


Programiranje I

Rekurzivne funkcije



Rekurzija - osnovni pojmovi

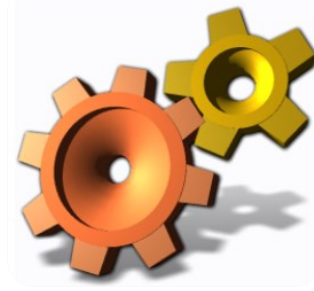
- Funkcija poziva samu sebe
- Mora postojati završetak (bazni slučaj) 
- Neki jezici (npr. FORTRAN) ne podržavaju rekurziju
- Rekurzivni programi su kraći, ali izvođenje programa je duže
- Koristi se struktura podataka stog (stek) za pohranjivanje rezultata i povratak iz rekurzije.
- **recur** latinski:
 - re** = *nazad*
 - currere** = *izvršavati* - desiti se opet, u ponovljenim intervalima
- Mnoge matematičke funkcije se mogu definisati rekurzivno:
 - *Faktorijel*
 - *Fibonacijevi brojevi*
 - *Euklidov NZD (najveći zajednički djelilac)*
 - *Kule Hanoja*
 - ...



Rekurzija - osnovni pojmovi

- Rekurzija je postupak rješavanja zadatka, u kome neka funkcija poziva samu sebe. Tom prilikom se vodi računa da se svaki naredni poziv izvršava za jednostavniji problem od polaznog, a da se najjednostavniji problemi rješavaju direktno, tj. bez dalje upotrebe rekurzije.
- Pri rješavanju zadatka rekurziju je najbolje shvatiti i koristiti na sljedeći način: *potrebno je znati neposredno riješiti najjednostavniji slučaj datog problema, te da se složeniji slučajevi svedu na jednostavnije, tj. da se riješe koristeći jednostavnije slučajeve. Tom prilikom se ne treba brinuti o tome kako će biti riješeni jednostavniji slučajevi na koje se svodi složeni slučaj (upravo u tom kontekstu rekurzija radi za nas).*

Primjer



- U nastavku je prikazan primjer izračunavanja sume kvadrata cijelih brojeva u rasponu od n do m – iteracijom i rekurzijom

//RJESENJE ITERACIJOM

```
int SumaKvadrata(int m, int n){  
    int suma = 0;  
    for (int i = m; i <= n; i++)  
        suma += i*i;  
    return suma;  
}
```

//RJESENJE REKURZIJOM

```
int SumaKvadrata(int m, int n){  
    if(m > n)  
        return 0;  
    return m*m + SumaKvadrata(m+1, n);  
}
```



Stog (Stack)

- Predstavlja strukturu podataka kod koje se posljednji pohranjeni podatak uzima prvi u obradu.
- Može se realizovati statičkom strukturom podataka. U jednodimenzionalno polje zadane strukture dodaju se ili brišu pojedine stavke po principu *LastInFirstOut (LIFO)*.
- Dodavanje (*push*) i brisanje (*pop*) elemenata s vrha (*top*):
 - Pojedina operacija *dodaj* ili *brisi* zahtijeva jednako vremena bez obzira na broj pohranjenih podataka.
 - Slučaj da je stog pun može zahtijevati alociranje dodatne memorije i ponovno izvođenje programa. Prazan stog ne mora značiti grešku.



Stog (Stack)

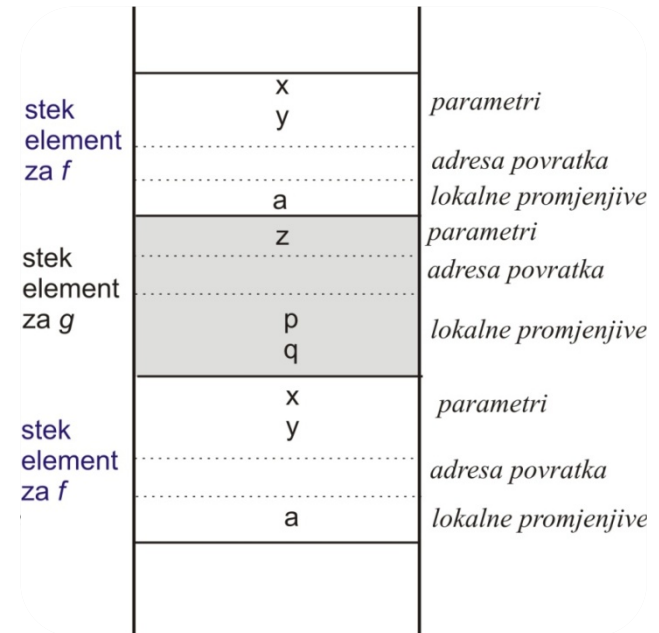
- Uobičajeno je da programi kompajlirani u modernim jezicima visokog nivoa koriste stek pozive kao radnu memoriju svake pozvane funkcije.
- Kada se pozove neka funkcija, određeni broj parametara poziva se stavlja na stek poziva. Nakon povratka iz funkcije u pozivajuću funkciju (npr. main), parametri poziva se uklanjaju sa steka
- Kada funkcija poziva drugu funkciju, najprije se njeni argumenti, zatim adresa povratka i konačno prostor za lokalne promjenljive stavljaju na stek poziva. Pošto svaka funkcija radi u sopstvenom *okruženju* ili *kontekstu*, postaje moguće da funkcija pozove samu sebe. Ova mogućnost je veoma korisna - jer se mnogi problemi elegantno specificiraju ili rješavaju na *rekurzivan* način



Stog (Stack)

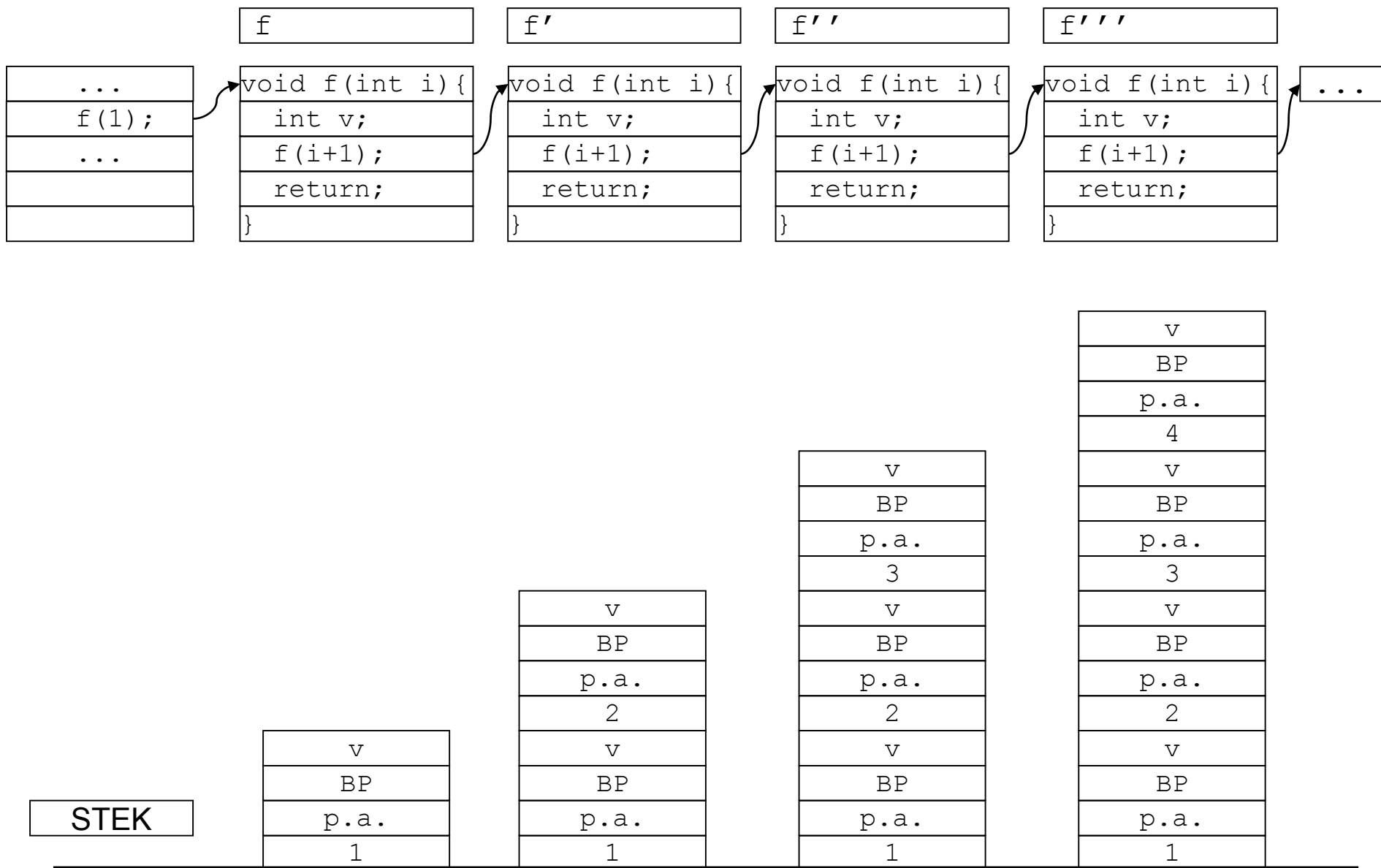
- Stek poziva poslije izvršavanja para uzajamno rekurzivnih funkcija:

```
int f(int x, int y) {  
    int a;  
    if (uslov_prekida)  
        return ...;  
    a = .....;  
    return g(a);  
}  
  
int g(int z) {  
    int p, q;  
    p = ...;  
    q = ...;  
    return f(p, q);  
}
```



- Vidimo da se čitava okruženja funkcija ***f*** i ***g*** (njihovi parametri i lokalne promjenljive) nalaze na steku poziva. Kada se funkcija ***f*** pozove po drugi put, iz funkcije ***g***, kreira se novi format poziva za poziv funkcije ***f***.

Primjer poziva funkcija na steku





Izračunavanje faktoriijela

- Jedan od jednostavnih rekurzivnih algoritama jest izračunavanje $n!$ za $n \geq 0$.

$$0! = 1$$

$$1! = 1$$

$$n! = n * (n - 1)!$$

Primjer: $4!$

`k = fakt (4);`

`= 4 * fakt (3);`

`= 3 * fakt (2);`

`= 2 * fakt (1);`

`= 1`

```
int Faktorijel(int n) {  
    if(n<=1)  
        return 1;  
    return n * Faktorijel(n-1);  
}
```



Izračunavanje faktoriijela

$$5! = 5 \times 4! = 5 \times 24 = 120$$

$$\downarrow$$
$$4! = 4 \times 3! = 4 \times 6 = 24$$

$$\downarrow$$
$$3! = 3 \times 2! = 3 \times 2 = 6$$

$$\downarrow$$
$$2! = 2 \times 1! = 2 \times 1 = 2$$

$$\downarrow$$
$$1! = 1 \times 0! = 1 \times 1 = 1$$

$$\downarrow$$
$$0! = 1$$

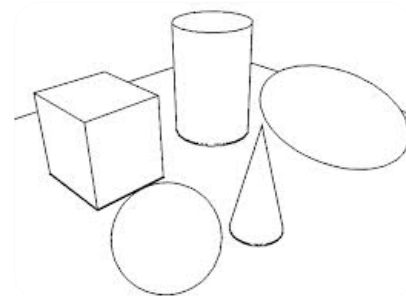
Fibonacci niz



- Malo geometrije
 - Nacrtati kvadrat veličine 1.
 - Rotirati za 90 stepeni, i dodati novi kvadrat veličine 1.
 - Rotirati, ponovo, za 90 stepeni, i dodati kvadrat veličine 2.
 - Ponovo rotirati i dodati kvadrat veličine 3, itd.
 - ...
 - Prethodne akcije nastaviti prateći niz u kome je naredni član niza jednak zbiru dva prethodna člana:

1, 1, 2, 3, 5, 8, 13, 21, . . . ,

- Šta smo dobili?



Fibonacci niz



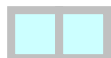
1



Fibonacci niz



1



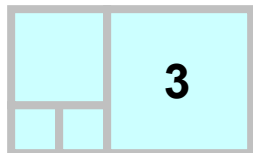
Fibonacci niz



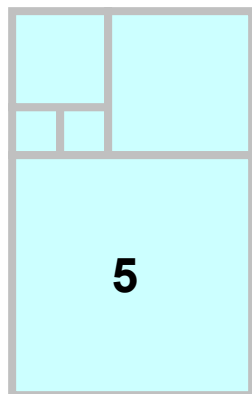
2



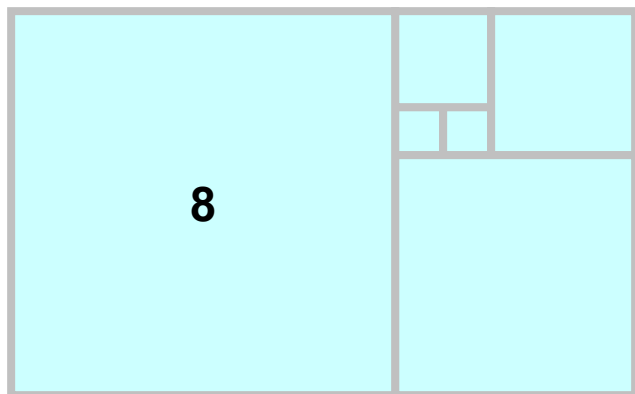
Fibonacci niz



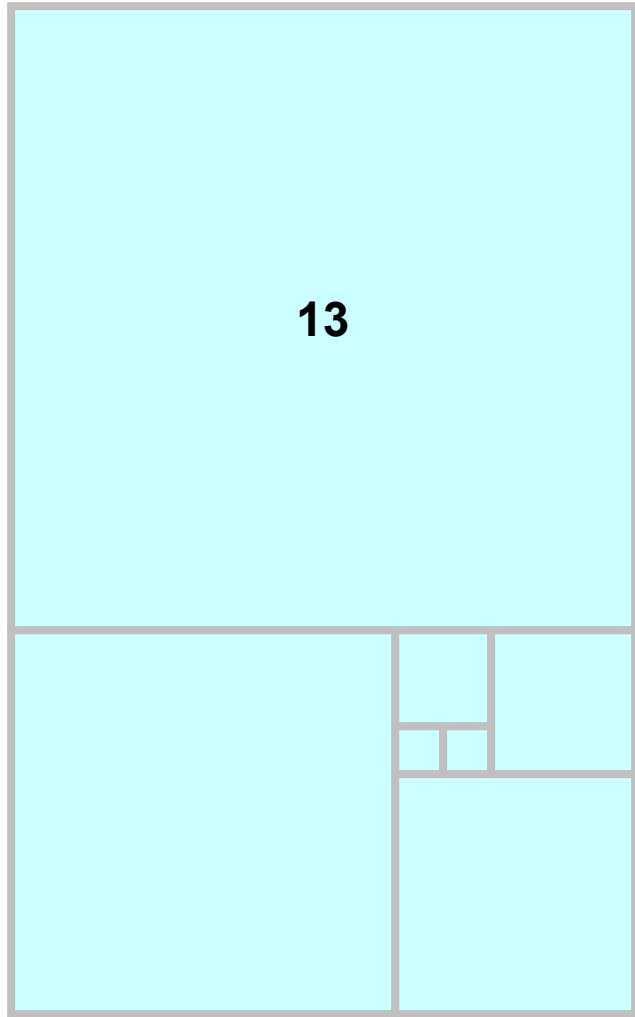
Fibonacci niz



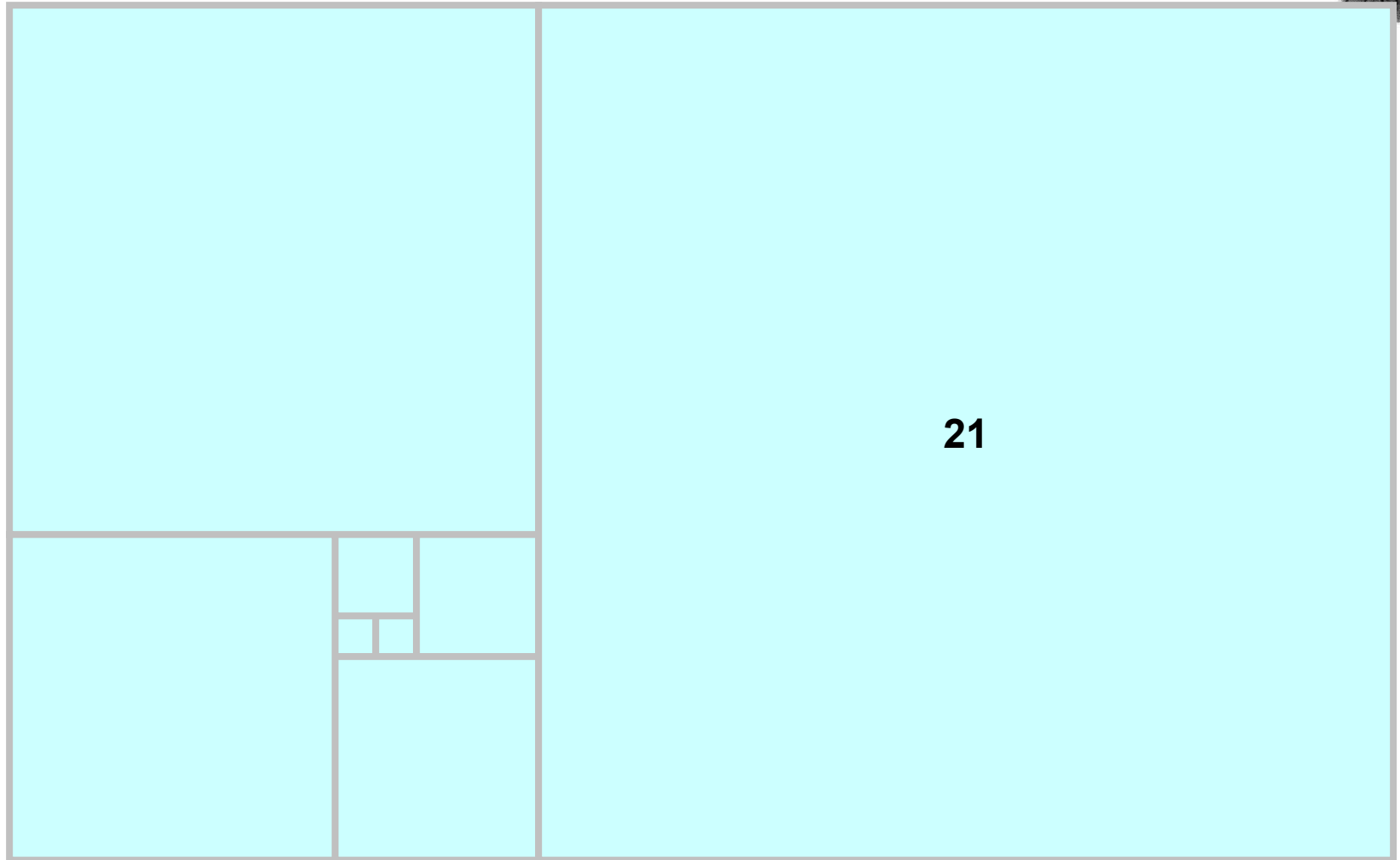
Fibonacci niz



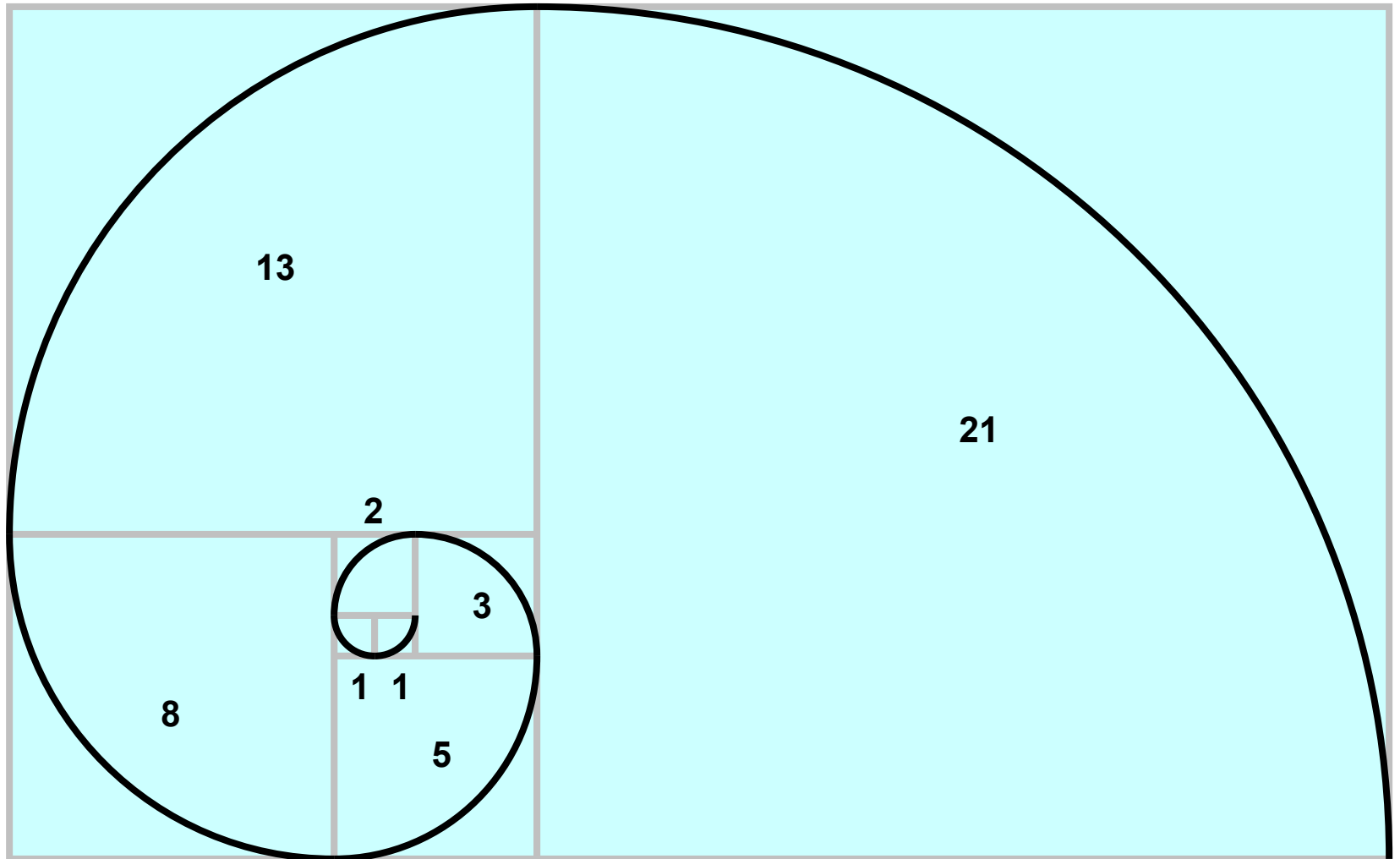
Fibonacci niz



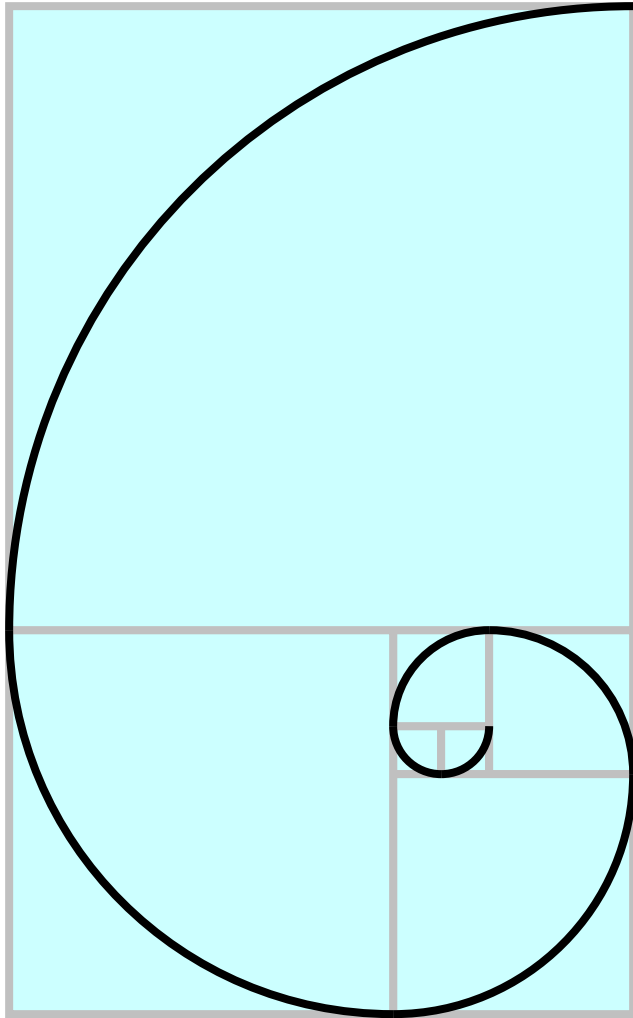
Fibonacci niz



Fibonacci niz

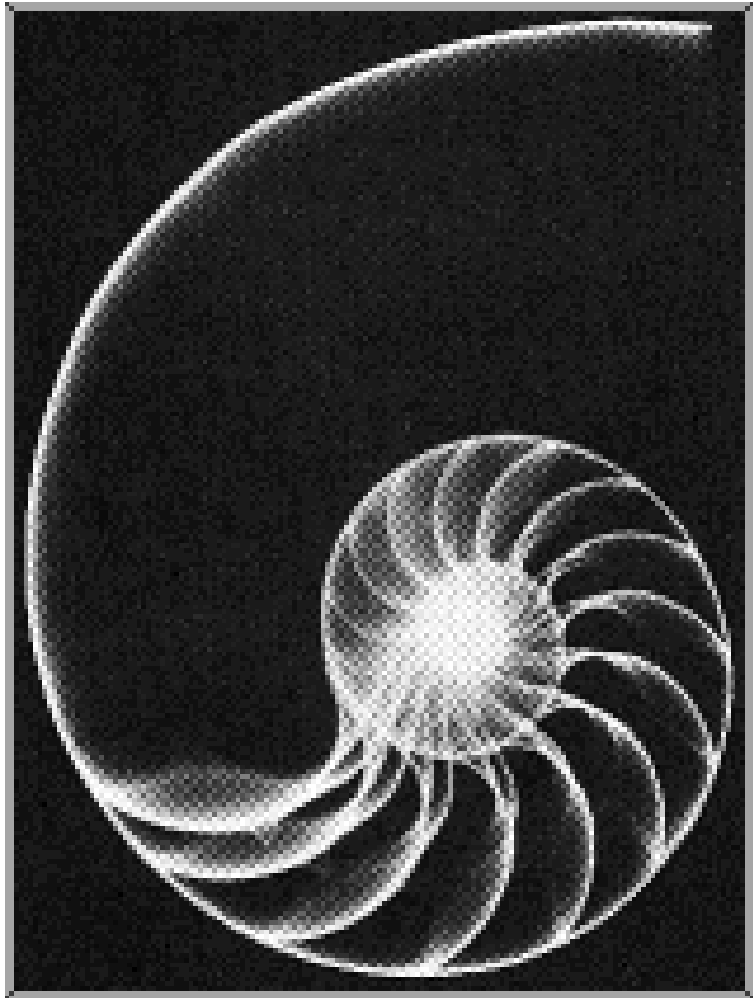
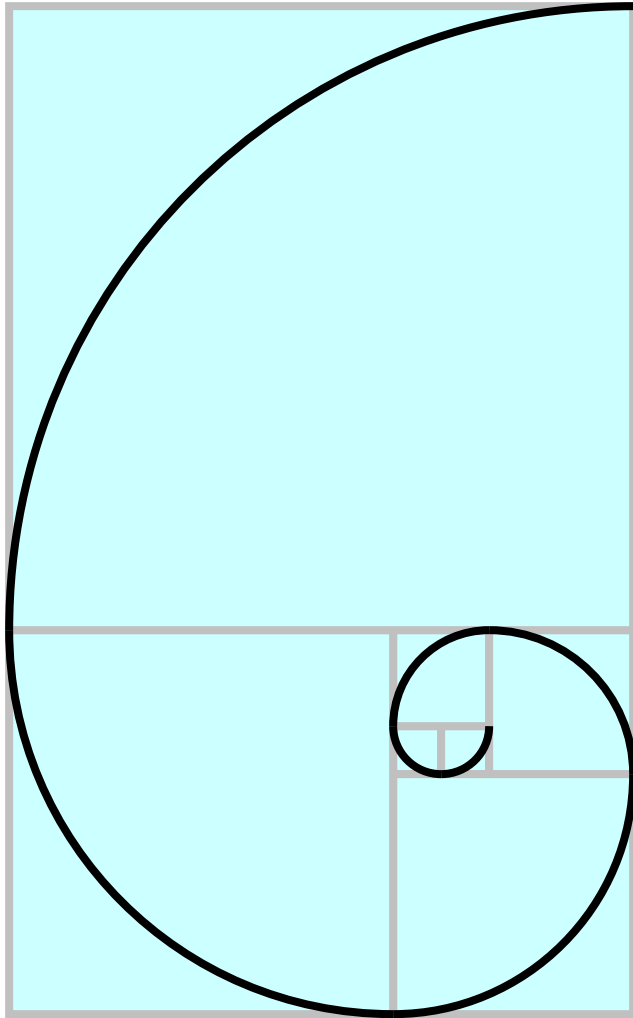


Fibonacci niz

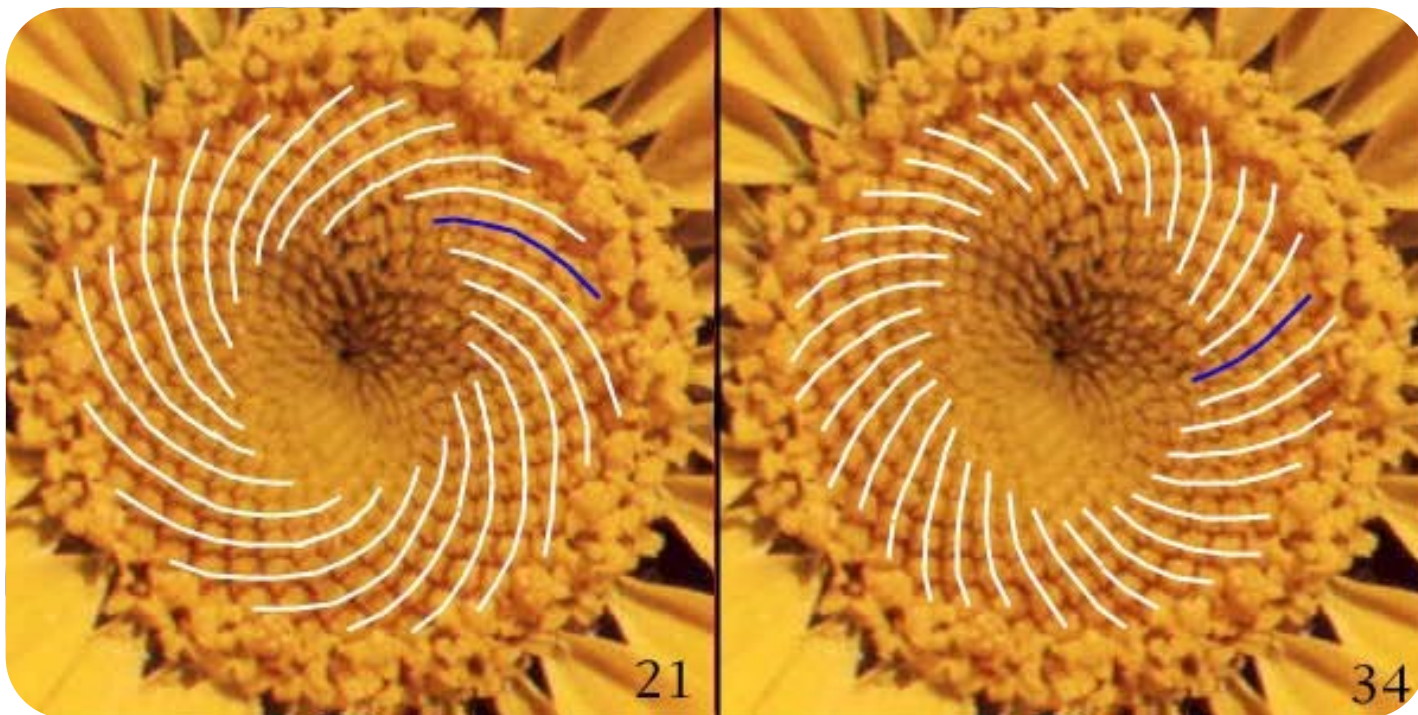


Da li ste ovaj oblik negdje već vidjeli?

Fibonacci niz



Fibonacci niz





Fibonacci niz - formula

- Da li je na osnovu datog niza:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... ,

moguće napisati formulu za bilo koji n ?

$$Fib(n) = \left[\frac{1 + \sqrt{5}}{2} \right]^n - \left[\frac{1 - \sqrt{5}}{2} \right]^n$$

$$Fib(n) = Fib(n - 1) + Fib(n - 2)$$

$$Fib(0) = 0;$$

$$Fib(1) = 1;$$

Fibonacci niz - primjer



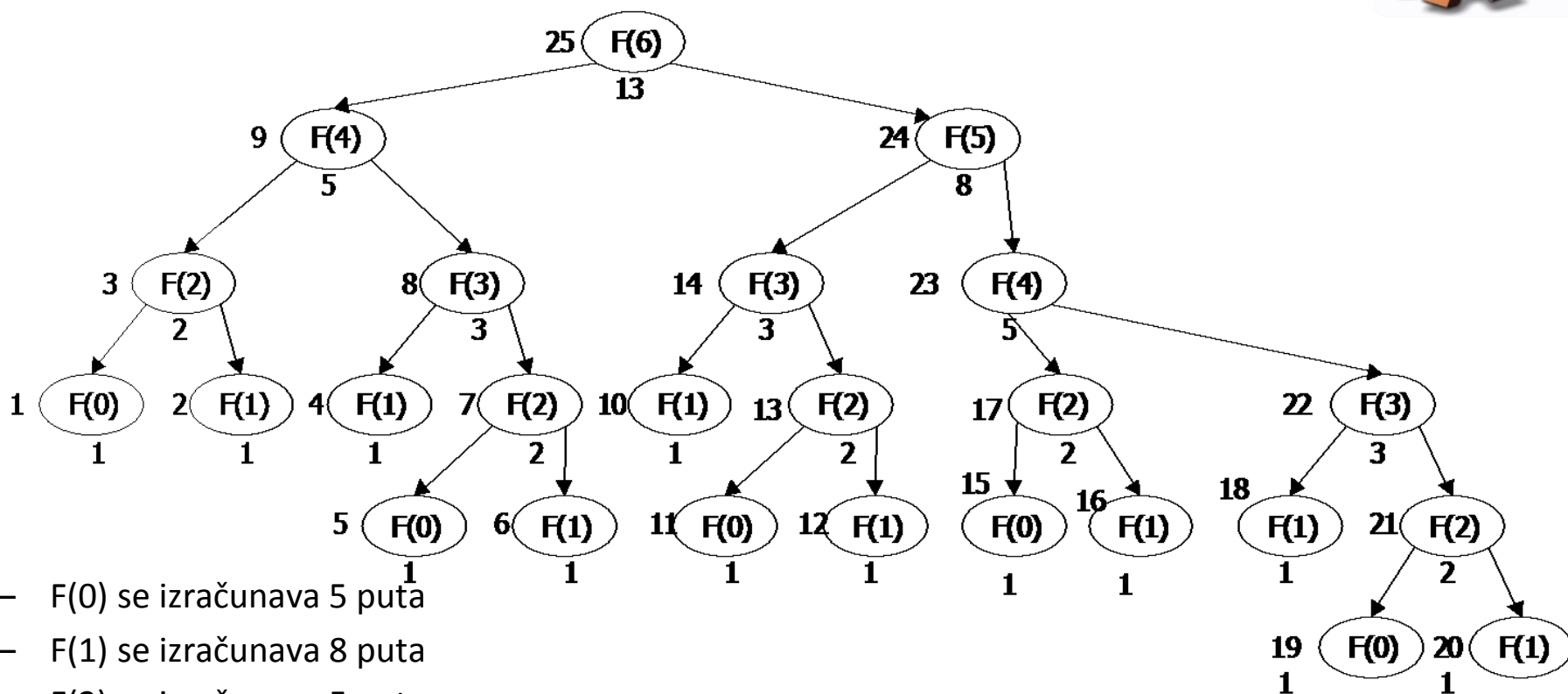
```
int Fibonacci (int n) {  
    if (n <= 1)  
        return 1;  
    return Fibonacci(n-2) + Fibonacci(n-1);  
}
```

$Fib(n) = Fib(n - 1) + Fib(n - 2)$
 $Fib(0) = 0;$
 $Fib(1) = 1;$

```
void main() {  
    int brojClanova = 0;  
    cout<<"Koliko clanova niza zelite: ";  
    cin>>brojClanova;  
    cout<<"Fibonacci niz -> ";  
    for (int i = 0; i < brojClanova; i++)  
        cout<<Fibonacci(i)<<", ";  
    cout<<endl;  
}
```



Fibonacci niz - primjer



- F(0) se izračunava 5 puta
- F(1) se izračunava 8 puta
- F(2) se izračunava 5 puta
- F(3) se izračunava 3 puta
- F(4) se izračunava 2 puta
- F(5) se izračunava 1 puta
- F(6) se izračunava 1 puta
- Ukupno izračunavanja za F(6): 25

Rekurzija



Remember!

- Nekoliko napomena vezanih za rekurziju:
 - ✓ Rekurzija postoji svuda u prirodi
 - ✓ Za svaki problem i ne postoji formula, ali većina problema može biti predstavljena kao serija malih ponovljenih koraka
 - ✓ Svaki korak u rekurzivnom procesu treba da bude mali i jednostavan (izračunljiv)
 - ✓ Apsolutno je neophodan uslov koji završava proces (bazni slučaj)
 - ✓ Tipični primjeri korištenja rekurzije su izračunavanje faktoriijela i članova Fibonacci niz
 - ✓ Mnoge formule i definicije su same po sebi rekurzivne, to treba iskoristiti!
 - ✓ Rekurzija je dobra za izradu algoritama za rekurzivno definisane strukture podataka kao što su binarna stabla. Mnogo lakše i pogodnije od iteracije!
 - ✓ Rekurzija je idealna za svaki “divide & conquer” algoritam



Funkcija `pow(x, y)`

- Koristeći rekurziju napisati funkciju koja će simulirati rad funkcije `pow`.

$$Pow(x, y) = x^y$$

$$x^1 = x$$

$$x^0 = 1$$

```
double pow(double vrijednost, int eksponent) {  
    if (eksponent == 0)  
        return 1;  
    else if (eksponent == 1)  
        return vrijednost;  
    else  
        return vrijednost * pow (vrijednost, eksponent - 1);  
}
```

Kule Hanoja (The Towers of Hanoi)



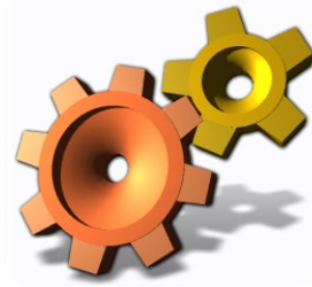
- Osnovna pravila:

- Kada se disk premješta postavlja se na jedan od stubova.
- Može se premještati samo po jedan disk i on mora biti na vrhu diskova na stubovima
- Veći disk ne može biti postavljen na manji.



- *Legenda kaže da kada svećenici premjeste 64 diska nastupa kraj svijeta*

Kule Hanoja (The Towers of Hanoi)



- Moguće rekurzivno rješenje za problem Kula Hanoja

```
void HanoiKule(int brojDiskova, char start, char pomocni,
               char destinacija)
{
    if (brojDiskova == 0)
        return;
    HanoiKule(brojDiskova - 1, start, destinacija, pomocni);
    cout<<"Disk sa tornja "<<start<<" na toranj "<<destinacija<<endl;
    HanoiKule(brojDiskova - 1, pomocni, start, destinacija);
}

void main() {
    HanoiKule(6, 'A', 'C', 'B');
}
```

- Program proširiti na način da onemogući izvršenje za 64 diska





Najveća zajednički djelilac (nzd)

- Jedan od najstarijih algoritama je Euklidov postupak za pronalaženje najvećeg zajedničkog djelioca (nzd) dva pozitivna cijela broja. Naprimjer:

$\text{nzd}(22, 8) = \text{nzd}(8, 6) = \text{nzd}(6, 2) = \text{nzd}(2, 0) = \mathbf{2}$

$\text{nzd}(21, 13) = \text{nzd}(13, 8) = \text{nzd}(8, 5) = \text{nzd}(5, 3) = \text{nzd}(3, 2) = \text{nzd}(2, 1) = \text{nzd}(1, 0) = \mathbf{1}$

$\text{nzd}(21, 0) = \mathbf{21}$

$\text{nzd}(0, 21) = \text{nzd}(21, 0) = \mathbf{21}$

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

Prednosti i nedostaci rekurzije



- Prednosti:
 - koncizniji opis algoritma
 - lakše je dokazati korektnost
- Nedostaci:
 - uvećano vrijeme izvođenja
 - neki jezici ne podržavaju rekurziju



KRAJ PREZENTACIJE