Rekursioon

Def. 1

Protseduur või funktsioon on **rekursiivne**, kui ta kutsub iseennast välja.

Def. 2
Rekursioon - vt. rekursioon.

Rekursioon, rekursiivsus e enesepoolepöördumine ei ole ainuomane programmeerimises kasutatavatele algoritmidele. Rekursiivseid kirjeldusi / definitsioone võib kohata mujalgi. Rekursiivsel defineerimisel kirjeldatakse objekt iseenda nö lihtsama eksemplari kaudu. Definitsioon määrab objekti tekkimise protsessi. Oluline on, et protsess peab mingil hetkel lõppema – seega rekursiivses definitsioonis on tavaliselt vähemalt kaks osa: üks üldise juhu jaoks (rekursiivne) ja teine viimase, lihtsama juhu jaoks (mitte rekursiivne).
Armastatud näide on faktoriaal:

```
n! = 1, kui n = 0, {mitterekursiivne erijuht}
   = n*(n-1)!, kui n>0 {rekursiivne üldjuht}
```

Ka meile tuttavad summeerimine `(summa = 0, summa = summa + arv)` ja loendamine `(loendur = loendur + 1)` on olemuselt rekursiivsed, sest kirjeldavad summa leidmist tema enda kaudu, samuti on olemas lihtne piirjuht.
Funktsioon rekursiivseks faktoriaali leidmiseks oleks järgmine:

```
def Fact(n):
# Erijuht, kui n on 0
   if n == 0:
      return 1
   else:
# rekursiivne üldjuht
      return n*Fact(n-1);
```

Tihti on rekursiivne algoritm kirjutatav ka iteratiivsel kujul (st kasutades tsüklit, nagu toodud näidetes), kuid rekursioon võib algoritmi lihtsustada, säästa meid ülearuste muutujate deklareerimisest, kiirendada programmi täitmist jne.

Funktsiooni `Fact(3)` täitmine koos pinuga:

```
Pinu: tühi
1. Väljakutse: Fact(3)
Pinusse pannakse: N=3
2. Väljakutse: Fact(2)
Pinusse pannakse: N=2
3. Väljakutse: Fact(1)
Pinusse pannakse: N=1
4. Väljakutse: Fact(0)
Piirjuht: Fact:=1, 4. väljakutse lõpp
Pinust võetakse N=1, Fact=1
Lõpetatakse avaldis: Fact:=N*Fakt e Fact:=1*1
3. väljakutse lõpp
Pinust võetakse N=2, Fact=1
Lõpetatakse avaldis: Fact:=N*Fact e Fact:=2*1
2. väljakutse lõpp
Pinust võetakse N=3, Fact=2
Lõpetatakse avaldis: Fact:=N*Fact e Fact:=3*2
1. väljakutse lõpp
Vastus on käes: Fact=6
```

Probleemi lahendamise teeb tihti keeruliseks andmehulk ja -struktuur, mida töödelda tuleb.

Rekursiooni idee abil vähendatakse probleem sellisesse mõõtu, millega on võimalik hakkama saada ja väikestest lahendustest ehitatakse üles kogu lahendus. Kõige raskem on õige tüki leidmine, mille lahendamisest oleks kasu kogu probleemi lahendamiseks. Tegemist pole klassikalise "jaga ja valitse"-printsiibiga, mida struktuurprogrammeerimise ja protseduuridega tutvumise käigus selgitatakse. Jagada ja valitseda tuleb sarnaseid rekursiivseid osi. Rekursiivsed algoritmid (ja struktuurid) peavad läbi lihtsustumise jõudma välja nn baasjuhtumini.

Rekursiivsed on mõned sorteerimisalgoritmid, avaldiste töötlemise algoritmid, kõigi variantide läbivaatamise (nn täisläbivaatuse) algoritmid jms.
Mõned matemaatilised suurused on oma olemuselt rekursiivsed. Olemuselt rekursiivsete suuruste leidmiseks ja rekursiivsete struktuuride töötlemiseks on loomulik kasutada ka rekursiivset algoritmi.

Rekursiivse algoritmi realiseerimisel ei pääse alamprogrammide kirjutamisest ja sellele mõtlemisest, kuidas muutujaid alamprogrammi toimetada (parameetrid) ja sealt uuesti kätte saada. Samuti tasub mõelda pinu(de)st, kus kõiki pooleli jäänud alamprogramme koos muutujate ja leitud väärtustega hoitakse. Enamasti pole siin kasu globaalsete muutujate kasutamisest.

Rekursiivse algoritmi sees pöördub AP iseenda poole ja kutsub ennast välja muudetud parameetritega. Väljakutse peab olema sõltuvuses mingite tingimuste täidetusest. Vastasel juhul ei teki rekursiooni lõppu. Viimane alamprogrammi väljakutse on nn **lõpuväljakutse** *(limit call*), mis realiseerib lõppjuhu. Peab olema põhjus rekursiooni lõpetamiseks. Kui see põhjus on tekkinud, uut pöördumist alamprogrammi poole ei tehta, vaid käsilolev AP täidetakse lõpuni.