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 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);
}
```

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)                                          robni pogoj
        return 1;
    else  {
        return x * potenca(x, p-1);    splošni primer
    }
}
```

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
```

hanoi('a','b','c',3)

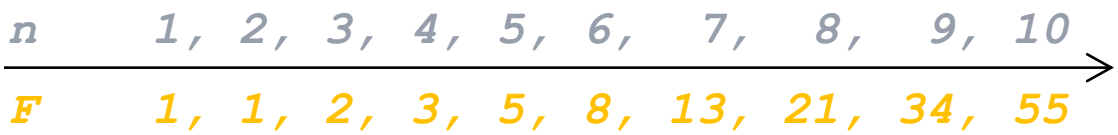
premik iz a na b
premik iz a na c
premik iz b na c
premik iz a na b
premik iz c na a
premik iz c na b
premik iz a na b

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 Fibonacci

Fibonaccijeva števila tvorijo naslednje zaporedje



Iterativno _ - Dinamično programiranje

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

```
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
```

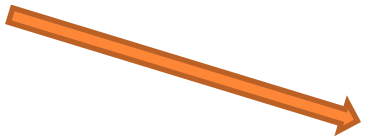
REKURZIJA → ITERACIJA S SKLADOM

```
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

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

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
```

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
```