ITERACIJA IN REKURZIJA

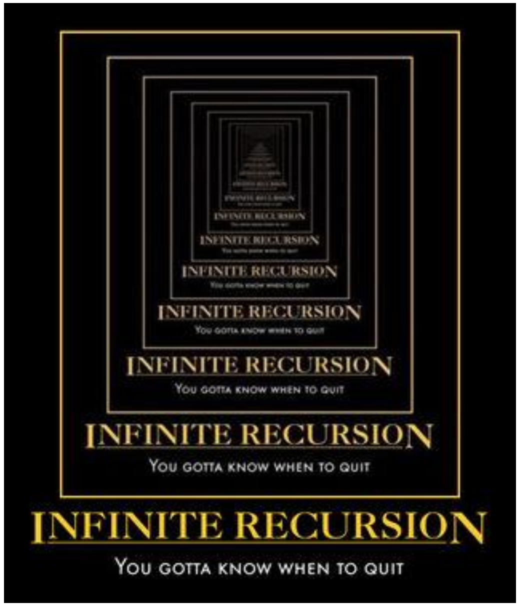
sta krmilna mehanizma, ki skrbita za PONAVLJANJE izvajanja

ITERACIJA: zanke

```
for (int i =0;i<=10;i++){
  ...
}
```

```
int i = 0;
while (i<=10){
  ...
  i++;
}
```

REKURZIJA: program kliče samega sebe, dokler ni izpolnjen ustavitveni pogoj



Iteracija in rekurzija

Definicija:

Nov pojem = ... znani pojmi ...

Rekurzivna definicija:

Nov pojem = ... znani pojmi + Nov pojem ...

rekurzija – *glej* rekurzija

rekurzija – *glej* rekurzivna definicija

rekurzivna definicija – *glej* rekurzija

Iteracija in rekurzija

Rekurzivna definicija:

1) Rekurzijska spremenljivka (n)

2) Robni pogoj (n = 0)
Nov_pojem(0) = znani pojmi

3) Splošni primer (n > 0)
Nov_pojem(n) = znani pojmi + Nov_pojem (n-1)

Primer: izračun fakultete

fakulteta(5) = 5! = 5*4*3*2*1 = 120

fakulteta(n) = n! = n*(n-1)*(n-2)*(n-3)* … * 2*1

Iterativno:

```
    fakulteta = 1;
    for(int i=1; i <= n; i++)
        fakulteta = fakulteta*i;
```

Rekurzivno:

- Kaj je rekurzijska spremenljivka?
- Kaj mi pomaga, če imam rešen manjši problem?
- Kakšen bo robni primer?
- Kakšen bo splošni primer?

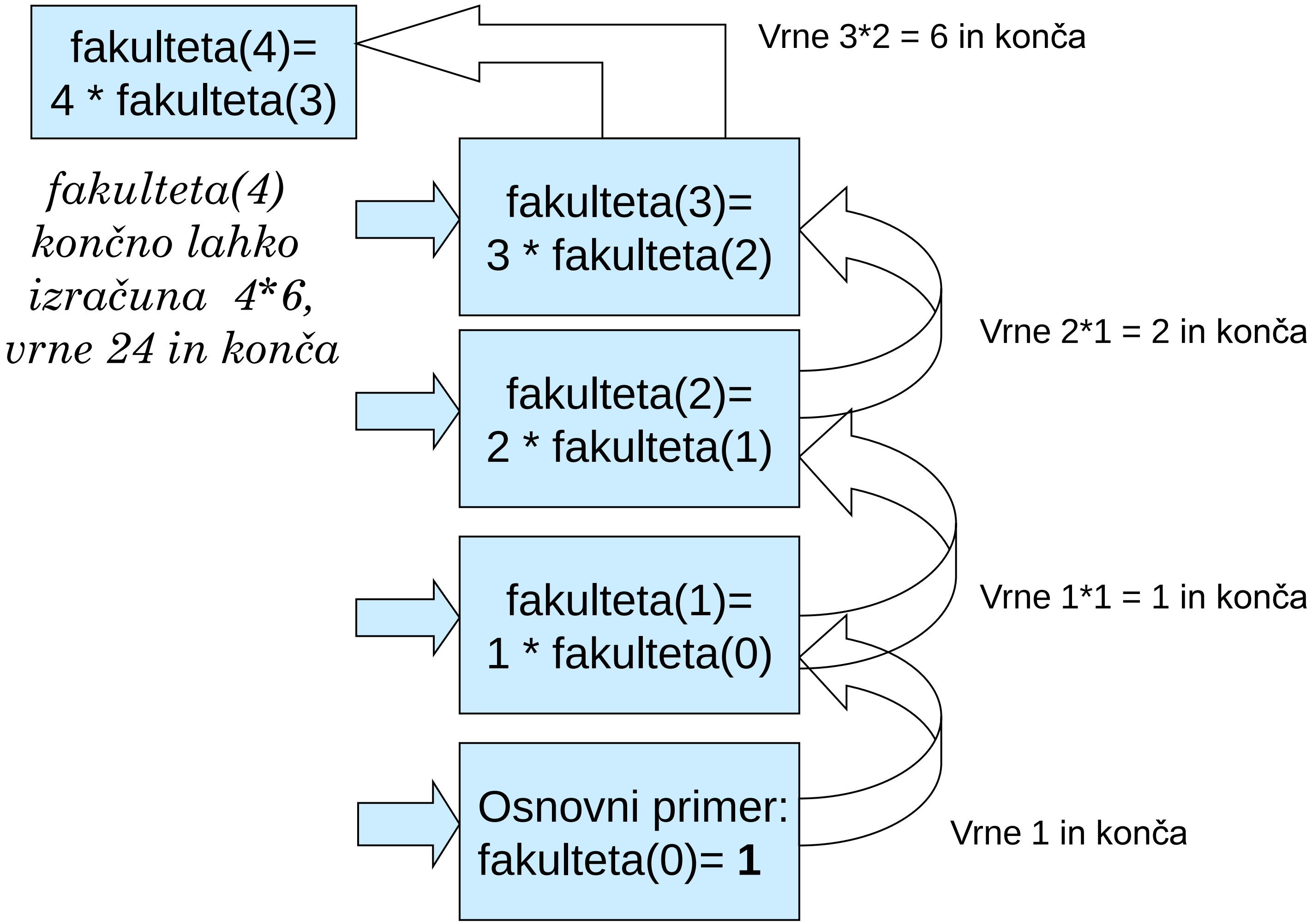
Primer: izračun fakultete

fakulteta(n) = n! = n*(n-1)*(n-2)*(n-3)* … * 2*1

1) Kaj je rekurzijska spremenljivka? n
2) Če imam izračunan (n-1)! lahko izračunam n!
3) Robni pogoj: (n = 0)
   0! = 1
3) Splošni primer (n > 0)
   n! = n * (n-1)!

```
private int fakulteta(int n) {
if (n==0) return 1;
    else  { return n * fakulteta(n-1); }
}
```

PRIMER: IZRAČUN FAKULTETE



Primeri rekurzivnih programov

1. Fibonacci
2. Maksimalno število
3. Potenciranje števila
4. Višina binarnega drevesa
5. Izračuna izraza
6. Hanojski stolpi

Primer Fibonacci

Fibonaccijeva števila tvorijo naslednje zaporedje

n	1,	2,	3,	4,	5,	6,	7,	8,	9,	10
F	1,	1,	2,	3,	5,	8,	13,	21,	34,	55

Rekurzivno

fib(1) = 1

fib(2) = 1

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

```
public static int fib(int n) {
     if (n <= 2)
          return 1;
     else
          return fib(n-1)+fib(n-2);
}
```

Primer Fibonacci

Rekurzivno

fib(1) = 1

fib(2) = 1

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

```
public static int fib(int n) {
    if (n <= 2)
        return 1;
    else
        return fib(n-1)+fib(n-2);
}
```

robni pogoj

splošni primer

PRIMERI REKURZIVNIH PROGRAMOV

1. Fibonacci
2. Maksimalno število
3. Potenciranje števila
4. Višina binarnega drevesa
5. Izračun izraza
6. Hanojski stolpi

Maksimalno število

1	2	3	4	5	6	7	8	…	n
17	25	13	66	12	4	45	9	…	65

Poiskati je treba maksimalno izmed n števil.

1) Kaj je rekurzijska spremenljivka? n

2) Če imam izračunan max(n-1):

vrnem večjega izmed max(n-1) in element[n]

3) Robni pogoj: (n = 1)

max(1) = element[1]

3) Splošni primer (n > 0)

max(n) = maximum(max(n-1), element[n])

PRIMERI REKURZIVNIH PROGRAMOV

1. Fibonacci
2. Maksimalno število
3. Potenciranje števila
4. Višina binarnega drevesa
5. Izračun izraza
6. Hanojski stolpi

POTENCIRANJE ŠTEVILA

Naloga: Izračunati želimo število X^p

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

Kaj je rekurzijska spremenljivka? p

Splošni primer

Kaj nam pomaga, če imamo izračunan X^{p-1}? Rezultat je $X^p = X * X^{p-1}$

Drugi rek. klic bo izračunal X^{p-2} in veljalo bo $X^{p-1} = X * X^{p-2}$

Tretji rek. klic bo izračunal X^{p-3} ......

.....

robni pogoj $X^0 = 1$!!!

POTENCIRANJE ŠTEVILA

// int potenca(int x, int p)

potenca(x, 0) = 1

potenca(x, p) = x * potenca(x, p-1)   za p > 0

```
private int potenca(int x, int p) {   // za p >= 0 vrne x^p
if (p==0)
        return 1;
    else  {
        return x * potenca(x, p-1);
    }
}
```

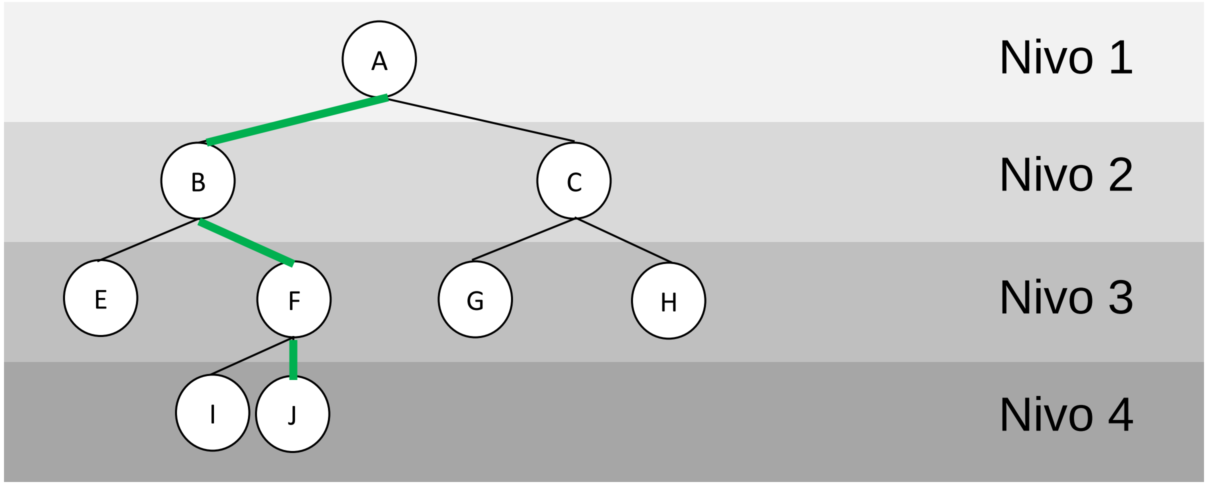
robni pogoj

splošni primer

PRIMERI REKURZIVNIH PROGRAMOV

1. Fibonacci
2. Maksimalno število
3. Potenciranje števila
4. Višina binarnega drevesa
5. Izračun izraza
6. Hanojski stolpi

VIŠINA BINARNEGA DREVESA



pot (*path*): zaporedje vozlišč, ki so v relaciji oče-sin (npr. A, B, F, J)

nivo (*level*) vozlišča: dolžina poti od korena do vozlišča (npr. nivo vozlišča E je 3)

višina (*height*) drevesa: dolžina najdaljše poti od korena do lista (npr. za zgornje drevo je 4)

VIŠINA BINARNEGA DREVESA

Rekurzivna definicija binarnega drevesa:

1. Prazno binarno drevo je brez vozlišč.
2. Binarno drevo ima koren in dve binarni poddrevesi.

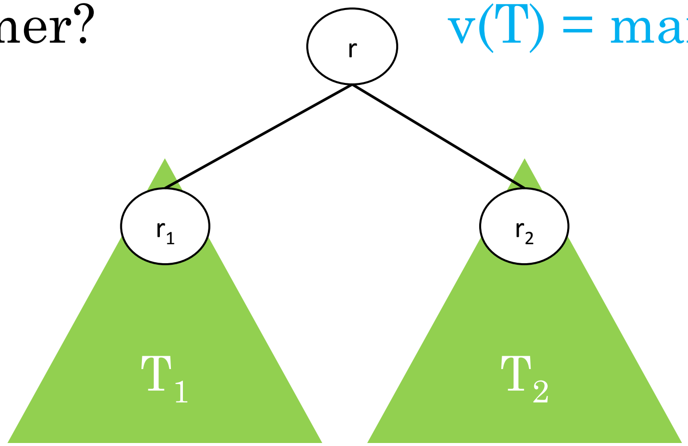
Kaj je rekurzijska spremenljivka? (T = koren drevesa)

Kaj mi pomaga rešitev manjšega problema?

Iz višin obeh poddreves lahko izračunam višino drevesa.

Kaj je robni pogoj? (T = null, v(null) = 0)

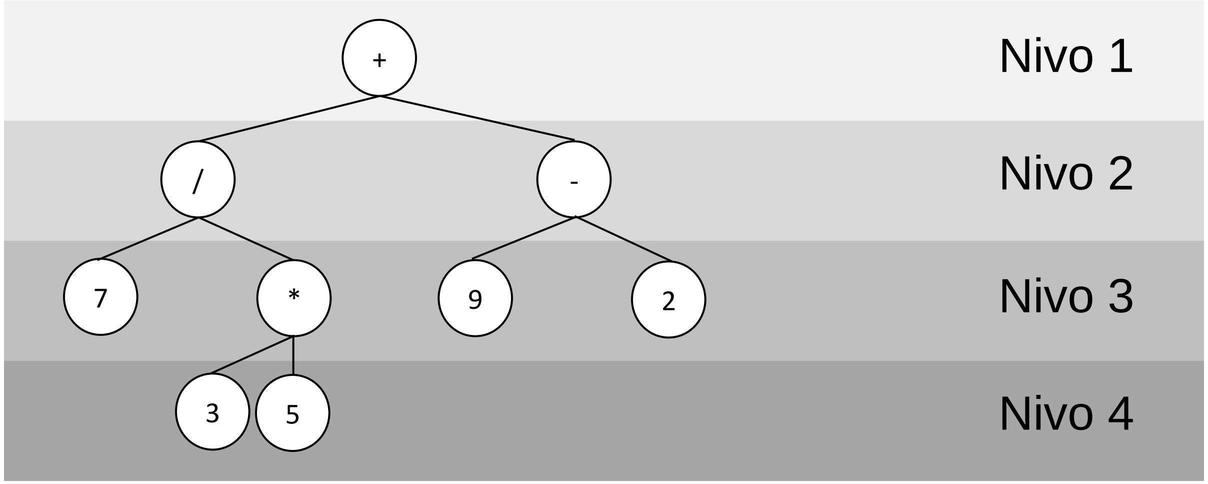
Kaj je splošni primer? v(T) = max(v(T1),v(T2)) + 1



PRIMERI REKURZIVNIH PROGRAMOV

1. Fibonacci
2. Maksimalno število
3. Potenciranje števila
4. Višina binarnega drevesa
5. Izračun izraza
6. Hanojski stolpi

IZRAČUN IZRAZA



Izrazno drevo: v notranjih vozliščih operacije, v listih števila

7/(3*5)+(9-2)

Kaj je rekurzijska spremenljivka? (T = koren drevesa)

Kaj mi pomaga rešitev manjšega problema?

Iz vrednosti obeh poddreves lahko izračunam vrednost drevesa.

Kaj je robni pogoj? (T = Število, v(Število) = Število)

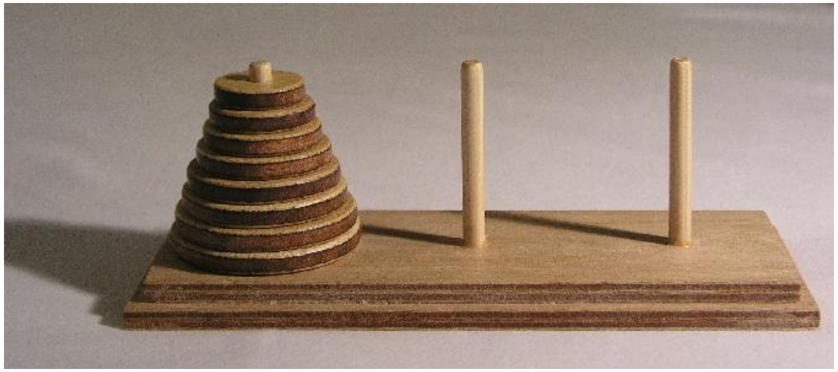
Kaj je splošni primer? v(operator) = v(T1) operator v(T2)

PRIMERI REKURZIVNIH PROGRAMOV

1. Fibonacci
2. Maksimalno število
3. Potenciranje števila
4. Višina binarnega drevesa
5. Izračun izraza
6. Hanojski stolpi

HANOJSKI STOLPI

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

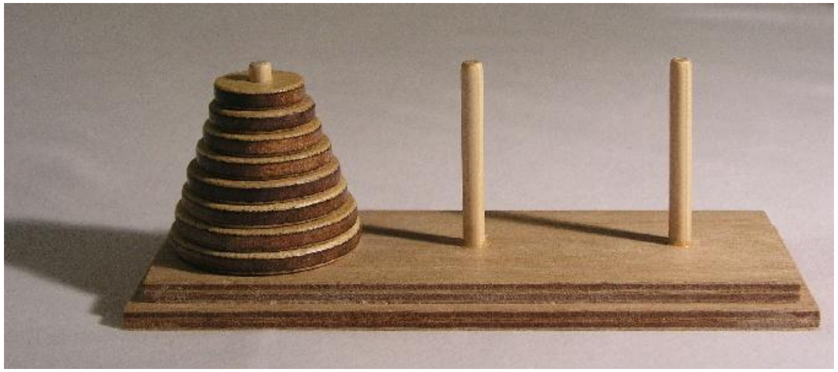


Cilj igre: Prestaviti vse ploščice na srednjo palico ob upoštevanju naslednjih pravil:

- 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.

HANOJSKI STOLPI



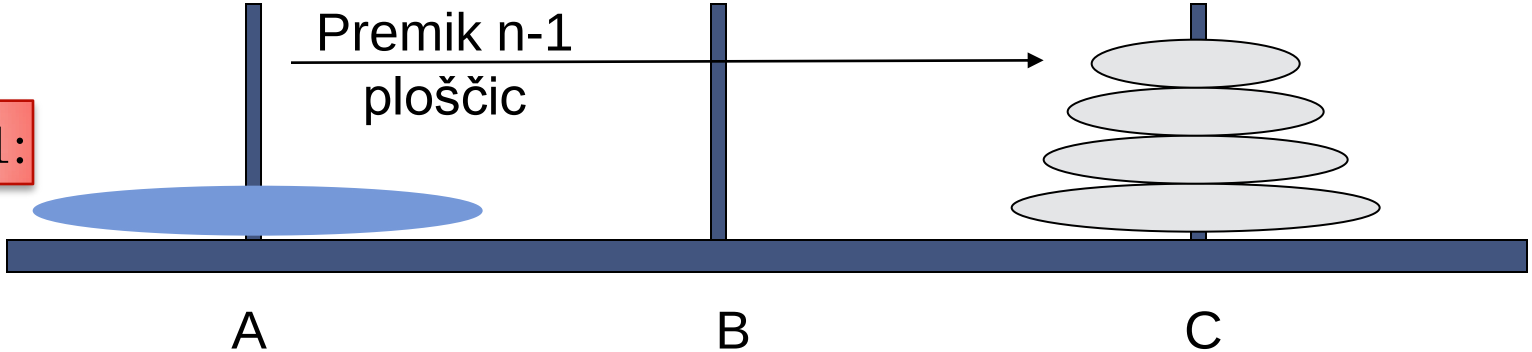
Poimenujem palice: a b c

Kaj je rekurzijska spremenljivka? (n = število ploščic na palici a)
Kaj mi pomaga rešitev manjšega problema?
Če znam premakniti n-1 ploščic iz a na c:
lahko največjo ploščico premaknem iz a na b (t.j. na CILJ).
Zatem moram premakniti samo še n-1 ploščic iz c na b.
Kaj je robni pogoj? (n=0)
Kaj je splošni primer? (n > 0)

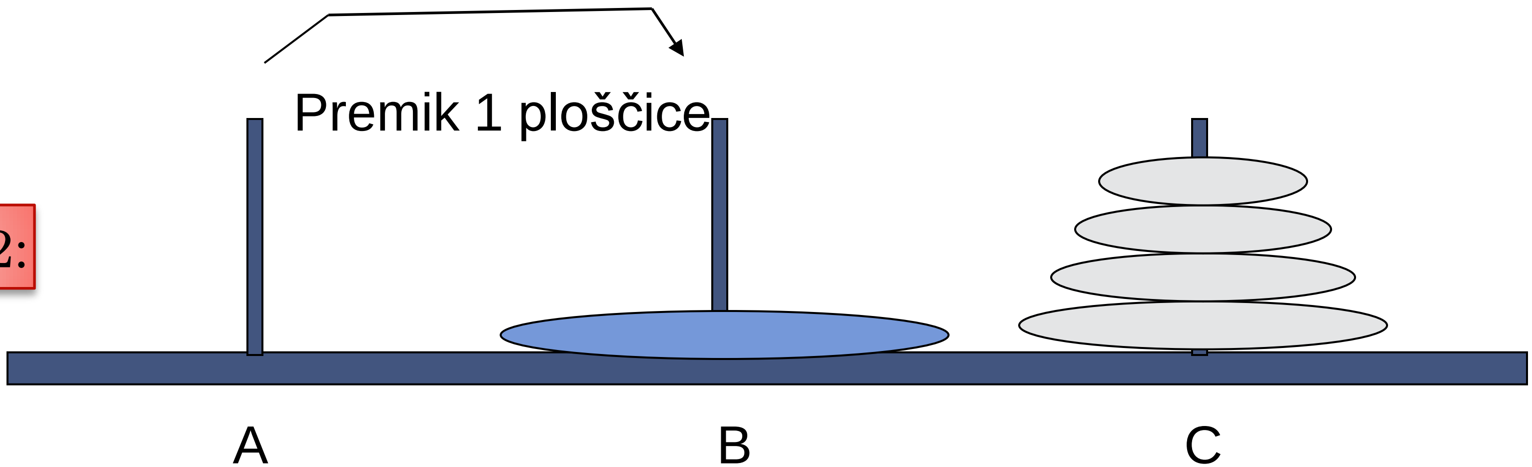
n: A → B
n-1: A → C
1: A → B
n-1: C → B

HANOJSKI STOLPI

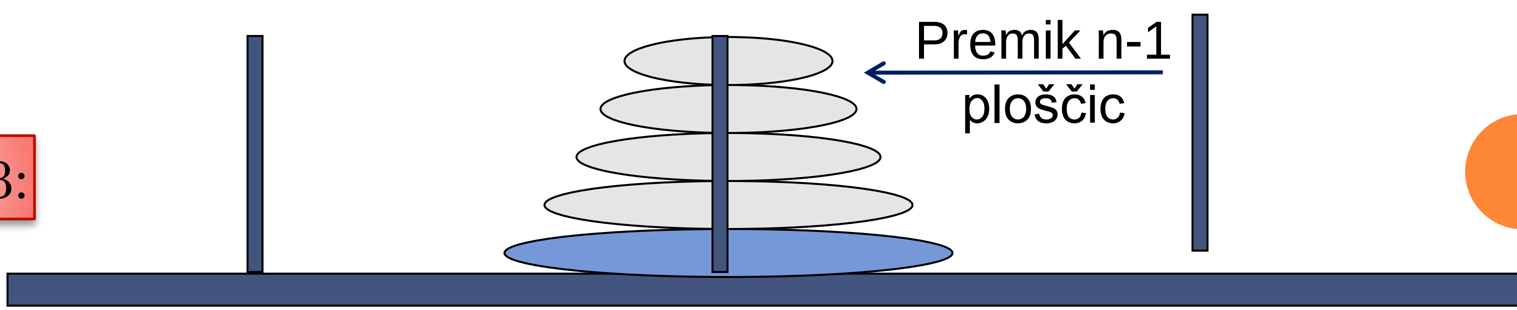
Korak 1:



Korak 2:



Korak 3:



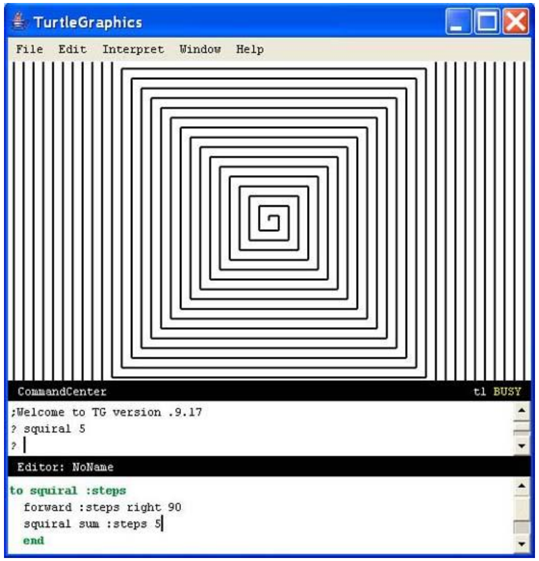
HANOJSKI STOLPI

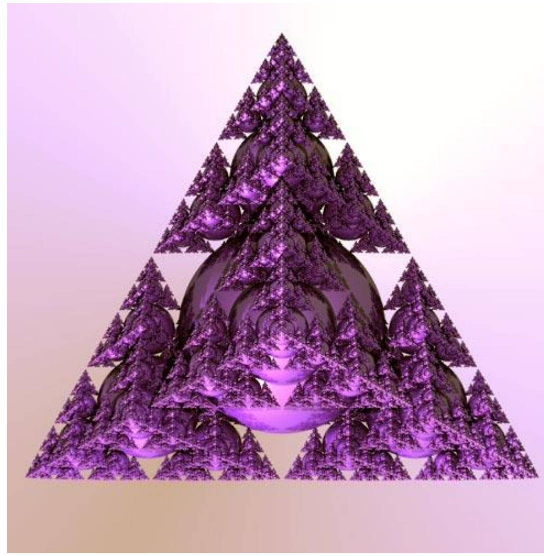
```
// premik n ploščic iz palice A na palico B z uporabo
// pomožne palice C
static public void hanoi(char A, char B, char C,int n) {
  if (n>0) {
    hanoi(A,C,B,n−1) ;
    System.out.println(”premik␣iz␣” + A + ”␣na␣” + B);
    hanoi(C,B,A,n−1) ;
  } // if
} // hanoi
```

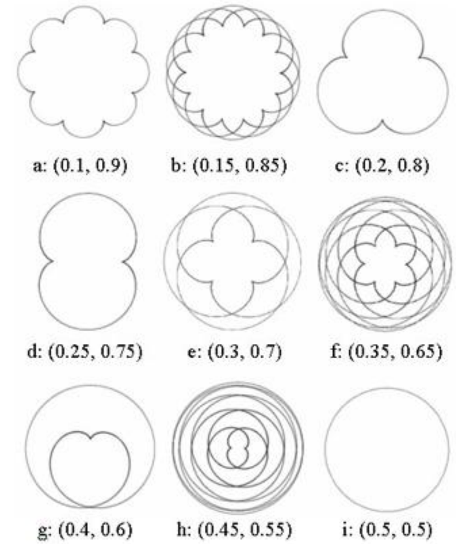
<u>hanoi('a','b','c',3)</u>

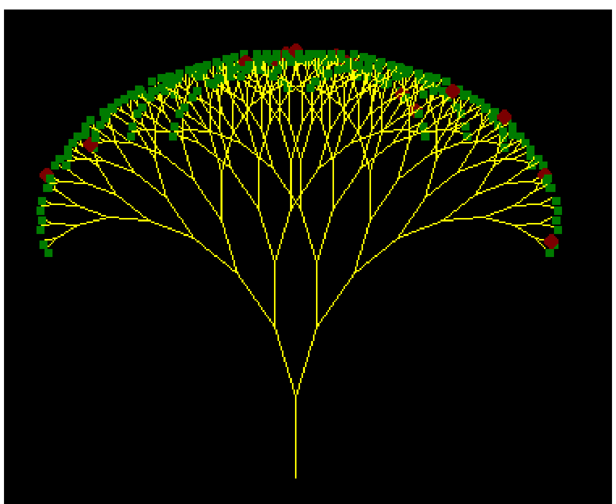
premik iz a na b
premik iz a na c
premik iz b na c
premik iz a na b
premik iz c na a
premik iz c na b
premik iz a na b

ŠE DRUGAČNE REKURZIJE…





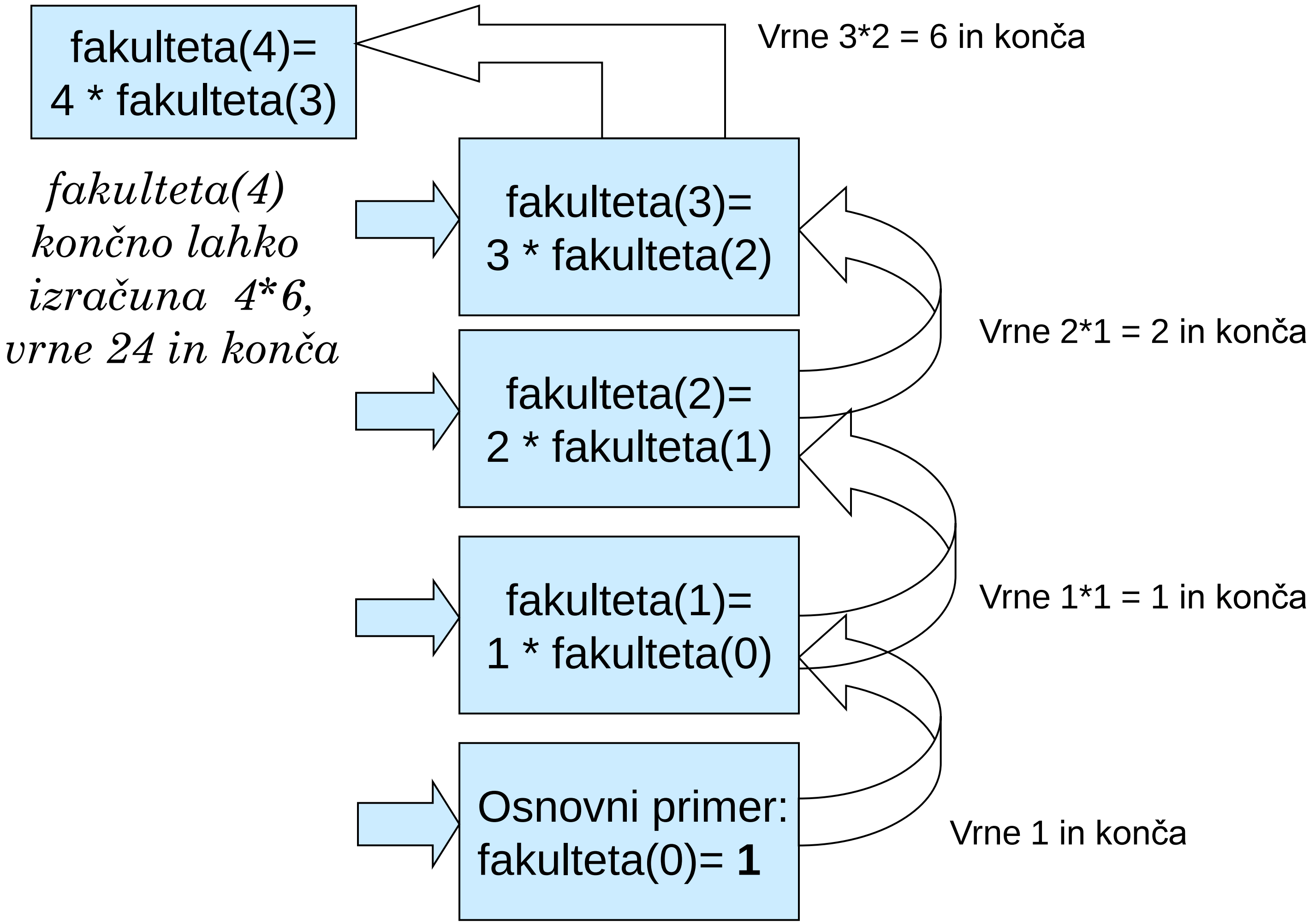




O REKURZIJI

Ali računalnik lahko izvaja rekurzivne programe?

Primer: izračun fakultete

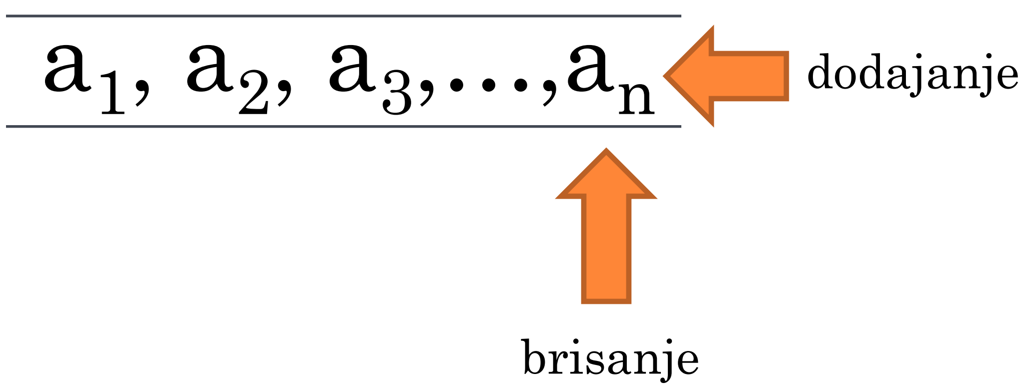


Sklad
(Stack)

ABSTRAKTNI PODATKOVNI TIP SKLAD

Sklad (stack) je zbirka elementov, kjer elemente vedno:

- dodajamo na vrh sklada
- brišemo z vrha sklada



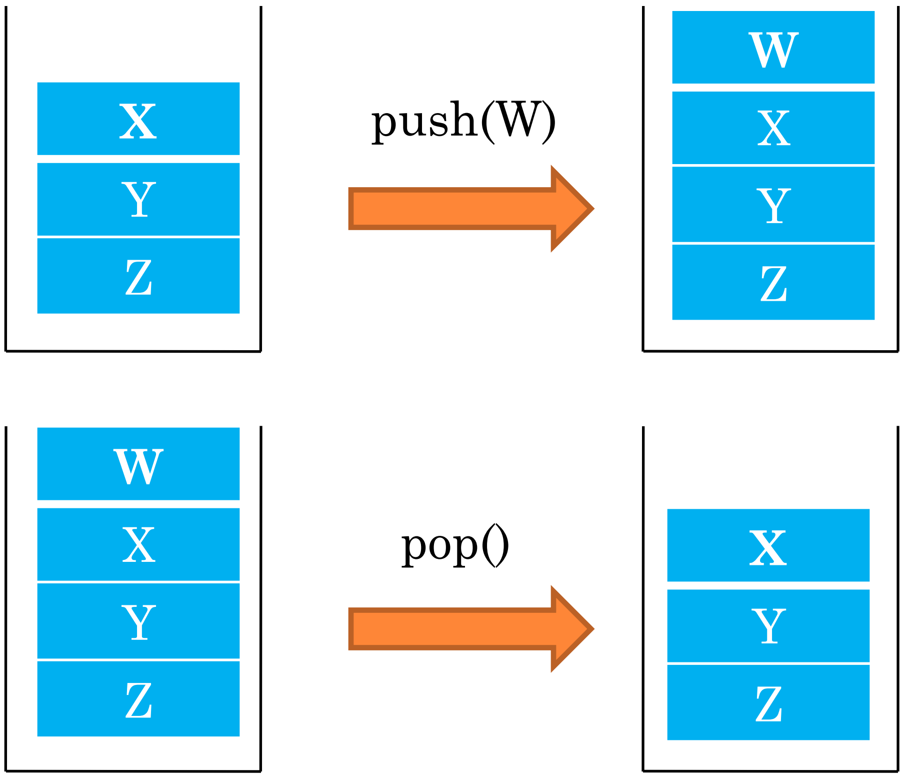
Skladu pravimo tudi LIFO (last-in-first-out).

ABSTRAKTNI PODATKOVNI TIP SKLAD

Operacije definirane za ADT STACK:

- MAKENULL(S) – naredi prazen sklad
- EMPTY(S) – ali je sklad prazen
- TOP(S) – vrne vrhnji element sklada
- PUSH(x, S)- vstavi element x na vrh sklada
- POP(S) – zbriše vrhnji element sklada

ABSTRAKTNI PODATKOVNI TIP SKLAD



HANOJSKI STOLPI: UPORABA SKLADA

n: A → C

n-1: A → B

1: A → C

n-1: B → C

		1: a→ c		1: c→ b	
	2: a→ b	2: a→ b	2: a→ b	2: a→ b	2: a→ b
3: a→c	3: a→ c	3: a→ c	3: a→ c	3: a→ c	3: a→ c

		1: b→ a		1: a→ c	
	2: b→ c	2: b→ c	2: b→ c	2: b→ c	2: b→ c
3: a→c	3: a→ c	3: a→ c	3: a→ c	3: a→ c	3: a→ c

3: a→c	

O REKURZIJI

Rekurzija...

- zahteva več režije kot iteracija in je pomnilniško bolj zahtevna od iteracije (sklici se shranjujejo na skladu),
- globina rekurzije = potrebna velikost sklada,
- ponavadi je rekurzivna koda krajša in preprostejša,
- rekurzivne probleme lahko rešujemo tudi z iteracijami:
  - *vsako repno rekurzijo lahko zamenjamo z iterativno zanko,*
  - *vsak rekurzivni program lahko spremenimo v iterativnega s skladom.*

Repna rekurzija

Rekurziji na koncu (repu) metode pravimo **repna rekurzija** (angl. *tail recursion*). Ko se izvajanje repne rekurzije konča, je s tem hkrati konec izvajanja procedure.

Repno rekurzijo lahko preprosto nadomestimo z iterativno zanko. Namesto rekurzivnega klica, ustrezno spremenimo vrednosti argumentov in poženemo celotno proceduro od začetka.

Repna rekurzija → iteracija

```
static public void hanoi(char A, char B, char C,int n) {
  if (n>0) {
    hanoi(A,C,B,n−1) ;
    System.out.println(”premik␣iz␣” + A + ”␣na␣” + B);
    hanoi(C,B,A,n−1) ;
  } // if
} // hanoi

static public void hanoi0tail(char A, char B, char C,int n) {
  char T ;
  while (n>0) {
    hanoi(A,C,B,n−1) ;
    System.out.println(”premik␣iz␣” + A + ”␣na␣” + B);
    // priprava argumentov za ponovitev (rekurzivni klic)
    T=A ; // zamenjamo zeblja A in C
    A = C ;
    C = T ;
    n=n−1 ;
  } // while
} // hanoi0tail
```

Primer: izračun fakultete

```
private int fakulteta(int n) {
if (n==0) return 1;
    else  { return n * fakulteta(n-1); }
}
```

Ali je ta rekurzija repna?

Primer Fibonacci

Fibonaccijeva števila tvorijo naslednje zaporedje

n	*1, 2, 3, 4, 5, 6, 7, 8, 9, 10*
F	*1, 1, 2, 3, 5, 8, 13, 21, 34, 55*

Rekurzivno

fib(1) = 1

fib(2) = 1

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

```
public static int fib(int n) {
      if (n <= 2)
            return 1;
      else
            return fib(n-1)+fib(n-2);
}
```

Primer Fibonacci

Rekurzivno – zelo neučinkovito!!!

fib(1) = 1

fib(2) = 1

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

fib(7) = fib(6) + fib(5)

= fib(5)+ fib(4) + fib(5) =

= fib(4) + fib(3) + fib(4) + fib(4) + fib (3)

= fib(3) + fib(2) + fib(3) + fib(3) + fib(2) + fib(3) + fib(2) + fib(3)

Primer Fibonacci

Fibonaccijeva števila tvorijo naslednje zaporedje

n	*1, 2, 3, 4, 5, 6, 7, 8, 9, 10*
F	*1, 1, 2, 3, 5, 8, 13, 21, 34, 55*

<u>Iterativno _- Dinamično programiranje</u>

1) **Reši trivialne probleme in rešitve shrani**
2) **Ponavljaj: iz dosedanjih rešitev zgradi rešitve večjih problemov**

```
public static int fib(int n) {
   int n1=1, n2=1; n3;
    for(int i=2; i < n; i++) {
       n3 = n1 + n2;
       n1 = n2;
       n2 = n3;
    }
    return n2;
}
```

REKURZIJA → ITERACIJA S SKLADOM

```
public void recursive(ArgumentType args0) {
  LocalVarsType locals0 ;

  if (robniPogoj())
    sRobni ;
  else {
     s0 ;
     recursive(args1) ;
     s1 ;
     recursive(args2) ;
     ...
     recursive(argsRECCALLS) ;
     sRECCALLS ;
    } // else
} // recursive
```

REKURZIJA → ITERACIJA S SKLADOM

Na sklad shranimo:

1. Argumente
2. Lokalne spremenljivke
3. Naslov (adreso) za nadaljevanje

```
public void recursive(ArgumentType args0) {
  LocalVarsType locals0 ;

  if (robniPogoj())
    sRobni ;
  else {
     s0 ;
     recursive(args1) ;
     s1 ;
     recursive(args2) ;
     ...
     recursive(argsRECCALLS) ;
     sRECCALLS ;
  } // else
} // recursive
```

0

1

2

RECCALLS

REKURZIJA → ITERACIJA S SKLADOM

```
class StackElementType {
    ArgumentType args ; // vrednosti argumentov
                      // bodisi za zacetek rekurzivnega klica
                      // bodisi za povratek iz rekurzivnega klica
    LocalVarsType locals ;// vrednosti lokalnih spremenljivk
                           // za povratek iz rekurzivnega klica
    int address ; // vrednosti med 0 in RECCALLS;

} // class StackElementType
```

REKURZIJA → ITERACIJA S SKLADOM

```
public void iterative(ArgumentType args0) {
  LocalVarsType locals0;
  Stack st = new StackArray();
  StackElementType e;

  st.makenull();
  e.args = args0; // samo argumenti, lokalne spremenljivke se niso definirane
  e.address = 0;
  st.push(e);

  do {
    e = st.top(); st.pop();
    args0 = e.args; // pripravi vrednosti
    locals0 = e.locals; // lokalnih spremenljivk
    switch (e.address) {

         …IZVEDI DEL ALGORITMA ZA DAN NASLOV…
    } // switch
  } while (! st.empty()) ;
} // iterative
```

REKURZIJA → ITERACIJA S SKLADOM

```
public void recursive(ArgumentType args0) {
  LocalVarsType locals0 ;

  if (robniPogoj())
    sRobni ;
  else {
     s0 ;
     recursive(args1) ;
     s1 ;
     recursive(args2) ;
     ...
     recursive(argsRECCALLS) ;
     sRECCALLS ;
   } // else
} // recursive
```

```
switch (e.address) {
  case 0:
    if (robniPogoj()) sRobni;
    else {
      s0;
      // priprava za povratek
      e.address = 1;
      e.args = args0;
      e.locals = locals0;
      st.push(e);
      // priprava za zacetek rek. klica
      e.address = 0;
      e.args = args1; // ustrezno za klic
      st.push(e);
    }
    break;
```

REKURZIJA → ITERACIJA S SKLADOM

```
public void recursive(ArgumentType args0) {
  LocalVarsType locals0 ;

  if (robniPogoj())
    sRobni ;
  else {
      s0 ;
      recursive(args1) ;
      s1 ;
      recursive(args2) ;
      ...
      recursive(argsRECCALLS) ;
      sRECCALLS ;
    } // else
} // recursive
```

```
switch (e.address) {
    case i: // 0 < i < RECCALLS
      si;
      // priprava za povratek
      e.address = i + 1;
      e.args = args0;
      e.locals = locals0;
      st.push(e);
      // priprava za zacetek rek. klica
      e.address = 0;
      e.args = args_(i+1); // ustrezno za klic
      st.push(e);
      break;
    case RECCALLS: sRECCALLS;
  } // switch
```

Primer: Permutacije

Dana imamo števila od 1 do n.
Treba je izpisati vse permutacije teh števil.

Npr. za n = 3:

1 2 3
2 1 3
1 3 2
3 1 2
3 2 1
2 3 1

Vseh permutacij je n!

Primer: Permutacije

Števila imamo v polju a:

```
a = new int[max] ;
for (int i=0 ; i < a.length ; i++)
   a[i] = i+1 ;
```

Rekurzivna rešitev:

1. Rekurzijska spremenljivka? n = velikost polja
2. Kaj mi pomaga, če znam permutirati polje velikosti n-1?
   Za vsako število naredim:
     ga postavim na konec in permutiram preostalih n-1 števil
3. Robni pogoj? n = 0 (takrat lahko izpišem permutacijo)
4. Splošni primer? Glej 2.

Primer: Permutacije

```
static public void permutationsRec(int n0) {
  if (n0==0)
    writePermutation() ;
  else {
    for (int i=0, temp ; i < n0 ; i++) {
      temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
      permutationsRec(n0−1) ;
      temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
    }
  }
} // permutationsRec
```

PRIMER: PERMUTACIJE

```
static public void permutationsRec(int n0) {
0     if (n0==0)
         writePermutation() ;
      else {
         for (int i=0, temp ; i < n0 ; i++) {
            temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
            permutationsRec(n0−1) ;
1           temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
         }
      }
} // permutationsRec
```

Primer: Permutacije

Na sklad shranimo:

1. Argument `n`
2. Lokalno spremenljivko `i`
3. Naslov `address`

```
class StackElement {
   int n,i, address ;
   StackElement() {}
   StackElement(StackElement e) {
      n=e.n ; i=e.i ; address= e.address ;
   }
} // class StackElement
```

PRIMER: PERMUTACIJE

```
static public void permutationsIter(int n0) {
   StackArray s = new StackArray() ;
   StackElement e = new StackElement() ;
   int temp ;
   e.address = 0 ; e.n = n0 ; s.push(new StackElement(e)) ;
   do {
      e = (StackElement) s.top() ; s.pop() ;
      switch (e.address) {
```

…IZVEDI DEL ALGORITMA ZA DAN NASLOV…

```
      } // switch
   } while (!s.empty()) ;
} // permutationsIter
```

Primer: Permutacije

```
static public void permutationsRec(int n0) {
  if (n0==0)
    writePermutation() ;
  else {
    for (int i=0, temp ; i < n0 ; i++) {
      temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
      permutationsRec(n0−1) ;
      temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
    }
  }
} // permutationsRec
```

```
switch (e.address) {
  case 0:
    if (e.n==0)
      writePermutation() ;
    else {
      e.i=0 ; // beginning of for loop
      temp = a[e.i] ; a[e.i] = a[e.n−1] ; a[e.n−1] = temp ;
      e.address = 1 ;
      s.push(new StackElement(e)) ; // return from recursion
      e.address = 0 ; e.n = e.n − 1 ;
      s.push(new StackElement(e)) ; // recursive call
    }
    break ;
```

Primer: Permutacije

```
static public void permutationsRec(int n0) {
  if (n0==0)
    writePermutation() ;
  else {
    for (int i=0, temp ; i < n0 ; i++) {
      temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
      permutationsRec(n0−1) ;
      temp = a[i] ; a[i] = a[n0−1] ; a[n0−1] = temp ;
    }
  }
} // permutationsRec
```

```
switch (e.address) {
  case 1:
      temp = a[e.i] ; a[e.i] = a[e.n−1] ; a[e.n−1] = temp ;
      e.i ++ ;
      if (e.i < e.n) { // another loop
        temp = a[e.i] ; a[e.i] = a[e.n−1] ; a[e.n−1] = temp ;
        e.address = 1 ;
        s.push(new StackElement(e)) ; // return from recursio
        e.address = 0 ; e.n = e.n − 1 ;
        s.push(new StackElement(e)) ; // recursive call
      }
      break ;
} // switch
```

O REKURZIJI

Rekurzija...

- zahteva več režije kot iteracija in je pomnilniško bolj zahtevna od iteracije (sklici se shranjujejo na skladu),
- globina rekurzije = potrebna velikost sklada,
- ponavadi je rekurzivna koda krajša in preprostejša,
- rekurzivne probleme lahko rešujemo tudi z iteracijami:
  - *vsako repno rekurzijo lahko zamenjamo z iterativno zanko,*
    (to optimizacijo naredijo boljši prevajalniki)
  - *vsak rekurzivni program lahko spremenimo v iterativnega s skladom.*
    (to počnemo le izjemoma, če dobro poznamo algoritem in ga lahko optimiziramo bolje kot prevajalnik)

IZZIV: RAZPOLOVI ŠT. OPERACIJ NA SKLADU!

```
public void iterative(ArgumentType args0) {
  LocalVarsType locals0;
  Stack st = new StackArray();
  StackElementType e;

  st.makenull();
  e.args = args0; // samo argumenti, lokalne spremenljivke se niso definirane
  e.address = 0;
  st.push(e);

  do {
    e = st.top(); st.pop();
    args0 = e.args; // pripravi vrednosti
    locals0 = e.locals; // lokalnih spremenljivk
    switch (e.address) {

      …IZVEDI DEL ALGORITMA ZA DAN NASLOV…
    } // switch
  } while (! st.empty()) ;
} // iterative
```