebacivanje *n* prstenova sa A na B, samo jednostavniji (jer ima jedan prsten manje).

- **OSNOVNI SLUČAJ**: za *n*=1, prsten se prebaci sa A na B
- **Problem se svodi na PREBACIVANJE SA jedne kule NA drugu kulu PREKO trece kule**

`PREBACI(n, SA, NA, PREKO)`

Primjeri rekurzija

Primjer (Hanojske kule – *Towers of Hanoi*):

Algoritam: PREBACI(n,SA,NA,PREKO)

```
1. Ako je n=1 ispisi SA->NA  (osnovni slučaj)

2. Inače

    2.1. PREBACI(n-1,SA,PREKO,NA)   (prebaci n-1, oslobodi najveći)

    2.2. ispisi SA->NA

    2.3. PREBACI(n-1,PREKO,NA,SA)   (prebaci preostalih n-1)
```

Poziv algoritma:

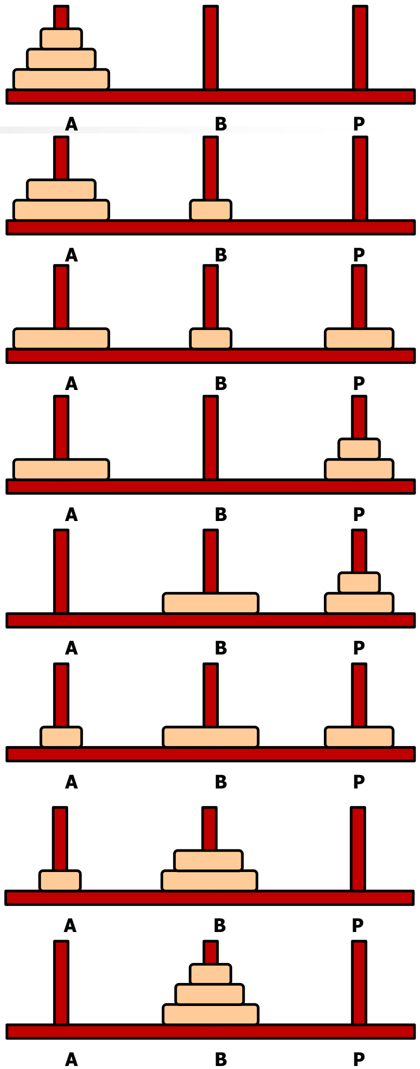
```
PREBACI(n, A, B, POM)
```

Primjeri rekurzija

Primjer (Hanojske kule – *Towers of Hanoi*):

Implementacija:

```
void prebaci(int n, char sa, char na, char preko)
{
    if (n==1)
       printf("%c->%c ", sa, na);
    else
    {
       prebaci(n-1,sa,preko,na);
       printf("%c->%c ", sa, na);
       prebaci(n-1,preko,na,sa);
    }
}
int main()
{
   prebaci(3,'A','B','P');
   return 0;
}
```



Rezultat izvršavanja:

```
A->B A->P B->P A->B P->A P->B A->B
```

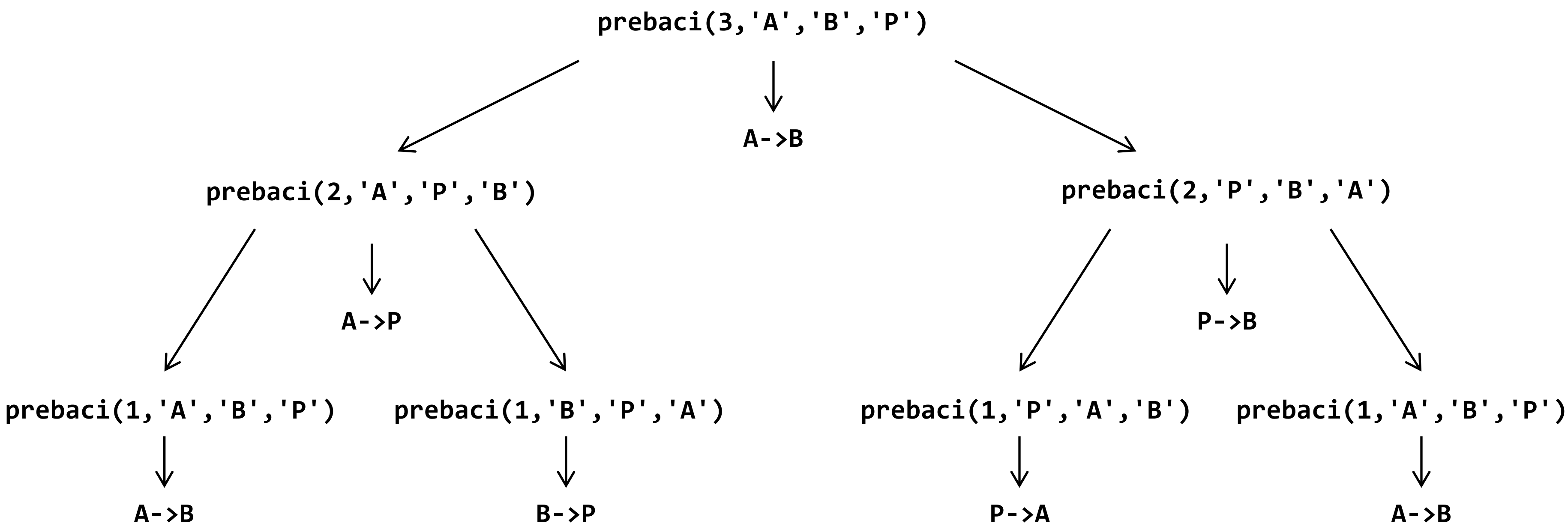
Primjeri rekurzija

Primjer (Hanojske kule – *Towers of Hanoi*):

Rezultat izvršavanja:

```
A->B A->P B->P A->B P->A P->B A->B
```

Analiza izvršavanja (za *n*=3):



n	1	2	3	4	5	6	7	8	9
broj izvršavanja	1	3	7	15	31	63	127	255	511