Funkcije (nastavak)

Prototipovi

Neophodni zbog linearne strukture C programa

Omogućavaju kompajliranje u jednom prolazu

Omogućavaju pozivanje funkcija iz main funkcije pre njihove implementacije

Deklarišu se tako što se navodi samo zaglavlje bez tela funkcije:

```
int saberi(int a, int b);
```

Moguće je u potpunosti izostaviti nazive promenljive, jer oni nisu bitni za slaganje formalnih i stvarnih parametara:

```
int saberi(int , int);
```

Umetnute (engl. *inline*) funkcije

Česta upotreba malih funkcija usporava izvršavanje – mehanizam poziva funkcija korišćenjem steka (engl. *stack*) i stek frejma (engl. *stack frame*)

Rezervisana reč *inline* je sugestija prevodiocu da pokuša da na mesto poziva funkcije direktno "umetne" mašinski kod funkcije – cilj ubzanje rada

Uvedene sa C99 standardom

Osnovna struktura C programa

Program se sastoji od jedne ili više funkcija

Funkcije su gradivne jedinice programa i mogu pozivati jedna drugu

Izvršavanje programa počinje od funkcije main()

Osobine strukture C programa:

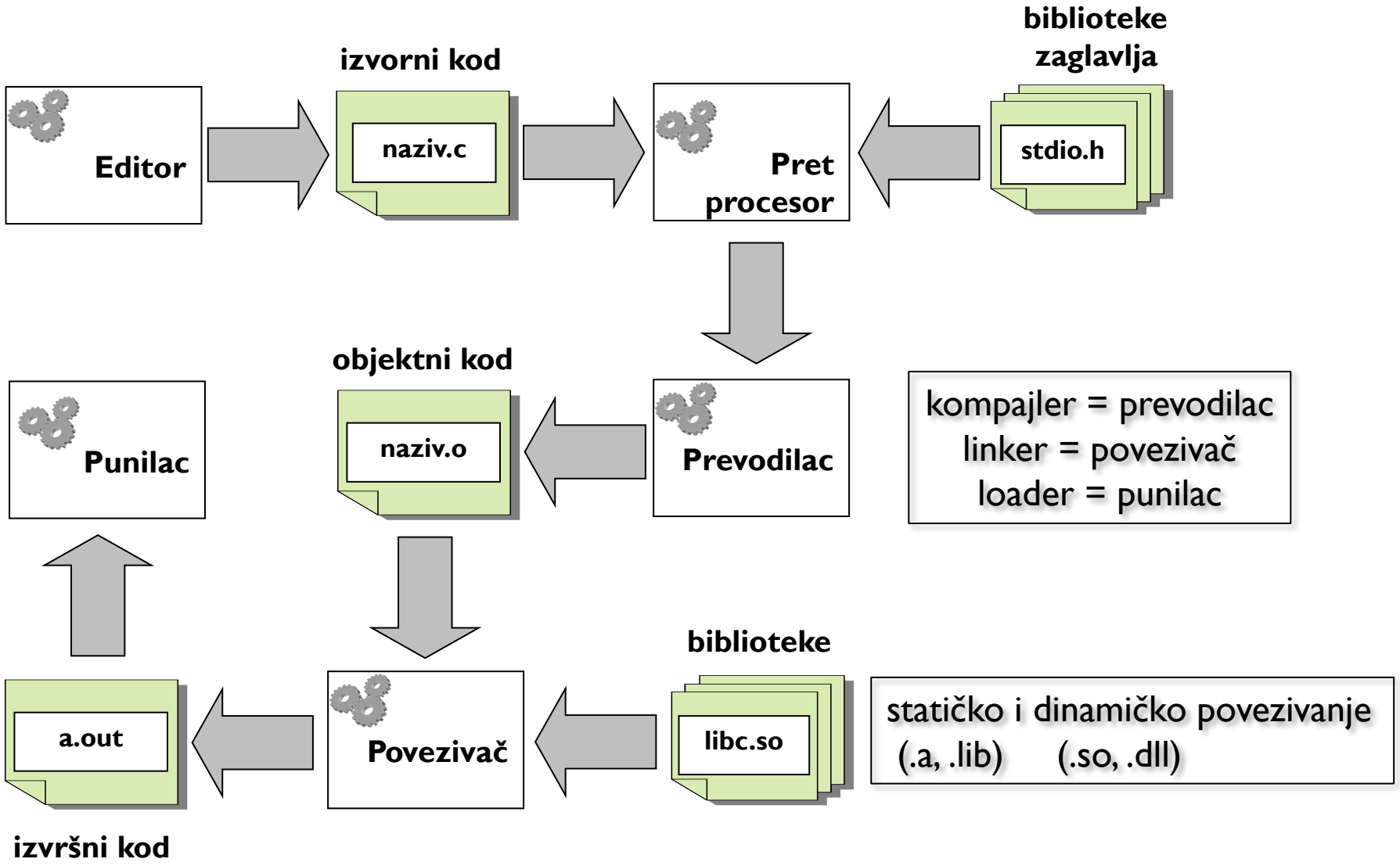
Linearna u jednom nivou

Top-down razvoj

Prevođenje u jednom prolazu

Prvo definiši, pa tek onda koristi!

Prevođenje i pokretanje C programa



Funkcije – primeri

Zadatak 1:
Implementirati program koji računa sumu, razliku, proizvod i količnik dva realna broja u jednostrukoj preciznosti. Brojeve zadaje korisnik preko tastature. Implementirati unos, sabiranje, oduzimanje, množenje, deljenje i prikaz rezultata kao zasebne funkcije.

Vežba 1:
Izmeniti prethodni program tako da korisnik može da bira željenu operaciju (sabiranje, oduzimanje, množenje ili deljenje), pri čemu za svaku operaciju ponovo definiše operande. Omogućiti ponavljanje ovih operacija sve dok korisnik ne odluči da izađe iz programa.

Funkcije – Zadatak I

```
#include <stdio.h>

void unesiOperande(float *op1, float *op2);
inline float saberi(float op1, float op2);
inline float oduzmi(float op1, float op2);
inline float pomnozi(float op1, float op2);
inline float podeli(float op1, float op2);
void prikaziRezultate(float op1, float op2, float zbir, float razlika, float proizvod, float kolicnik);

int main()
{
   float operand1 = 0, operand2 = 0;
   float zbir = 0, razlika = 0, proizvod = 0, kolicnik = 0;

   printf("\n--Program za simuliranje kalkulatora--\n\n");

   unesiOperande(&operand1, &operand2);

   zbir = saberi(operand1, operand2);
   razlika = oduzmi(operand1, operand2);
   proizvod = pomnozi(operand1,operand2);
   kolicnik = podeli(operand1, operand2);

   prikaziRezultate(operand1, operand2, zbir, razlika, proizvod, kolicnik);

   return 0;
}
```

Funkcije – Zadatak I

```
void unesiOperande(float *op1, float *op2)
{
    printf("Unesite prvi operand:\t");
    scanf("%f", op1);

    printf("Unesite drugi operand:\t");
    scanf("%f", op2);

    printf("\n-------------------------------------");
}

void prikaziRezultate(float op1, float op2, float zbir, float razlika, float proizvod, float kolicnik)
{
    printf("\n\nRezultati osnovnih operacija su:\n");

    printf("\n%4.2f + %4.2f = %4.2f\n", op1, op2, zbir);

    printf("\n%4.2f - %4.2f = %4.2f\n", op1, op2, razlika);

    printf("\n%4.2f * %4.2f = %4.2f\n", op1, op2, proizvod);

    printf("\n%4.2f / %4.2f = %4.2f\n", op1, op2, kolicnik);

    printf("\n\n-------------------------------------\n");
}
```

Funkcije – Zadatak I

```
inline float saberi(float op1, float op2)
{
   return op1 + op2;
}

inline float oduzmi(float op1, float op2)
{
   return op1 - op2;
}

inline float pomnozi(float op1, float op2)
{
   return op1 * op2;
}

inline float podeli(float op1, float op2)
{
   return op1 / op2;
}
```

- Koji problem može da se javi prilikom deljenja?

Funkcije – primeri

Zadatak 2:
Implementirati program za računanje sume vrednosti elemenata niza celih brojeva koji sadrži maksimalno 50 elemenata. Program prihvata od korisnika broj elemenata ($0 < n \leq 50$) i vrednost svakog pojedinačnog elementa. Unos elemenata i računanje sume realizovati kao zasebne funkcije.

Vežba 2:
Implementirati program koji od korisnika prihvata *n* elemenata niza (koji maksimalno može sadržati do 50 elemenata). Omogućiti korisniku da bira neku od sledećih operacija: izračunavanje sume elemenata niza, računanje srednje vrednosti niza, nalaženje minimuma, nalaženje maksimuma. Omogućiti ponavljanje ovih operacija sve dok korisnik ne odluči da izađe iz programa.

Funkcije – Zadatak 2

```
#include <stdio.h>

#define MAX 50

void unesiElementeNiza(int *pN, int nizPrirodnihBrojeva[MAX]);
int sumirajNiz(int pN, int nizPrirodnihBrojeva[MAX]);

int main()
{
    int n = 1;
    int nizCelihBrojeva[MAX];
    printf("\n Program za racunanje sume elemenata niza N celih brojeva.\n\n");
    unesiElementeNiza(&n, nizCelihBrojeva);
    printf("\n Suma elemenata niza iznosi %d.\n\n", sumirajNiz(n, nizCelihBrojeva));
    return 0;
}
```

Funkcije – Zadatak 2

```
void unesiElementeNiza(int *pn, int nizCelihBrojeva[MAX])
{
    int i;

    printf("Unesite N:\t");
    scanf("%d", pn);

    for (i = 0; i < *pn; i++)
    {
        printf("Unesite %d. element niza:\t", i+1);
        scanf("%d", &nizCelihBrojeva[ i ]);
    }
}

int sumirajNiz(int pn, int nizCelihBrojeva[MAX])
{
    int i;
    int suma = 0;

    for (i = 0; i < pn; i++)
        suma += nizCelihBrojeva[ i ];

    return suma;
}
```

Funkcija i deklaracije tipova – primer

```
#include <stdio.h>
#define MAX 50

typedef  int tNiz[MAX];

void unesiElementeNiza(int *pn, tNiz nizCelihBrojeva);
int sumirajNiz(int pn, tNiz nizCelihBrojeva);

int main()
{
   int n = 1;
   tNiz nizCelihBrojeva;
   printf("\n Program za racunanje sume elemenata niza N celih brojeva.\n\n");
   unsiElementeNiza(&n, nizCelihBrojeva);
   printf("\n Suma elemenata niza iznosi %d.\n\n", sumirajNiz(n, nizCelihBrojeva));
   return 0;
}
```

Funkcija i deklaracije tipova – primer

```
void unesiElementeNiza(int *pN, tNiz nizCelihBrojeva)
{
    int i;

    printf("Unesite N:\t");
    scanf("%d", pN);

    for (i = 0; i < *pN; i++)
    {
        printf("Unesite %d. element niza:\t", i+1);
        scanf("%d", &nizCelihBrojeva[ i ]);
    }
}

int sumirajNiz(int pn, tNiz nizCelihBrojeva)
{
    int i;
    int suma = 0;

    for (i = 0; i < pn; i++)
        suma += nizCelihBrojeva[ i ];

    return suma;
}
```

Funkcije – primeri

Zadatak 3:
Napraviti program koji prihvata podatke o polaznicima (ime, prezime, JMBG, grad), dobijene podatke sortira po JMBG-u i prikazuje ih korisniku. Podaci se smeštaju u niz, može biti maksimalno 40 polaznika. Implementirati unos, prikaz i sortiranje kao zasebne funkcije.

Vežba 4:
Modifikovati prethodnu vežbu tako da korisnik može da bira kriterijum po kojem će se vršiti sortiranje. Implementirati program oslanjajući se na funkcije.

Funkcije – Zadatak 3

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 40

typedef char tString30[31];
typedef char tString13[14];

typedef struct
{
   tString30 ime;
   tString30 prezime;
   tString13 jmbg;
   tString30 grad;
} tPolaznik;

typedef  tPolaznici tPolaznik[MAX];

void unesiPolaznike(tPolaznici polaznici, int *brPolaznika);
void prikaziPolaznike(tPolaznici polaznici, int brPolaznika, tString30 vrstaSortiranja);
void sortirajPolaznikePoJmbg(tPolaznici polaznici, int brPolaznika);
void zameni(tPolaznici polaznici, int koga, int kim);
```

Funkcije – Zadatak 3

```
int main()
{

    tPolaznici listaPolaznika;
    int brPolaznika = 0;

    printf("\n--Program za unos i sortiranje liste polaznika--\n\n");

    unesiPolaznike(listaPolaznika, &brPolaznika);

    sortirajPolaznikePoJmbg(listaPolaznika, brPolaznika);

    prikaziPolaznike(listaPolaznika, brPolaznika, "JMBG");

    return 0;
}
```

Funkcije – Zadatak 3

```
void unesiPolaznike(tPolaznici polaznici, int *brPolaznika)
{
    int i;
    printf("Unesite broj polaznika (maksimalno 40):\t");
    scanf("%d", brPolaznika);
    printf("\n-----------------------------------------------");
    for (i = 0; i < *brPolaznika; i++)
    {
        printf("\nUnesite podatke za %d. polaznika:\n", i+1);

        printf("\nIme:\t\t");
        scanf(" %[^\t\n]s", polaznici[i].ime);

        printf("\nPrezime:\t");
        scanf(" %[^\t\n]s", polaznici[i].prezime);

        printf("\nJMBG:\t\t");
        scanf(" %[^\t\n]s", polaznici[i].jmbg);

        printf("\nGrad:\t\t");
        scanf(" %[^\t\n]s", polaznici[i].grad);
        printf("\n-----------------------------------------------");
    }
}
```

Funkcije – Zadatak 3

```
void prikaziPolaznike(tPolaznici polaznici, int brPolaznika, tString30 vrstaSortiranja)
{
    int i;
    printf("\n\n\n\nLista polaznika sortirana prema %s:\n", vrstaSortiranja);
    printf("-----------------------------------------------");
    for (i = 0; i < brPolaznika; i++)
    {
        printf("\nR.br:\t\t%d", i+1);
        printf("\nIme:\t\t%s", polaznici[i].ime);
        printf("\nPrezime:\t%s", polaznici[i].prezime);
        printf("\nJMBG:\t\t%s", polaznici[i].jmbg);
        printf("\nGrad:\t\t%s", polaznici[i].grad);
        printf("\n-----------------------------------------------");
    }
}

void sortirajPolaznikePoJmbg(tPolaznici polaznici, int brPolaznika)
{
    int i, j;
    for(i = 0; i < brPolaznika-1; i++)
        for(j = i+1; j < brPolaznika; j++)
            if (strcmp(polaznici[i].jmbg, polaznici[j].jmbg) > 0)
                zameni(polaznici, i, j);
}
```

Funkcije – Zadatak 3

```
void zameni(tPolaznici polaznici, int koga, int kim)
{
    tPolaznik tmpPolaznik;

    strcpy(tmpPolaznik.ime, polaznici[koga].ime);
    strcpy(tmpPolaznik.prezime, polaznici[koga].prezime);
    strcpy(tmpPolaznik.jmbg, polaznici[koga].jmbg);
    strcpy(tmpPolaznik.grad, polaznici[koga].grad);

    strcpy(polaznici[koga].ime, polaznici[kim].ime);
    strcpy(polaznici[koga].prezime, polaznici[kim].prezime);
    strcpy(polaznici[koga].jmbg, polaznici[kim].jmbg);
    strcpy(polaznici[koga].grad, polaznici[kim].grad);

    strcpy(polaznici[kim].ime, tmpPolaznik.ime);
    strcpy(polaznici[kim].prezime, tmpPolaznik.prezime);
    strcpy(polaznici[kim].jmbg, tmpPolaznik.jmbg);
    strcpy(polaznici[kim].grad, tmpPolaznik.grad);
}
```

Funkcije i dinamički nizovi – Vektor

```
//vektor.h
// korisnicki tip dinamickog vektora koji u sebi sadrzi cele brojeve

#include <stdio.h>
#include <stdlib.h>

#define VEKTORMAX 100

typedef struct {
    unsigned int velicina;     // trenutna velicina vektora
    unsigned int kapacitet;    // maksimalni kapacitet vektora
    int *podaci;               // niz celih brojeva
} tVektor;

unsigned short inicijalizujVektor(tVektor *vektor);
unsigned short prosiriVektor(tVektor *vektor, int vrednost);
int pribaviVektor(tVektor *vektor, unsigned int indeks);
unsigned short postaviVektor(tVektor *vektor, unsigned int indeks, int vrednost);
unsigned short proveriKapacitetVektora(tVektor *vektor);
void oslobodiVektor(tVektor *vektor);
```

Funkcije i dinamički nizovi – Vektor

```
//vektor.c
#include "vektor.h"

unsigned short inicijalizujVektor(tVektor *vektor) {
    // inicijalna velicina i kapacitet
    vektor->velicina = 0;
    vektor->kapacitet = VEKTORMAX;

    // alociraj memoriju za vektor
    if (vektor->podaci = (int*)calloc(vektor->kapacitet, sizeof(int)))
        return 1;   // uspeo
    else
        return 0;   // nije uspeo
}

unsigned short prosiriVektor(tVektor *vektor, int vrednost) {
    if (proveriKapacitetVektora(vektor)){
        vektor->podaci[vektor->velicina++] = vrednost;
        return 1;
    }
    else
        return 0;
}
```

Funkcije i dinamički nizovi – Vektor

```
//vektor.c

int pribaviVektor(tVektor *vektor, unsigned int indeks) {
  if (indeks >= vektor->velicina || indeks < 0) {
      printf("Opseg vektora je %d\n", vektor->velicina);
      exit(1);
  }
  return vektor->podaci[indeks];
}

unsigned short postaviVektor(tVektor *vektor, unsigned int indeks, int vrednost) {
  // postavi na nule do zeljenog indeksa
  unsigned short uspeh = 1;
  while (indeks >= vektor->velicina) {
      uspeh = prosiriVektor(vektor, 0);
          if (!uspeh)
              break;
  }
  // ? while ((indeks >= vektor->velicina)&&(uspeh = prosiriVektor(vektor, 0));
  // postavi vrednost elementa na zeljenom indeksu
  if (uspeh)
      vektor->podaci[indeks] = vrednost;
  return uspeh;
}
```

Funkcije i dinamički nizovi – Vektor

```
// vektor.c

unsigned short proveriKapacitetVektora(tVektor *vektor) {
    unsigned short imaMesta = 1;
    if (vektor->velicina >= vektor->kapacitet) {
        // stiglo se do kraja vektora, dupliraj kapacitete
        vektor->kapacitet *= 2;
        if ((vektor->podaci = (int*)realloc(vektor->podaci, sizeof(int) * vektor->kapacitet))==NULL)
            imaMesta = 0;
        }
    return imaMesta;
}

void oslobodiVektor(tVektor *vektor) {
    free(vektor->podaci);
};
```

Funkcije i dinamički nizovi – Vektor

```
// main.c

#include "vektor.h"


int main() {
  tVektor vektor;

  unsigned int i;

  if (inicijalizujVektor(&vektor)){

      for (i = 105; i > 95; i--) {
          prosiriVektor(&vektor, i);
      }

      if (postaviVektor(&vektor, 5, 1023532))
          printf("\nUspelo dodavanje novog elementa (%d) u vektor.", pribaviVektor(&vektor, 5));
      else
          printf("\nNije uspelo dodavanje novog elementa u vektor.");

      oslobodiVektor(&vektor);
  }

  return 0;
};
```

Rekurzija i rekurzivne funkcije

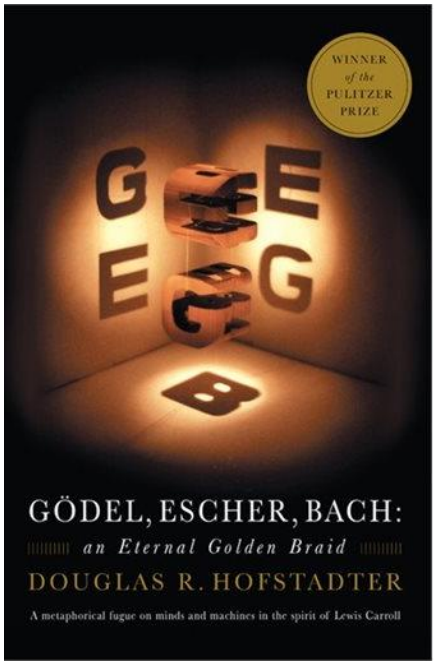
Rekurzija

Rekurzija nastaje kada se **pojam definiše pomoću sebe samog**

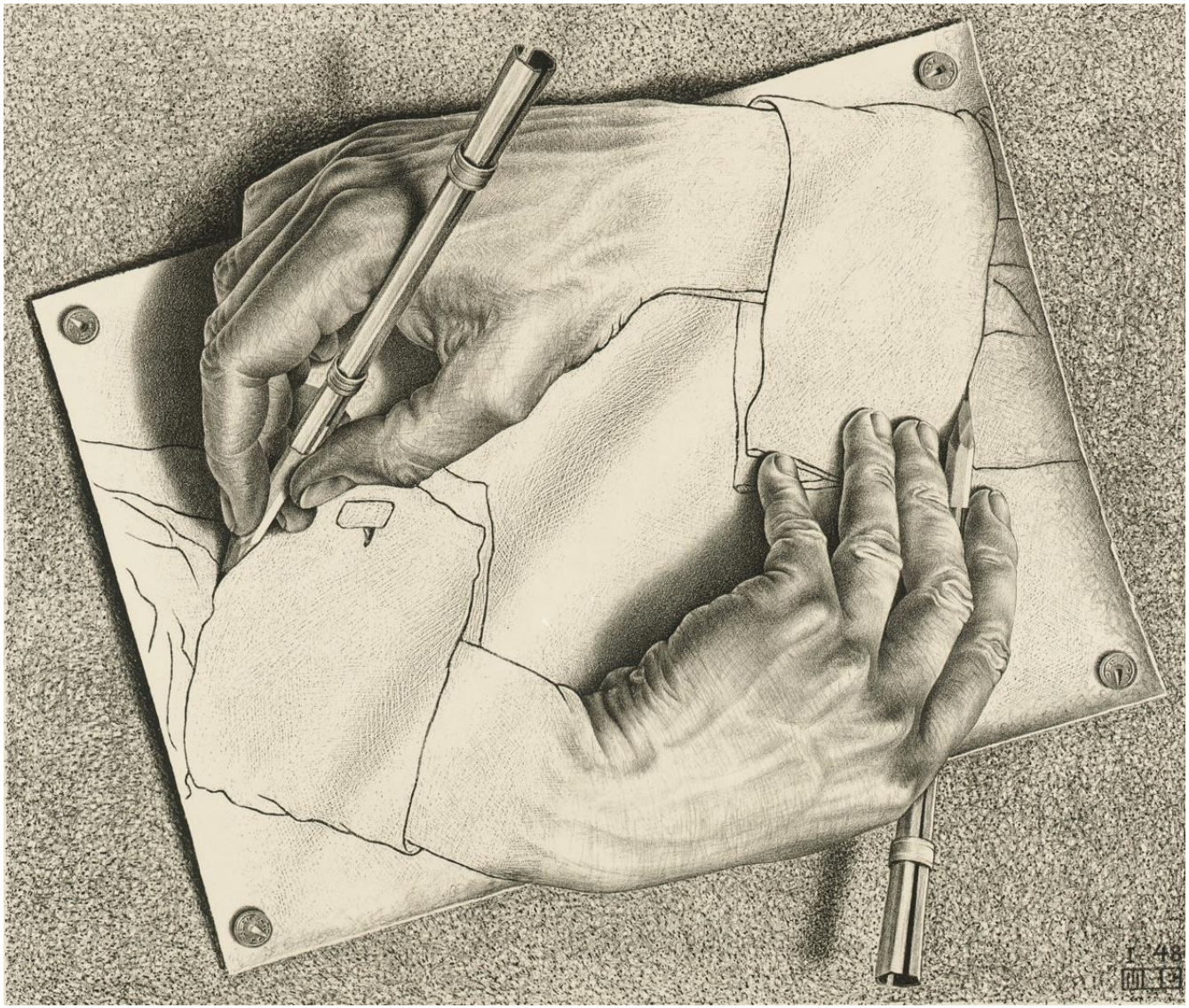
Javlja se u različitim oblastima, od lingvistike i logike, preko matematike i računarstva, do umetnosti

Primeri:

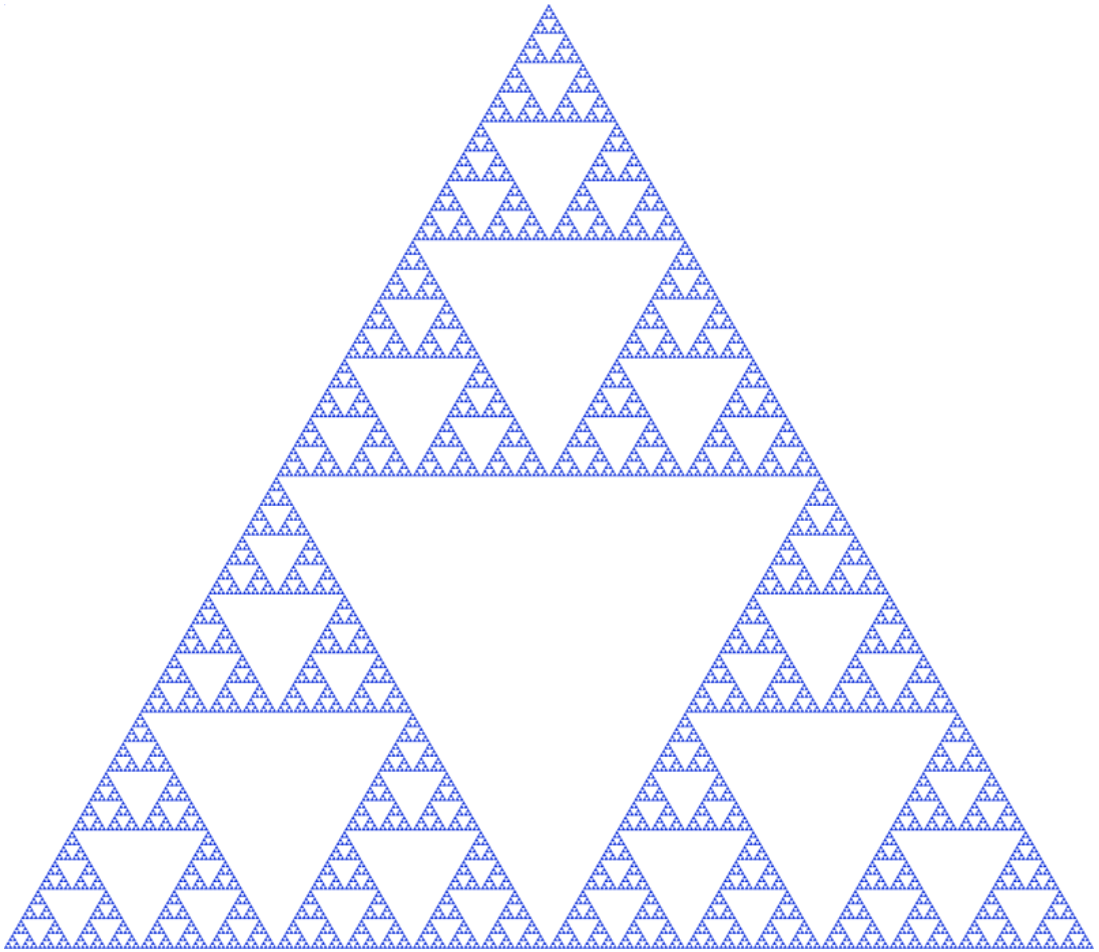
Binarno pretraživanje, faktorijel, fraktali, Fibonačijev niz, trougao Sierpinskog, Hanojske kule, ...



Rekurzija u umetnosti



Trougao Sierpinskog



Postupak konstrukcije:

Iteracija i rekurzija

Iterativna funkcija je ona koja koristi iteracije kako bi izvršila određeni kod veći broj puta, dok **rekurzivna funkcija** poziva samu sebe kako bi izvršila određeni kod veći broj puta

Svaka iteracija može se pretvoriti u rekurziju i obratno

Rekurzija se na nivou izvršavanja programa modeluje putem petlji (tj. uslovnih i bezuslovnih skokova) i steka

Rekurzija u matematici

Klasa objekata ili metoda **ponaša se rekurzivno** kada se može definisati pomoću sledeća dva svojstva:

1. Jednostavan **osnovni slučaj** – poseban slučaj koji ne koristi rekurziju kako bi proizveo odgovor

2. **Skup pravila** kojim se svi ostali slučajevi redukuju na osnovni slučaj

Rekurzivni procesi – **linearni** i u vidu **stabla**

Rekurzivni i iterativni procesi – iterativna realizacija rekurzivnih problema

Rekurzija u matematici

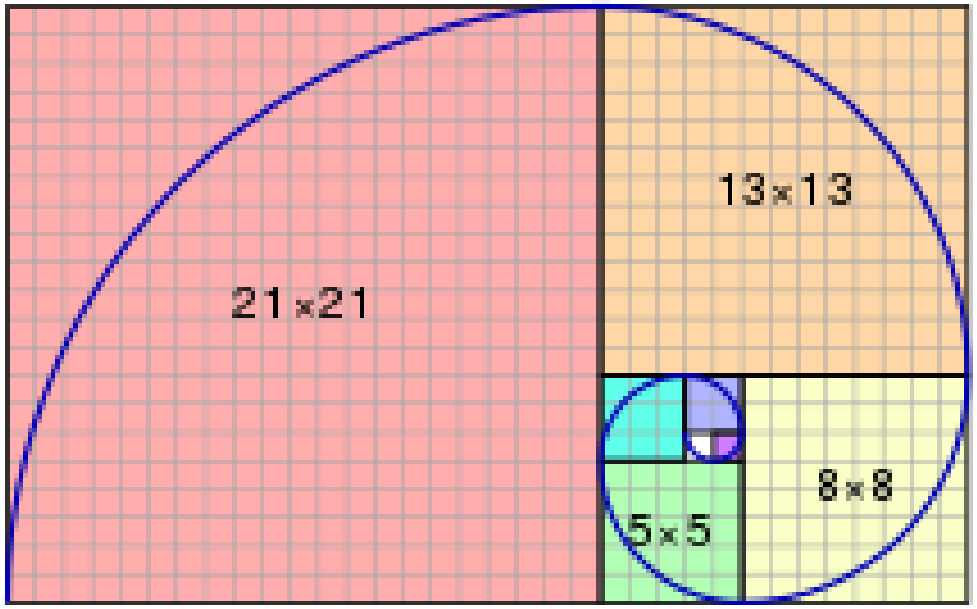
Primeri:

Računanje faktorijela

$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

Fibonačijev niz

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-1} + F_{n-2}, & n \geq 2 \end{cases}$$

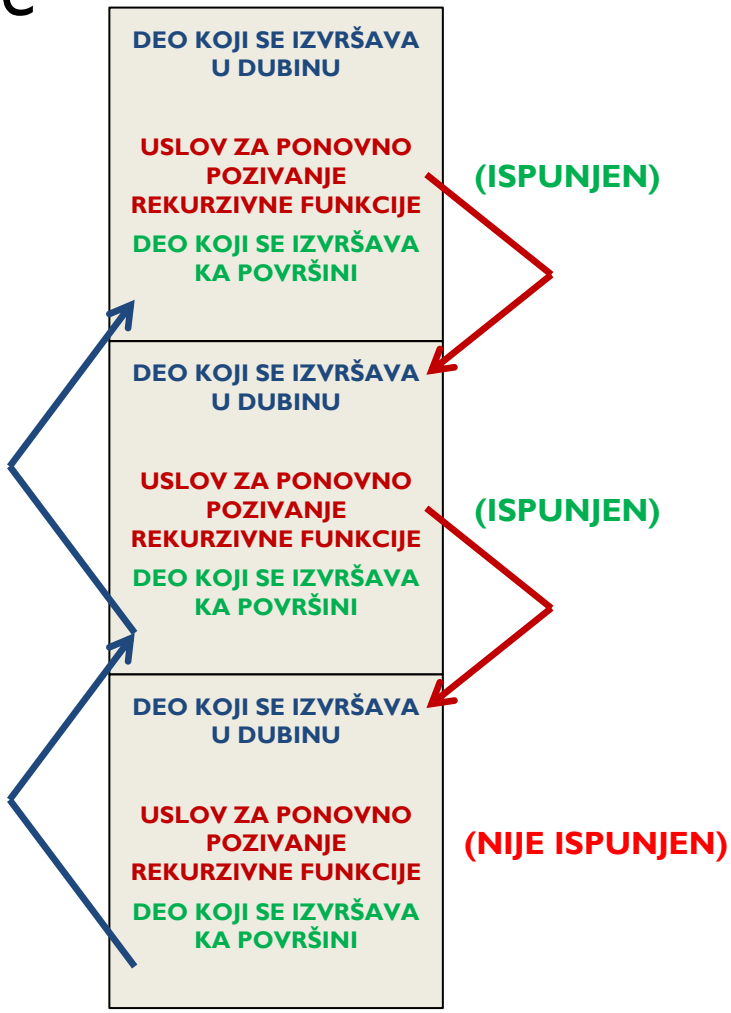


Rekurzivne funkcije

Situacija u kojoj je funkcija sama sebi i nadređena i podređena, tj. situacija kada funkcija poziva samu sebe

Postoje tri dela rekurzivne funkcije:

- **deo koji se izvršava u dubinu**
- **uslov za ponovno pozivanje rekurzivne funkcije**
- **deo koji se izvršava ka površini**

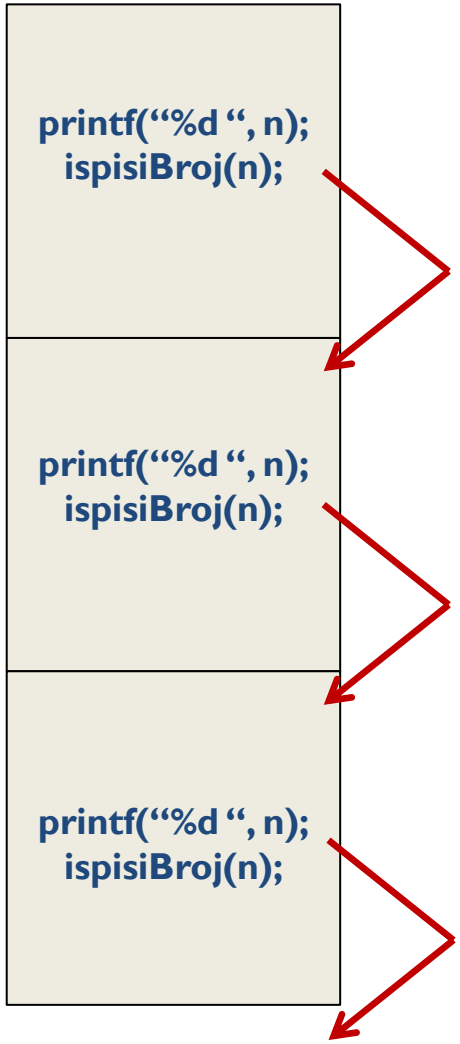


Rekurzivne funkcije – primer

Program za ispisivanje svih celih brojeva koji slede nakon navedenog broja do nule (isključujući nulu)

```
void ispisiBroj(unsigned n) {
    printf("%d ", n);
    ispisiBroj(n);        // rekurzivni poziv
}
```

Šta nije ispravno u kodu ove rekurzivne funkcije?



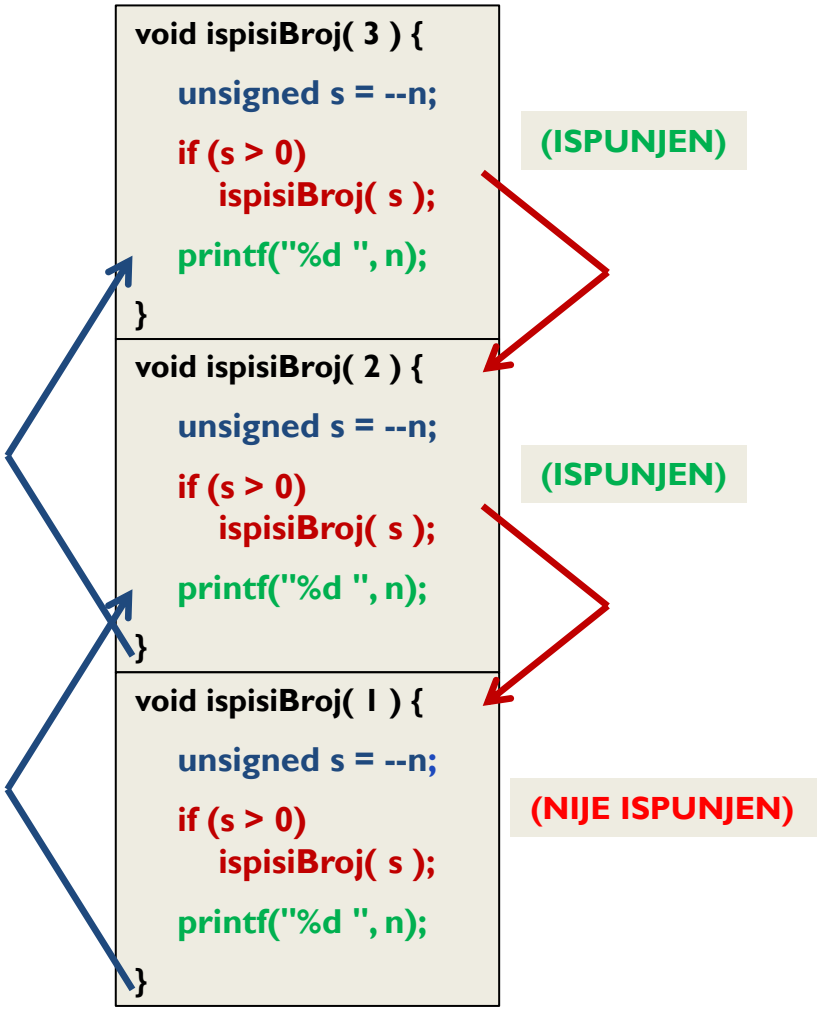
Rekurzivne funkcije – primer

Tačno rešenje:

```
void ispisiBroj(unsigned n) {
        unsigned s = --n;
        if (s > 0)
            ispisiBroj(s); //rekurzivni poziv
         printf("%d ", n);
    }
```

Primer poziva:

```
    int main( ) {
        ispisiBroj(3);
    }
```



Rekurzija – faktorijel

Rekurzivna definicija faktorijela:

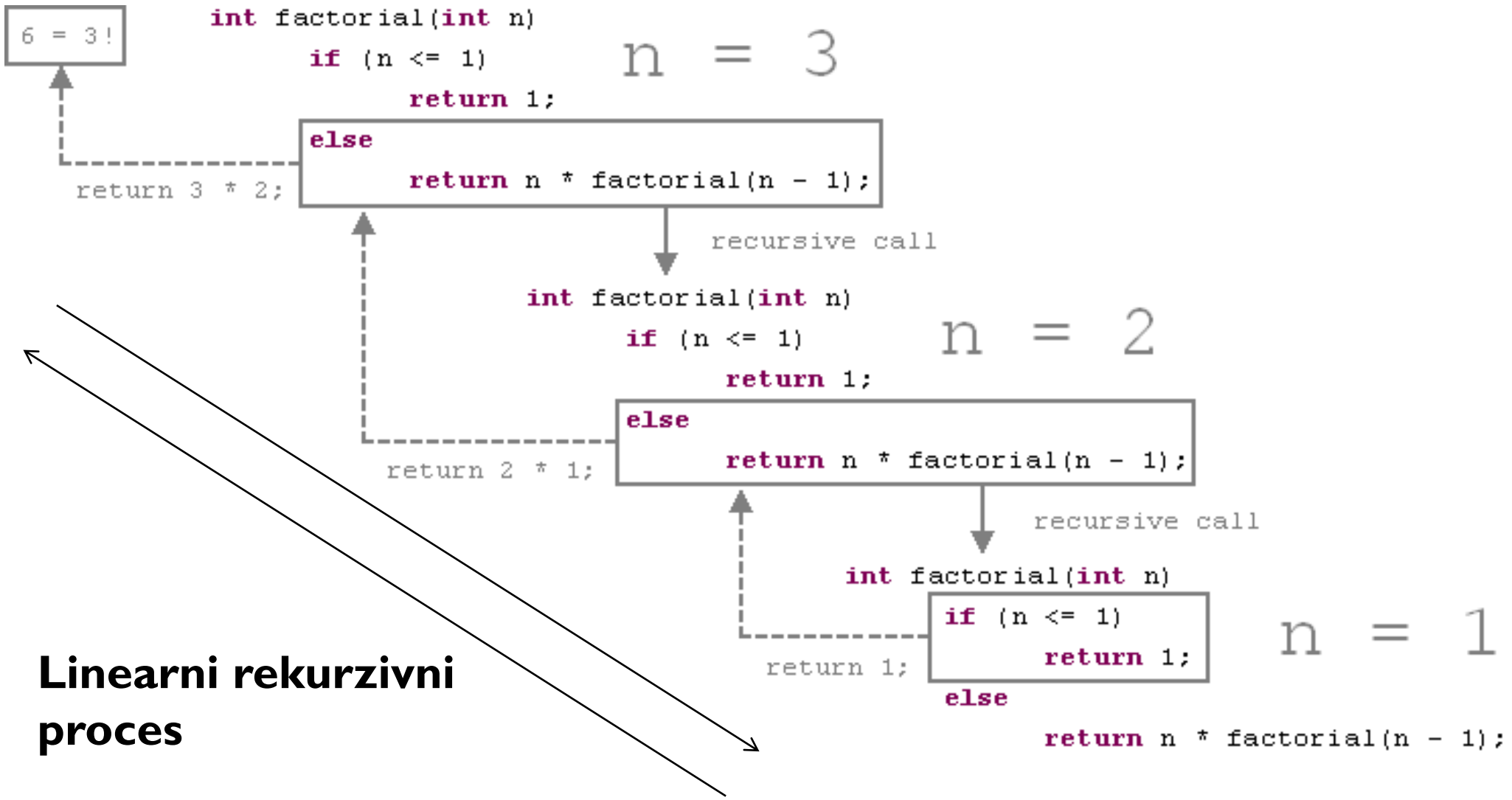
$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

Izlazak iz rekurzije omogućen je osnovnim slučajem ($n = 0$)

Rekurzivna funkcija za računanje faktorijela:

```
int faktorijel(int n) {
        if (n <= 1)
                return 1;
        else
                return n*faktorijel(n-1);      // rekurzivni poziv
}
```

Rekurzija – faktorijel



Linearni rekurzivni proces

Rekurzija – faktorijel

Iterativna funkcija za računanje faktorijela:

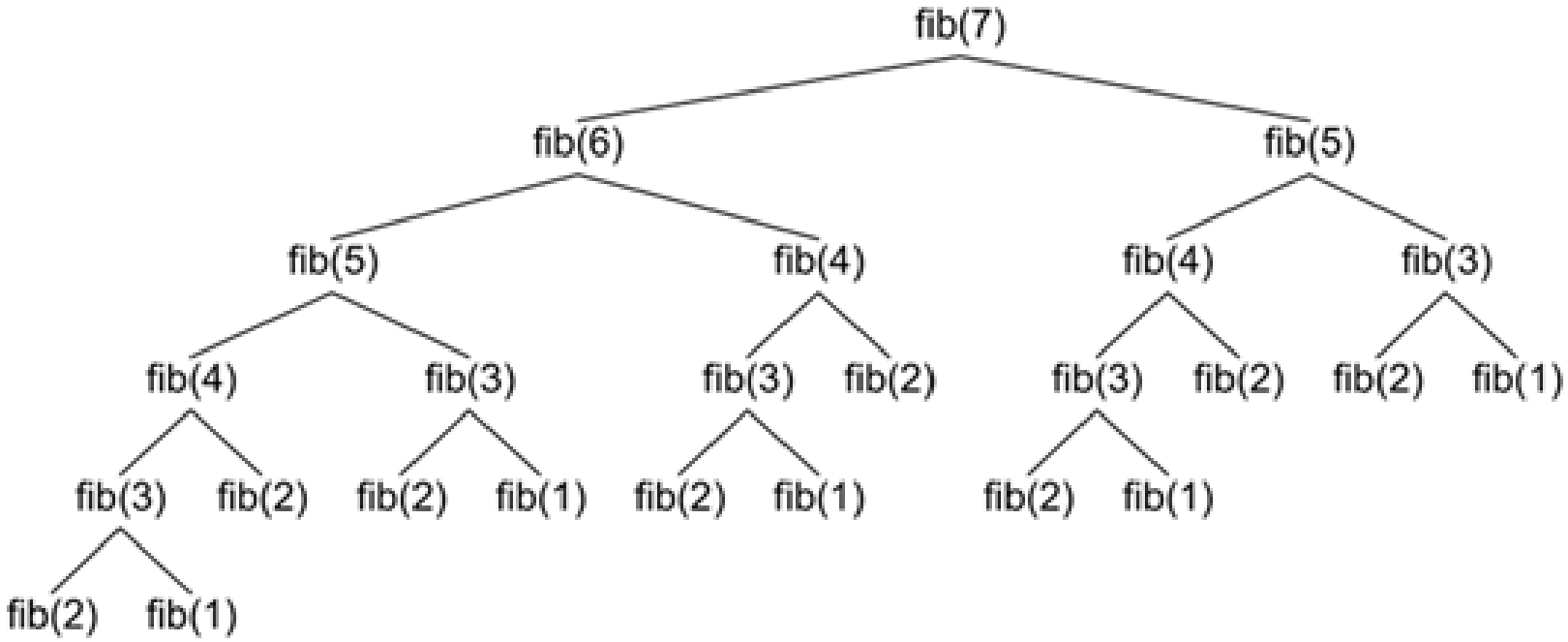
```
int faktorijel(int n) {
        int i;
        int fakt = 1;
        if (n == 1) return fakt;
        for (i = 2; i <= n; i++)
        {
                fakt*=i;
        }
        return fakt;
}
```

Rekurzija – Fibonačijev niz

Fibonačijevi brojevi

$$F_n = \begin{cases} 0, n = 0 \\ 1, n = 1 \\ F_{n-1} + F_{n-2}, n \geq 2 \end{cases}$$

Rekurzivni proces u vidu stabla (engl. *tree recursion*):



Rekurzija – Fibonačijev niz

Rekurzivna funkcija za generisanje Fibonačijevog niza:

```
int fibonacciRekurzivno(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacciRekurzivno(n - 1)+fibonacciRekurzivno(n - 2);
}
```

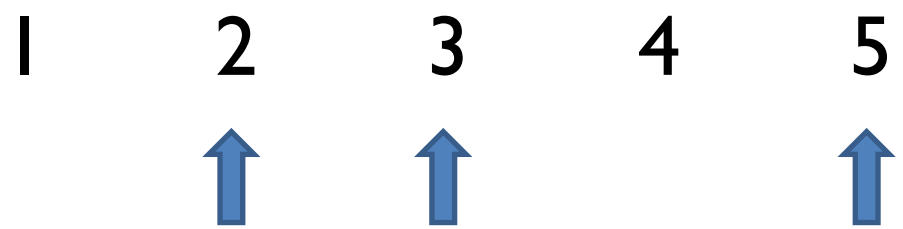
Rekurzija – Fibonačijev niz

Iterativna funkcija za generisanje Fibonačijevog niza:

```
int fibonacciIterativno(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    int pretPret = 0, pret = 1, rezultat = 0;
    for (int i = 2; i <= n; i++)
    {
        rezultat = pret + pretPret;
        pretPret = pret;
        pret = rezultat;
    }
    return rezultat;
}
```

Rekurzija – binarno pretraživanje

Binarno pretraživanje niza (engl. *binary search*)



Podeli-pa-vladaj (engl. divide-and-conquer) algoritam

bsearch() – deo standardne C biblioteke - stdlib.h

Različite podvarijante npr. uniformno binarno pretraživanje

Rekurzija – binarno pretraživanje

```
int trazi(int *podaci, int broj, int brojac)
{
   // pocetak = 0 (pocetni indeks)    kraj = brojac - 1 (krajnji indeks)
   return binarnoPretrazivanje(podaci, broj, 0, brojac-1);
}

int binarnoPretrazivanje(int *podaci, int broj, int pocetak, int kraj)
{
   //pronadji sredinu
   int sredina = pocetak + (kraj - pocetak)/2;   //celobrojno deljenje
   //uslov za zaustavljanje
   if (pocetak > kraj)
       return -1;
   else if (podaci[sredina] == broj)        //pronadjen?
       return sredina;
   else if (podaci[sredina] > broj)         //pod. veći od broja, trazi u "nizoj" polovini
       return binarnoPretrazivanje(podaci, broj, pocetak, sredina-1);
   else                                     //pod. je manji od broja, trazi u "visoj" polovini
       return binarnoPretrazivanje(podaci, broj, sredina+1, kraj);
}
```

Rekurzija – Hanojske kule

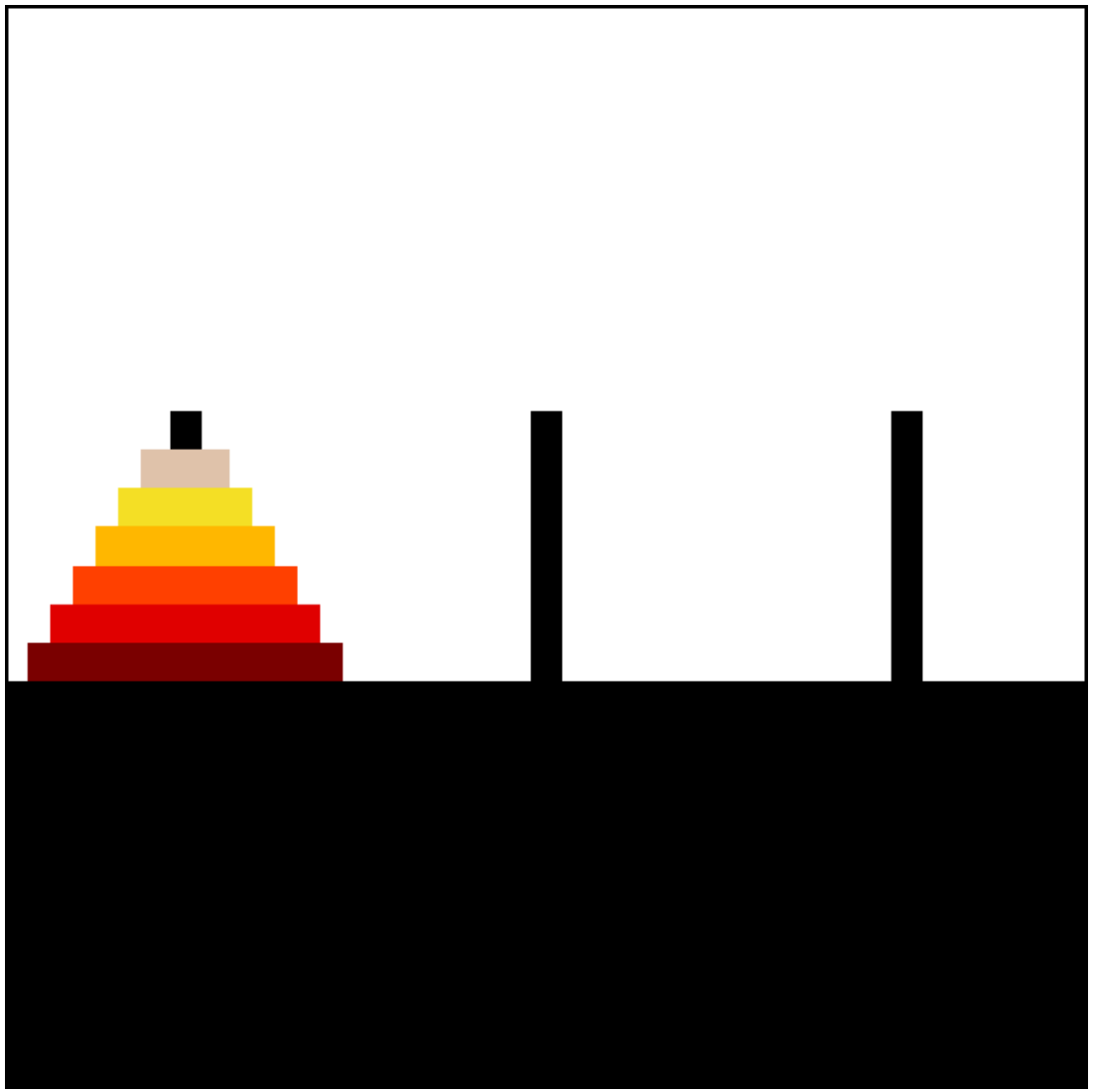
Tri štapa i diskovi različite veličine

Cilj je prebaciti sve diskove sa jednog na drugi štap poštujući sledeća pravila:

- Svakim potezom pomera se samo po jedan disk
- Svaki potez sastoji se od uzimanja najvišeg diska sa jednog štapa i njegovog prebacivanja na vrh drugog štapa
- Nijedan disk ne sme biti stavljen na manji disk

$$h_n = \begin{cases} 1, n = 1 \\ 2h_{n-1} + 1, n > 1 \end{cases}$$

Rekurzija – Hanojske kule



Izvor: https://commons.wikimedia.org/wiki/File:Iterative_algorithm_solving_a_6_disks_Tower_of_Hanoi.gif

Rekurzija – Hanojske kule

```
#include <stdio.h>

void hanojskeKule(int, char, char, char);

int main(){
    int broj;
    printf("Unesite broj diskova: ");
    scanf("%d", &broj);
    printf("Redosled poteza je:\n");
    hanojskeKule(broj, 'A', 'C', 'B');
    return 0;
}

void hanojskeKule(int broj, char saStapa, char naStap, char pomocuStapa){
    if (broj == 1){
        printf("\n Prebaci disk 1 sa stapa %c na stap %c", saStapa, naStap);
        return;
     }
     hanojskeKule(broj - 1, saStapa, pomocuStapa, naStap);
     printf("\n Prebaci disk %d sa stapa %c na stap %c", broj, saStapa, naStap);
     hanojskeKule(broj - 1, pomocuStapa, naStap, saStapa);
}
```