A rekurzió a matematikában és az informatikában gyakran használt eszköz. A probléma megoldás különböző fázisaiban, így a specifikációban, az algoritmikus tervezésben és a programnyelven történő kódolásban fordulhat elő.

A *specifikáció* megadására függvényt szokás használni, amely a bemenet és a kimenet között hatékonyan adja meg a kapcsolatot. Kezdésként nézzük a hatványozás műveletét. Ennek definíciója: $x^n = x \cdot x^{n-1}$, és $x^0 = 1$, másképpen:

$$\text{hatvány}\,(x,n) = \begin{cases} 1 & \text{ha } n = 0, \\ x \cdot \text{hatvány}\,(x, n-1) & \text{ha } n > 0. \end{cases}$$

Így tehát az 5^3-t visszavezetjük $5 \cdot 5^2$-ra, vagyis 5^2-ra, azt 5^1-re, végül azt az 5^0-ra, amit már nem vezetünk vissza. A feladat rekurzív *algoritmussal* hatékonyan megoldható:

```
Hatvány(x,n):
      Ha n=0 akkor szorzat:=1
              különben szorzat:=x*Hatvány(x,n-1)
      Elágazás vége
      Hatvány:= szorzat
Függvény vége
```

Funkcionális programozási nyelven, például Imagine Logo-ban a kódolás természetesen következik:

```
eljárás Hatvány :x :n
      ha :n = 0 [eredmény 1]
      eredmény :x * Hatvány :x :n - 1
vége
```

A nemrekurzív nyelvek is engedélyezhetik a kódolást, így például Pascal nyelven a kód:

```
Function Hatvany (x, n : integer) : integer;
var szorzat : integer;
begin
      if n=0 then szorzat:=1
               else szorzat:=x*Hatvany(x,n-1);
      Hatvany:=szorzat;
end;
```

Az iteratív módon meghatározott algoritmusok, programok átírhatók rekurzívvá. A legfontosabb algoritmikus egységekkel, a szekvenciával és az elágazással nincs gond, változtatás nélkül átvihetők. A ciklust tartalmazó eljárásokat kell rekurzívvá tenni. Az elöltesztelő ciklus átírása:

Iteráció

```
Eljárás(X):
    Ciklus amíg T(X)
        S(X)
    Ciklus vége
Eljárás vége
```

Rekurzió

```
Eljárás(X):
    Ha T(X) akkor
        S(X)
        Eljárás(X)
    Elágazás vége
Eljárás vége
```

Minta: Állítsuk elő a természetes számok sorozatát A-tól B-ig ($1 \leq A, B \leq 100$). Például az `Előállít(5, 10, Sor, 1)` eljárás a `Sor` tömbbe elhelyezi az 5 6 7 8 9 10 számsort.

I. Iteratív megoldás:

```
Konstans sorh = 100
Típus TSor = Tömb[0..sorh: Egész]
Változó Sor : Tsor

Eljárás Előállít(Változó A,B: Egész; Változó Sor: Tsor; Változó mut: Egész):
       Ciklus i:=A-tól B-ig
           Sor[mut]:=i
           mut:=mut+1
       Ciklus vége
Eljárás vége
```

II. Rekurzív megoldás:

```
Konstans sorh = 100
Típus TSor = Tömb[0..sorh: Egész]
Változó Sor : Tsor
```

```
Eljárás Előállít(Változó A,B: Egész; Változó Sor: Tsor; Változó mut: Egész):
      Ha A<=B akkor
                  Sor[mut]:=A
                  mut:=mut+1
                  Előállít(A+1, B, Sor, mut)
      Elágazás vége
Eljárás vége
```

Készítsünk programot, amely számsorozatokat állít elő úgy, hogy a programban ciklus utasítást nem használunk.

a) Írjunk függvényt, ami egy N ($N \geq 1$) magas számhegyet hoz létre!
Példa: `számhegy 5` Eredménye: 1 2 3 4 5 4 3 2 1;

b) Írjunk függvényt, ami egy N magas számhegységet hoz létre!
Példa: `számhegység 4` Eredménye: 1 1 2 1 1 2 3 2 1 1 2 3 4 3 2 1;

c) Írjunk függvényt, ami egy N magas számlépcsőt hoz létre!
Példa: `számlépcső 6` Eredménye: 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6;

d) Írjuk fel az alábbi, rekurzív képlettel megadott sorozat első N elemét:
$a_1 = -1$ és $a_n = -4 - 3a_{n-1}$.

A program első argumentuma N értéke és a második egy kimeneti fájl legyen. A kimeneti fájlban soronként jelenítsük meg a sorozatok szóközzel elválasztott elemeit. (A fájlba íráskor se használjunk ciklust.)

Beküldendő a program forráskódja (`i259.pas`, `i259.cpp`, ...), valamint a program rövid dokumentációja (`i259.txt`, `i259.pdf`, ...), amely tartalmazza a megoldás rövid leírását, és megadja, hogy a forrásállomány melyik fejlesztő környezetben fordítható.