Iteracije in rekurzija

Mehanizmi krmiljenja poteka

- Zaporedno izvajanje (prednost izvedbe izrazov)
- Izbira (Selection)
- **Iteracije**
- Proceduralna abstrakcija
- **Rekurzija**
- Konkurenčnost
- Nedeterminizem

*Ta dva krmilna mehanizma dovolita računalniku **ponavljanje izvajanja** iste množice operacij*

- *Imperativni jeziki v glavnem temeljijo na iteracijah*
- *Funkcionalni jeziki v glavnem temeljijo na rekurziji.*

Kaj je iteracija?

Iteracija v računalništvu je ponavljanje procesa v računalniškem programu. Pri tem se neko stanje spreminja.

Iteracija pomeni stil programiranja v imperativnih programskih jezikih. V nasprotju s tem ima rekurzija bolj deklarativni pristop.

Primer iteracije, opisan v imperativni psevdokodi:

```
var i, a := 0 // initialize a before iteration f
For i from 1 to 3 { // loop three times
    a := a + i // increment a by the current value of i
}
print a // the number 6 is printed
```

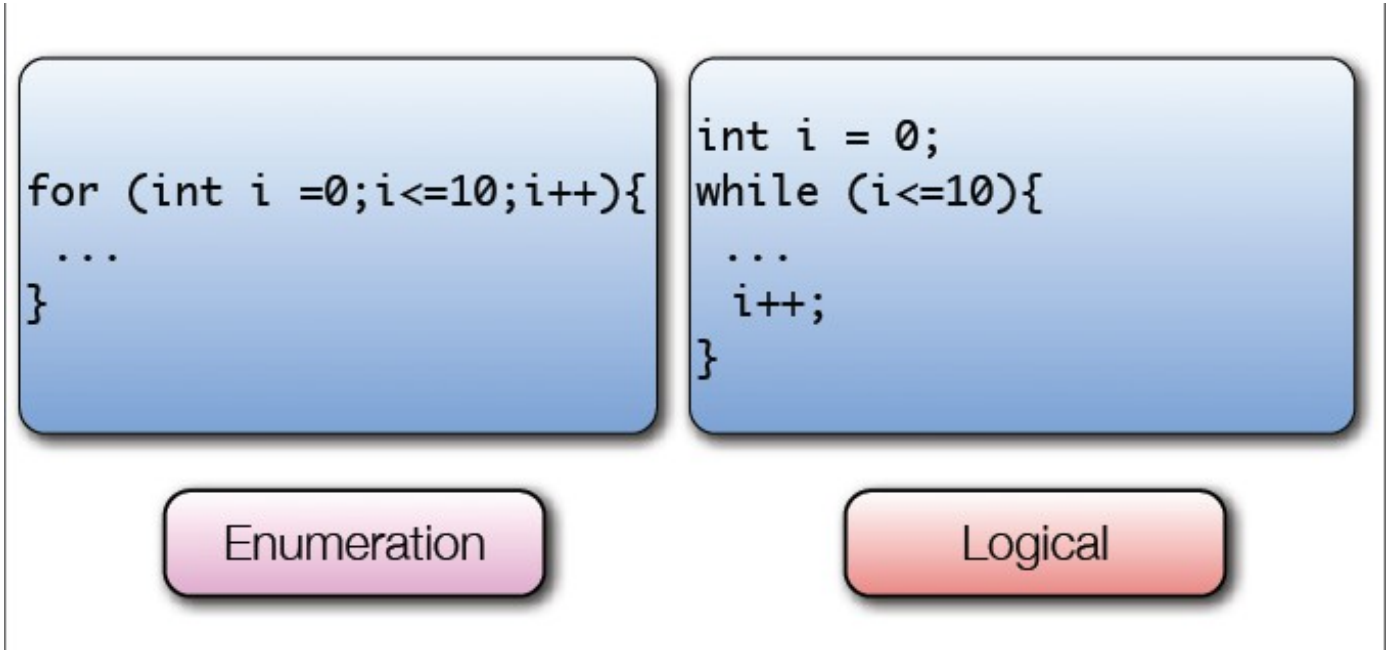
Iteracije

Iteracija ima običajno obliko zanke

•Dve osnovni obliki:

- Iteracija preko oštevilčenih množic**.**
- Iteracijo krmilimo z logiko (dokler ni izpolnjen nek pogoj)

Primeri:



Primer: Fibonaccijeva števila

- Fibonaccijeva števila tvorijo naslednje zaporedje

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

n 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,...

- Definiramo jih lahko tudi rekurzivno:

fib(1) = 1

fib(2) = 1

fib(n) = fib(n – 1) + (fib(n – 2) za n > 2

- To zaporedje je znano tudi kot problem množečih se zajčkov ☺

Primer: plenilci in plen

V danem področju imamo množico plenilcev (na primer lisic) in množico plenov (na primer zajčkov).

Če ni plenilcev, se zajčki preveč razmnožijo. Če ni plena, plenilci stradajo in izumrejo.

Ob dani začetni populaciji uporabimo diferenčne enačbe za simulacijo razvoja obeh populacij. Na primer :

$L_{n+1} = L_n + (c*Z_n - d)* L_n;$

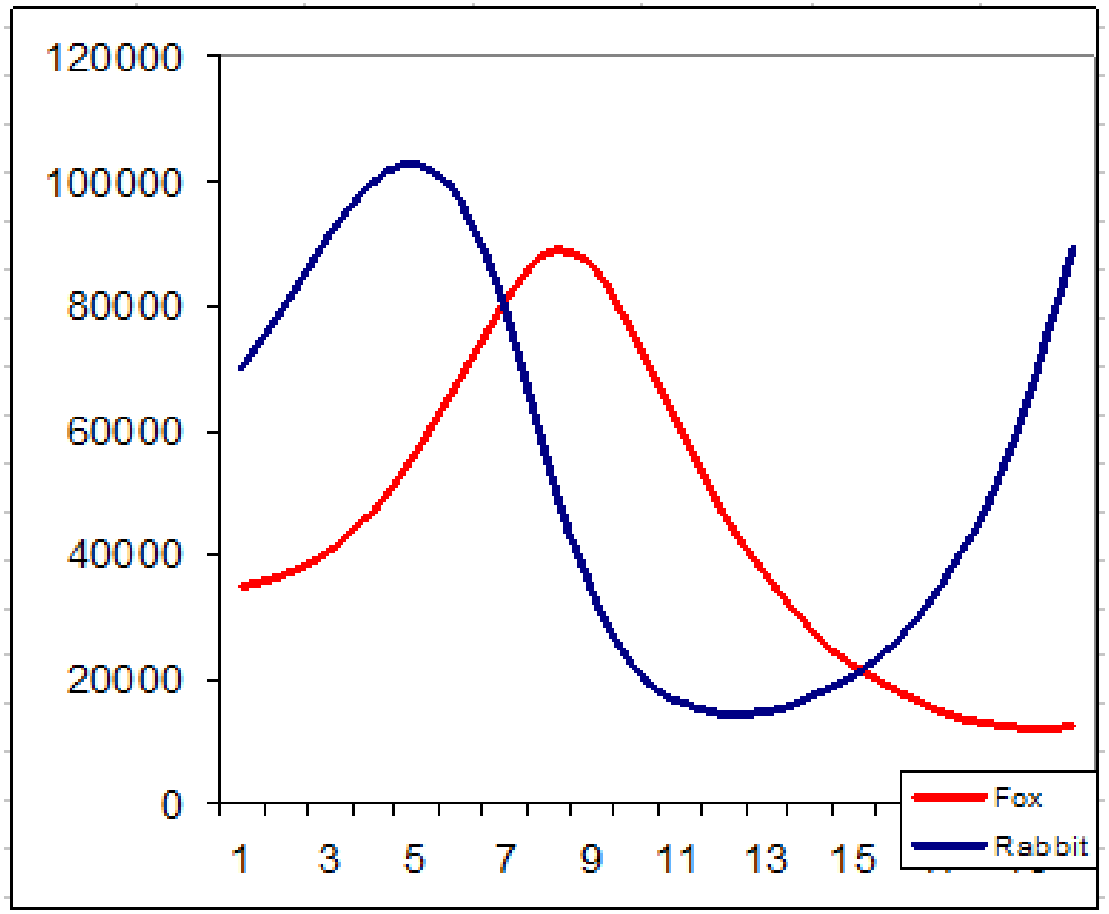
$Z_{n+1} = Z_n + (f - g* L_n)*Z_n;$

L so lisice, Z so zajci.
c, d,f, g so parametri

.

Nastavi začetne vrednosti populacij na Z_0 in L_0

Populacija Lisic in zajcev

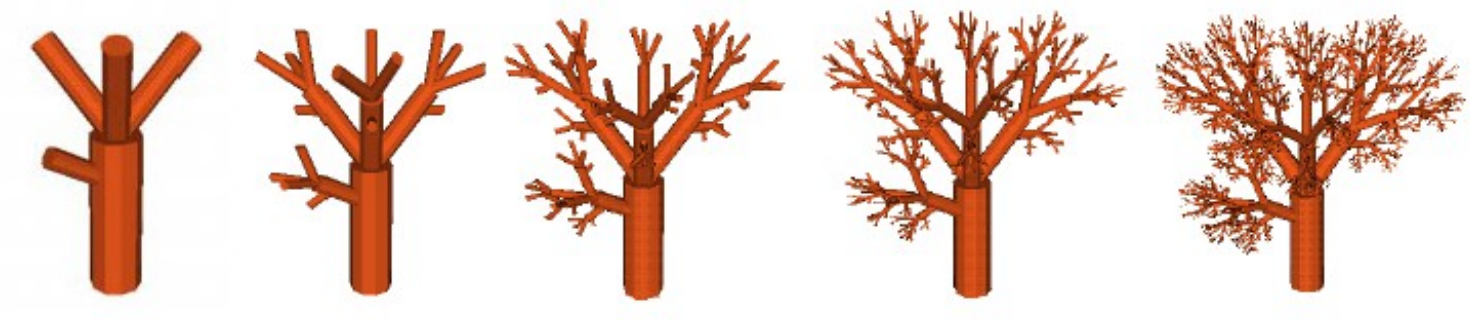


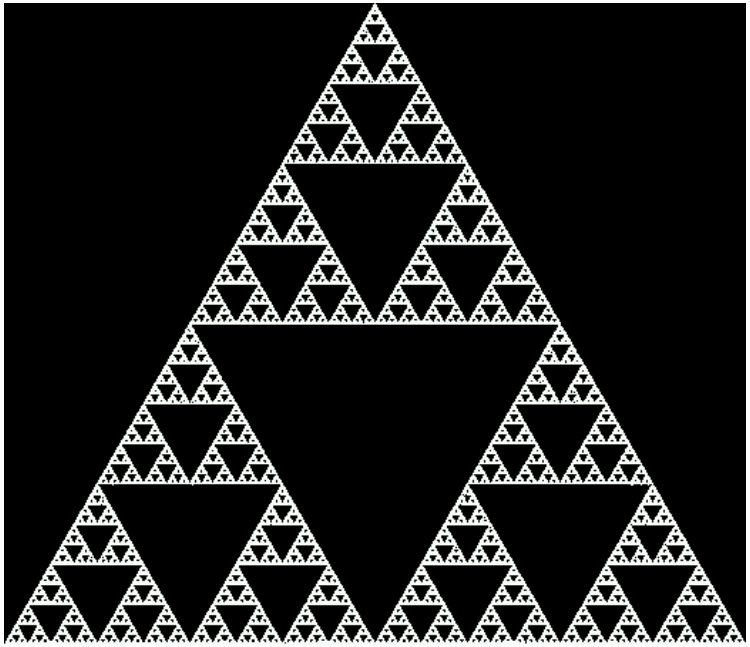
Lisice in zajci : koda

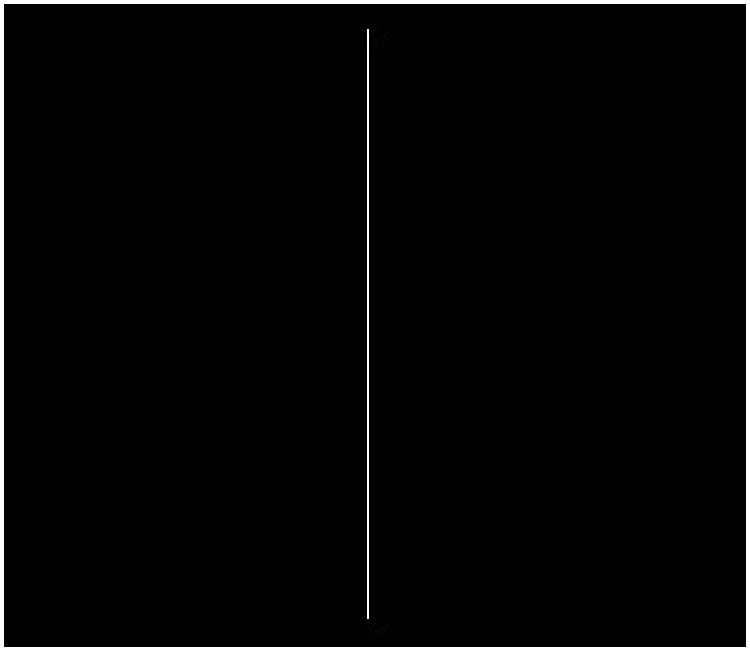
```
zajcevPrej = 100;  lisicPrej = 100; // zacetna populacija
// c,d,f,g  so parametri, ki jih moramo nastaviti

for (time = tMin; time <= tMax; time += stepSize) {
     lisic = lisicPrej + (c*zajcevPrej - d)* lisicPrej;
     zajcev = zajcevPrej + (f- g* lisicPrej) * zajcevPrej;

     // zajcev oziroma lisic ne more biti manj kot nek minimum
      if (.lisic<10) lisic = 10;
       if (zajcev<10) zajcev=10;

       // priprava na naslednjo iteracijo
       zajcevPrej = zajcev;
       lisicPrej = lisic;
}
```

Demo

Primeri rekurzije







Rekurzija

Do rekurzije pride, če metoda kliče samo sebe, direktno ali indirektno preko verige klicev drugih metod.

Metoda, ki kliče samo sebe, je rekurzivna metoda.

Primer: Izpis niza

```
void print(String str, int index){
   if (index < str.length()){
      System.out.print(str.charAt(index));
      print(str, index + 1);
   }
}
```

```
print("Hello", 0);
   print("Hello", 1);
      print("Hello", 2);
         print("Hello", 3);
            print("Hello", 4);
               print("Hello", 5);
               Return
            Return
         Return
      Return
   Return
Return
```

Hello

Primer s potenco

Definicija potence je

$$X^{y} = 1 * X * X * \ldots * X \text{ (y krat)}$$

Zamislimo si, da bi to morali računati ročno:

$X = 1 * X * X * \ldots * X$ (y krat) Kaj, če je y velik?

Leni pristop:

Imejmo asistenta, ki bo izračunal X^{y-1}

in rezultat je $= X * X^{y-1}$

Imejmo asistenta, ki bo izračunal X^{y-2}

Imejmo asistenta, ki bo izračunal X^{y-2}

.....

do X^{0}

Rekurzivno računanje potence

Postopek naj bo naslednji:

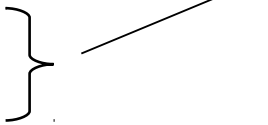
Za izračun potence x na y

- Če je y enak 0, ni kaj množiti, rezultat je 1.

- Če je y večji od 0, tedaj
  - Prosimo asistenta, da izračuna potenco ***x na y-1*.**
  - Rezultat je x krat rezultat asistenta.

Rekurzivna metoda

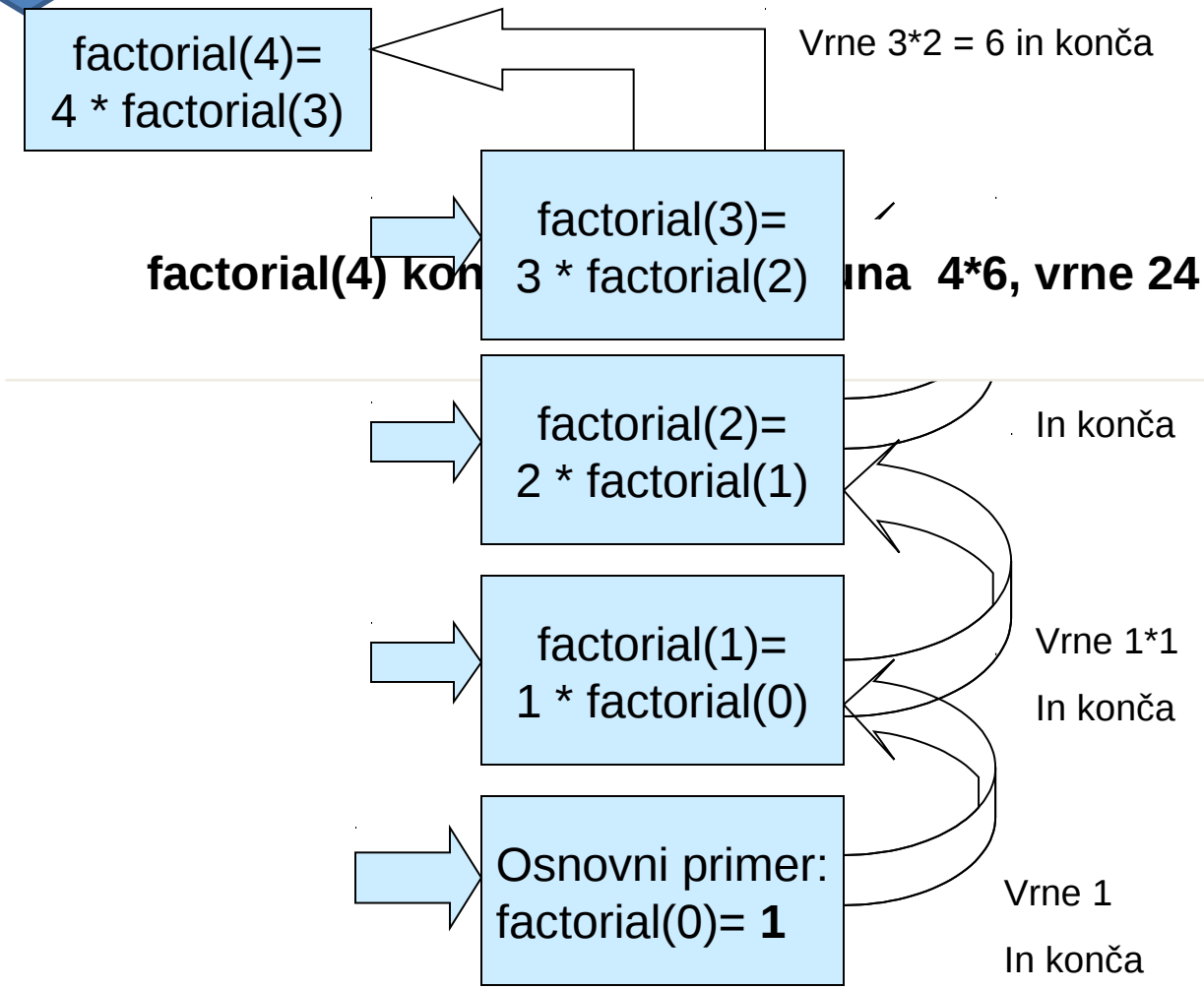
```
private  int  potenca(int x, int y) {      // y>=0, returns x**y

    int    rezultatAsistenta;

    if  (y==0)
            return 1;

    else  {
            rezultatAsistenta = potenca(x,y-1);
            return x*rezultatAsistenta;
    }
}
```



Korak prekinitve mora biti pred rekurzivnim klicem
Preveri, če lahko postopek zaključimo brez rekurzije.

Rekurzivni klic
Argumenti v rekurzivnem klicu morajo določati nalogo,
ki je lažja ali manjša od naloge,
ki jo je zahteval klicatelj.

Sledenje klica : int z = factorial(4)



Lastnosti rekurzivnih metod

Vse rekurzivne metode imajo naslednje lastnosti:

- Enega ali več osnovnih primerov (najbolj enostavni primer), ki ga uporabimo za prekinitev rekurzije.

- Vsak rekurzivni klic mora originalni problem poenostaviti tako, da bo postopoma bližje osnovnemu primeru, dokler se s tem osnovnim primerom ne izenači.

Kako načrtujemo rekurzivne metode

Korak 1. Bodimo leni.

- Uporabimo rešitev manjše ali lažje verzije problema in z njeno pomočjo rešimo naš problem.
- Ta korak vodi v rekurzivni klic.

Korak 2. Vedeti, kdaj moramo nehati.

- Vedeti moramo, kdaj je problem tako enostaven, da ga lahko rešimo direktno.
- Ta korak vodi v kodo prekinitve.

Primer: innverzija niza

Funkcija vrne niz, v katerem so znaki pomnjeni v obratnem vrstnem redu.

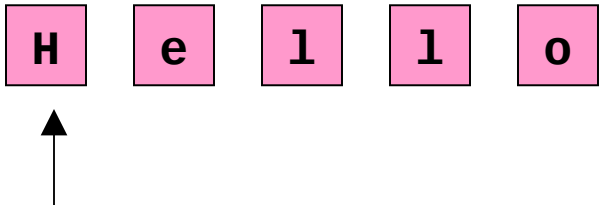
If there are no more characters to examine
** Return the empty string**
Else
** Reverse the rest of the string after the current char**
** Return the concatenation of this result**
**   and the char at the current position**

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

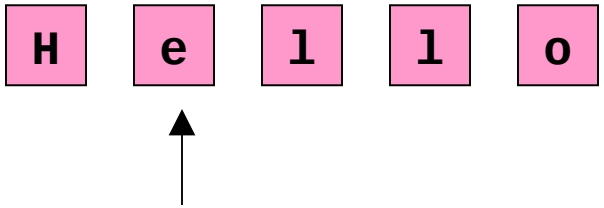
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
            Return ""
          Return "o"
        Return "ol"
      Return "oll"
    Return "olle"
  Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

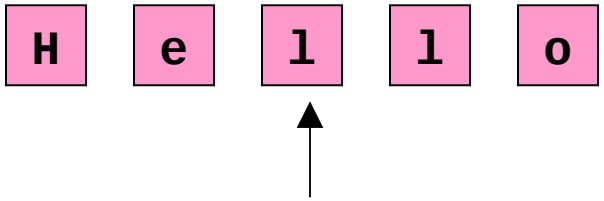
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
            Return ""
          Return "o"
        Return "ol"
      Return "oll"
    Return "olle"
  Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

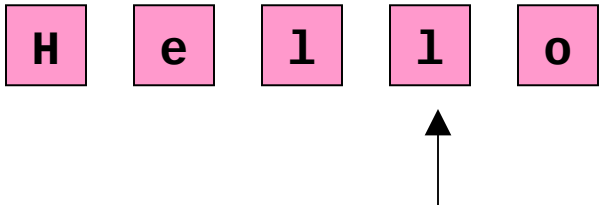
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
            Return ""
          Return "o"
        Return "ol"
      Return "oll"
    Return "olle"
  Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

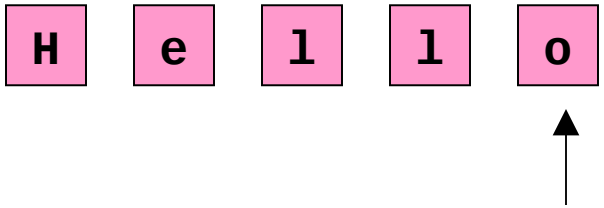
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
            Return ""
          Return "o"
        Return "ol"
      Return "oll"
    Return "olle"
  Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

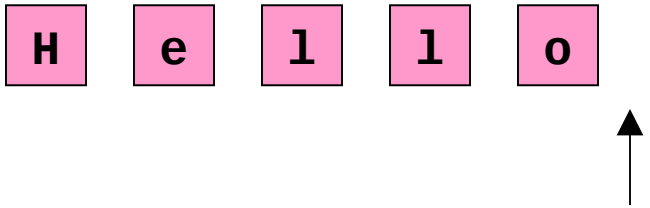
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
            Return ""
          Return "o"
        Return "ol"
      Return "oll"
    Return "olle"
  Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

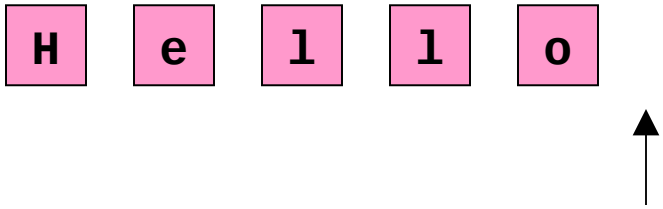
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
            Return ""
          Return "o"
        Return "ol"
      Return "oll"
    Return "olle"
  Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

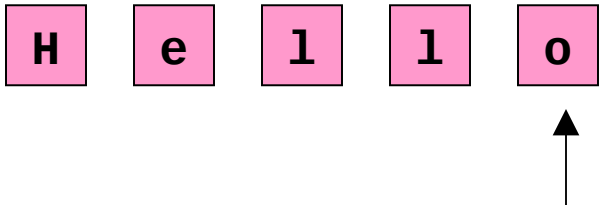
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
          Return ""
        Return "o"
      Return "ol"
    Return "oll"
  Return "olle"
Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

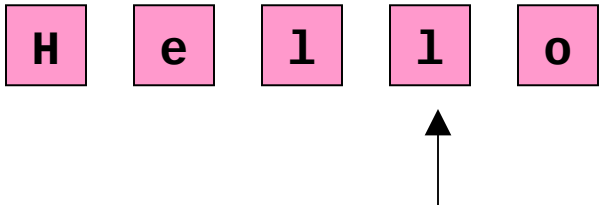
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
          Return ""
        Return "o"
      Return "ol"
    Return "oll"
  Return "olle"
Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

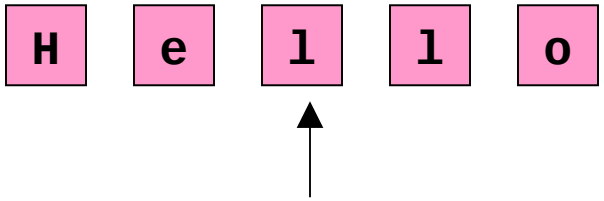
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
          Return ""
        Return "o"
      Return "ol"
    Return "oll"
  Return "olle"
Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

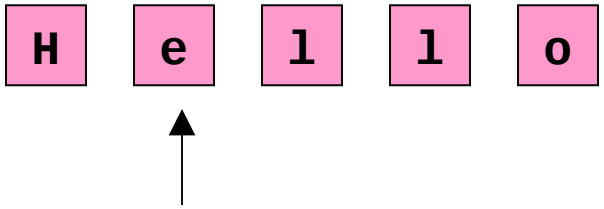
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
          Return ""
        Return "o"
      Return "ol"
    Return "oll"
  Return "olle"
Return "olleH"
```



Inverzija niza

```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

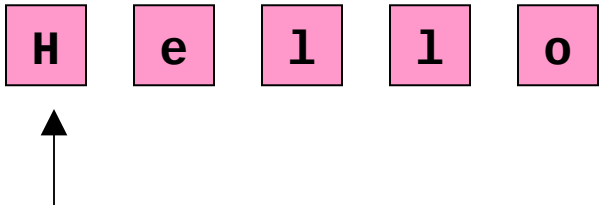
```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
          Return ""
        Return "o"
      Return "ol"
    Return "oll"
  Return "olle"
Return "olleH"
```



Inverzija niza

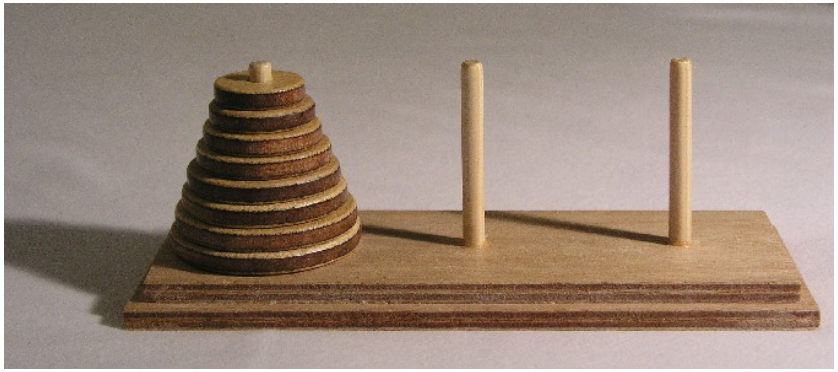
```
String reverse(String str, int index){
   if (index == str.length())
      return "";
   else{
      String rest = reverse(str, index + 1);
      return rest + str.charAt(index);
   }
}
```

```
reverse("Hello", 0);
  reverse("Hello", 1);
    reverse("Hello", 2);
      reverse("Hello", 3);
        reverse("Hello", 4);
          reverse("Hello", 5);
          Return ""
        Return "o"
      Return "ol"
    Return "oll"
  Return "olle"
Return "olleH"
```



Hanojski stolpiči

Problem s Hanojskimi stolpiči je klasičen rekurzivni problem, ki temelji na preprosti igri. Imamo tri palice. Na eni je sklad ploščic z različnimi premeri.



Cilj igre: Prestaviti vse ploščice na desno palico ob upoštevanju naslednjih pravil:

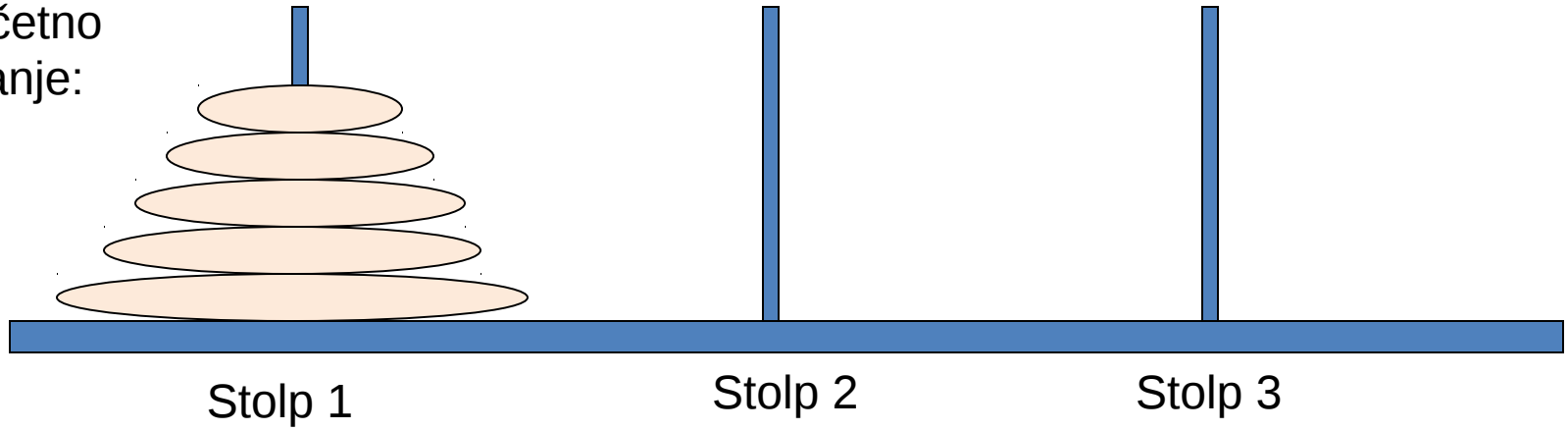
- Nobena ploščica ne sme biti nikoli na vrhu manjše ploščice.
- Naenkrat smemo premikati le po eno ploščico.
- Vsako ploščico moramo vedno odložiti na eno od palic, nikoli ob strani.
- Premaknemo lahko vedno le ploščico, ki je na vrhu nekega stolpiča.

Zgodba pravi, da bi za fizičen premik 64 ploščic iz ene palice na drugo potrebovali toliko časa, da bi prej bil konec sveta.

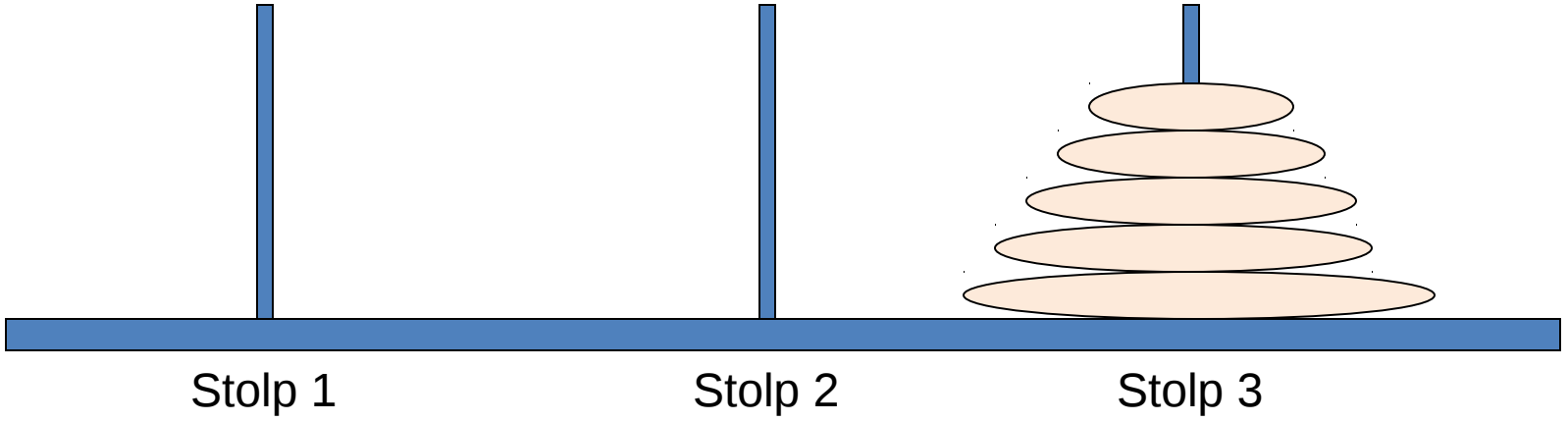
Demo

N ploščic s stolpa 1 na stolp 3

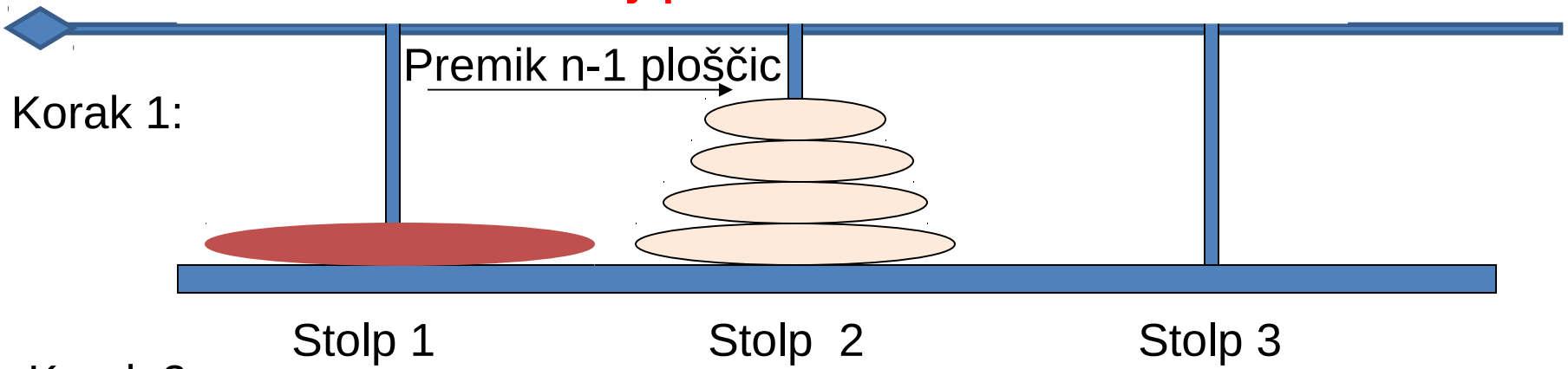
Začetno stanje:

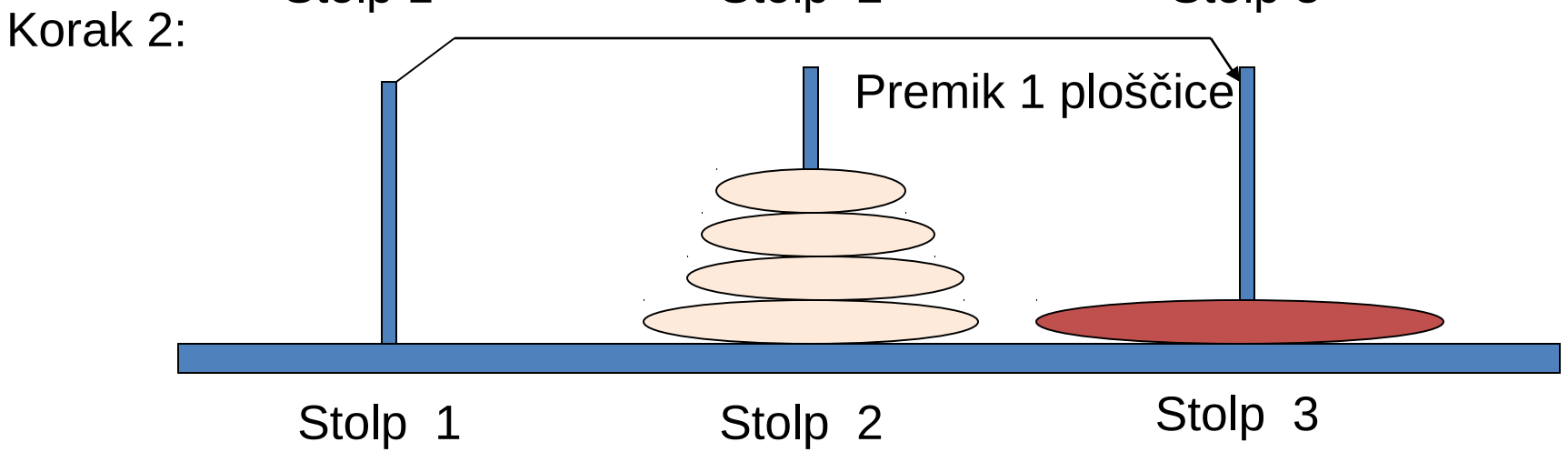


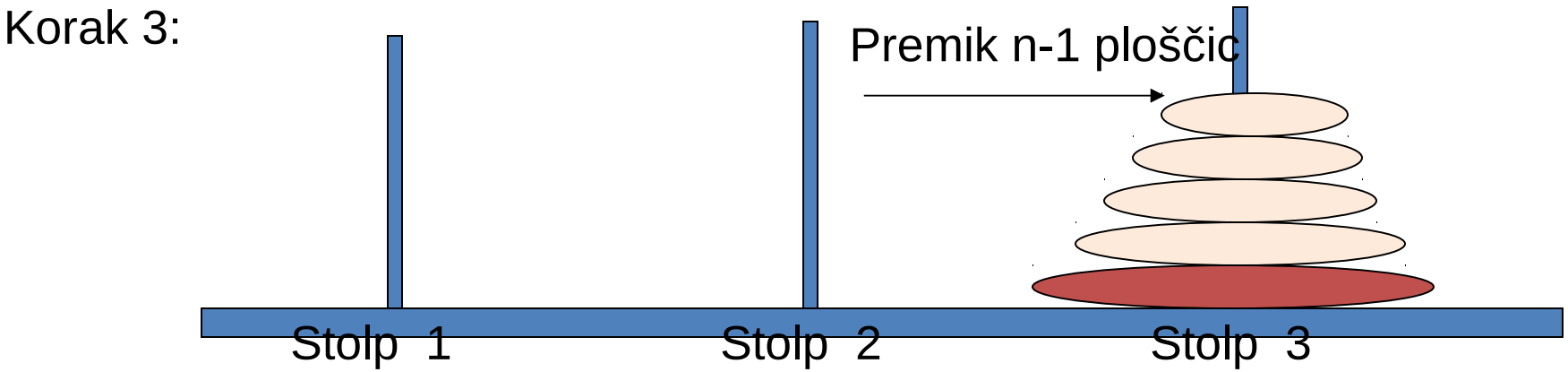
Cilj:



Nekaj prvih korakov







Koda v pascalu

```
procedure Hanoi(n: integer; from, to, by: char);
Begin
    if (n=1) then
        writeln('Move the plate from ', from, ' to ', to)
    else begin
        Hanoi(n-1, from, by, to);
        Hanoi(1, from, to, by);
        Hanoi(n-1, by, to, from);
    end;
End;
```

Koda v javi (bolj popolna)

```
public class TowersOfHanoi {

  public static void main(String[] args) {
    moveDisks(3, "Tower 1", "Tower 3", "Tower 2");
  }

  public static void moveDisks(int n, String fromTower, String toTower, String auxTower)
  {
      if (n == 1)
          System.out.println("Move disk " + n + " from " + fromTower + " to " + toTower);
      else
      {
              moveDisks(n-1, fromTower, auxTower, toTower);


              System.out.println("Move disk " + n + " from " + fromTower + " to " + toTower);

              moveDisks(n-1, auxTower, toTower, fromTower);
      }
  }
}
```

Move n-1 disks from **from** to **temp**, using **to** as a temporary.

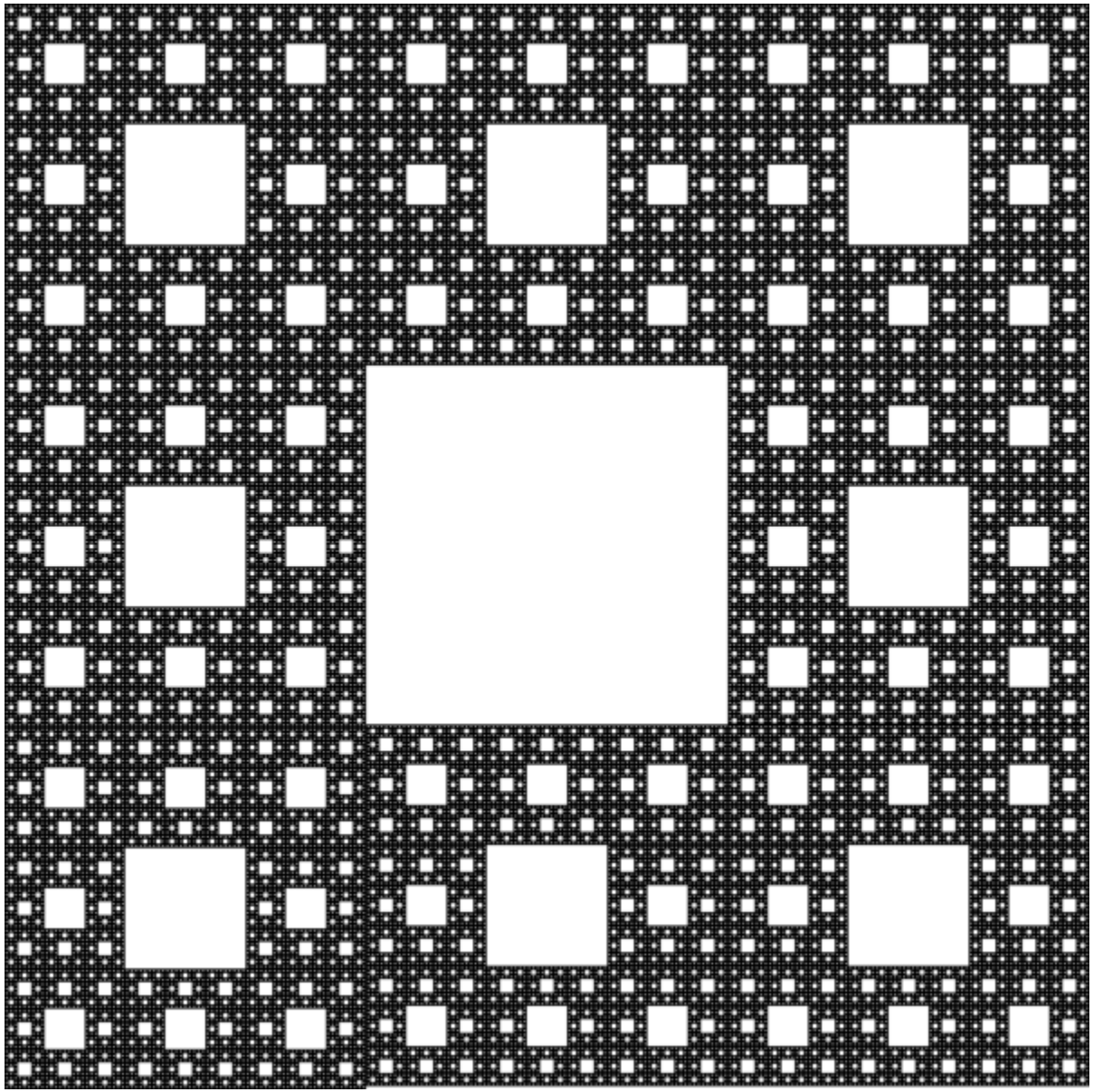
Move one disk from **from** to **to**.

Move n-1 disks from **temp** to **to**, using **from** as a temporary.

Kakšen bi bil izpis

Move disk 1 from Tower 1 to Tower 3
Move disk 2 from Tower 1 to Tower 2
Move disk 1 from Tower 3 to Tower 2
Move disk 3 from Tower 1 to Tower 3
Move disk 1 from Tower 2 to Tower 1
Move disk 2 from Tower 2 to Tower 3
Move disk 1 from Tower 1 to Tower 3

Primer s fraktali



Demo

Preproga Sierpinski

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class SierpinskiCarpet extends Applet {

    int level;
    public void init() {
        repaint();
    }

    public synchronized void paint(Graphics g){
        int i = getSize().width;
        int j = getSize().height;
        g.setColor(Color.white);
        g.fillRect(0, 0, i, j);
        level = 5;
        carpet(g, level, 0, 0, i, j);
    }
```

Preproga Sierpinski (nadaljevanje)

```
public void carpet(Graphics g, int i, int j, int k, int l, int i1){
        if(i == 0) {
            g.setColor(Color.black);   g.fillRect(j, k, l, i1);   return;
        }
        else {
            int j1 = l / 3;
            int k1 = i1 / 3;
            carpet(g, i - 1, j, k, j1, k1);
            carpet(g, i - 1, j + j1, k, j1, k1);
            carpet(g, i - 1, j + 2 * j1, k, j1, k1);
            carpet(g, i - 1, j, k + k1, j1, k1);
            carpet(g, i - 1, j + 2 * j1, k + k1, j1, k1);
            carpet(g, i - 1, j, k + 2 * k1, j1, k1);
            carpet(g, i - 1, j + j1, k + 2 * k1, j1, k1);
            carpet(g, i - 1, j + 2 * j1, k + 2 * k1, j1, k1);
            return;
        }
    }
}
```

Primerjava iteracija : rekurzija (1)

- **Iteracija**
  - Uporabljamo strukture ponavljanja (`for`, `while` ali `do…while`)
  - Ponavljanje skozi eksplicitno uporabo struktur ponavljanja
  - Konča se, ko pogoj za ponavljanje zanke ne velja več
  - (običajno) ponavljanja krmilimo s števcem
- **Rekurzija**
  - Uporabljamo strukture izbiranja(`if`, `if…else` ali `switch`)
  - Ponavljanje skozi ponovne klice metod oziroma funkcij
  - Konča se, ko je izpolnjen osnovni primer
  - Ponavljanje krmilimo z deljenjem problema na bolj enostavnega

Primerjava iteracija : rekurzija (1)

- **Še o rekurziji**
  - Terja več režije kot iteracija
  - Je pomnilniško bolj zahtevna od iteracije
  - Rekurzivne probleme lahko rešujemo tudi z iteracijami
  - Pogosto nam za rekurzijo zadošča le nekaj vrstic kode

Malo za sprostitev

Podano je nekaj zaporedij. Ugotoviti moramo naslednje člene zašporedij. Vsak od navedenih problemov ima dva odgovora: eden je jasen, drugi pa bolj skrit.

Problem A: 3, 1, 4, 1, 5, … Kateri je naslednji člen zaporedja?

Problem B: 2, 3, 5, 8, … Kateri je naslednji člen zaporedja?

Problem C: 2, 7, 1, 8, 2, 8, 1, 8, … Katera sta dva naslednja člena zaporedja?

Odgovori A: preprosto = 1; skrito = 9;
Odgovori B: preprosto = 12; skrito = 13;
Odgovori C: preprosto =(2, 9); skrito = (2, 8);

Nedeterministični konstrukti

Nedeterminstični konstrukti namenoma ne specificirajo izbire med alternativami.

•Ta mehanizem je posebno uporaben v konkurenčnih programih