rendszerprogramok és a fejlesztési programok ugyanazt a programozási nyelvet használhatják. Legújabb C++ változata már az *objektumorientált programozást is lehetővé teszi.

Pascal - az *erősen típusos nyelvek első képviselője, magasszintű algoritmikus *programozási nyelv. 1968-ban N.Wirth elsősorban oktatási célokra tervezte. Közérthetősége, egyszerűsége, ugyanakkor tudományos megalapozottsága és viszonylag könnyű megvalósíthatósága révén igen elterjedt, elsősorban a *mikroprocesszoros rendszerek (*személyi számítógépek) területén. Népszerűségéhez nagyban hozzájárult a Jensen-Wirth szerzőpáros által 1974-ben megjelentetett pontos nyelvleírás. Végleges formáját 1982-ben standardizálta az *ISO. Mind az *adatstruktúrák definiálásában, mind a *strukturált programozásban használt vezérlőszerkezetek megvalósításában hatékony, könnyen használható eszközöket biztosít a programozóknak. Magas szinten biztosítja a *rekurzivitást. Moduláris felépítése, blokkstruktúrája szinte kényszeríti a programozót arra, hogy jól strukturált, magas hatásfokú programokat írjon. A ~ az első olyan nyelv, amely ténylegesen gyakorlatba ültette az *[adat]típus fogalmát, s így kiindulópontja a legmodernebb algoritmikus nyelveknek (*Modula, *CHILL, *Ada).

Jodál Endre

ALGORITMUSOK

3. Függvények és eljárások

Sokszor megtörténik, hogy valamilyen feladat megoldásában a matematikából jól ismert függvények is szerepelnek. Ezeket természetes módon használhatjuk az algoritmus leírásában. Ha például, az a feladatunk, hogy számítsuk ki az e alapú logaritmus-függvény értékeit egy megadott intervallumban, akkor ezt így írhatjuk le:

```
Adott a,b,l                          { a,b,l pozitív számok }
Minden x:=a,b,l -re végezd el    { x értéke sorra a,a+l,a+2l,..}
   Eredmény  x, ln x
(Minden)vége
```

Ha a használt függvény nem közismert, akkor azt is meg kell adni a leírás során egy különálló részben. Ez a rész a FÜGGVÉNY szóval kezdődik, és a FÜGGVÉNY VÉGE szavakkal végződik. A függvény definiálása az eddig

ismertetett utasítások segítségével történik. Minden függvénynek kell, hogy legyen neve (ez az adott algoritmus-leírásban egyértelmű kell, hogy legyen), és ezenkívül egy vagy több argumentuma (esetleg egy sem, de ez ritka eset).

Példaként, számítsuk ki n adott természetes szám legnagyobb közös osztóját. Mivel két szám legnagyobb közös osztója egyetlen szám, ennek megkeresésére az első részben bemutatott bármelyik algoritmust függvényként értelmezhetjük. Ennek leírása a következő:

```
FÜGGVÉNY  lnko (a,b)
m := a
n := b
Amíg  n > 0  végezd el
      r := m - [m/n] n    { m-nek n-nel való osztási maradéka}
      m := n
      n := r
(Amíg)vége
lnko := m
FÜGGVÉNY VÉGE
```

Figyeljük meg, hogy a leírásban a függvény neve (argumentumok nélkül) értéket kap. Ez lesz a függvény értéke az adott argumentumokra. Itt a és b formális argumentum, ami azt jelenti, hogy a függvényre való hivatkozáskor a és b helyén bármilyen más egész érték, változó esetleg kifejezés szerepelhet. Ekkor a függvény az így megadott aktuális értékekre számítja ki a legnagyobb közös osztót. Igy pl. lnko(10,20) egyenlő 10-zel, míg lnko(x,y), ahol x=25 és y=35, egyenlő lesz 5-tel.

Alkalmazzuk ezt a függvény előbbi feladatunk megoldására. Először kiszámítjuk az első két szám legnagyobb közös osztóját, majd a kapott eredménynek és a harmadik számnak a legnagyobb közös osztóját és így tovább.

```
Adottak n, xi, i=1,2,...n
d := x1                    { legyen d az első szám }
Minden i:=2,n -re végezd el    {i =2,3,...,n }
d := lnko (d,xi)           { d és xi lnko-ja lesz d új értéke }
(Minden)vége
Eredmény d
```

Az algoritmusnak azon részét amely többször ismétlődik, vagy amelyet több algoritmus is használhat, érdemes a függvényhez hasonlóan külön önálló egységként megadni. Ennek eljárás a neve. Leginkább abban különbözik a függvénytől, hogy eredménye nem egy érték, mint a függvény esetében, hanem több vagy egy sem, mert lehet, hogy csak átrendezi az adatokat. Hasonlóképpen adjuk meg, mint a függvényt, kezdetét az ELJÁRÁS szó jelöli,

a végét pedig az ELJÁRÁS VÉGE. Az algoritmus leírásában csupán a nevével s az aktuális paramétereivel (argumentumaival) hívatkozunk rá. Mutassuk be az eljárás használatát egy nagyon egyszerű példán. Egy szöveget oldalakba akarunk tördelni úgy, hogy minden oldal egy megadott fejléccel kezdődjék. A fejléc kiírását egy FEJLÉCIRÁS nevű eljárás valósítja meg, hogy hogyan, arra most nem térünk ki. A szöveg egy adott sorának a kiírását a SORIRÁS nevű eljárás végzi el. Tételezzük fel, hogy minden oldal 30 sort tartalmaz. Az i változó számolja a sorokat, ha ez 30-nak többszöröse akkor abba a sorba ki kell írni a fejlécet.

```
Adott  a szöveg
i:=0
Amíg a szövegnek nincs vége végezd el
   Ha [i/30]*30 =i  akkor            { i 30-nak többszöröse }
      FEJLÉCIRÁS
      i:=i+1
  (Ha)vége
   SORIRÁS
   i:=i+1
(Amíg)vége
```

Minden sor kiírása után növelni kell az i-t, akár fejlécet írtunk ki (ez csupán egy sorból áll), akár közönséges sort.

Természetesen, minden függvény átírható eljárássá is. Térjünk vissza első példánkhoz, és írjuk meg a legnagyobb közös osztó megkeresésének algoritmusát eljárás formájában!

```
ELJÁRÁS lnko (a,b,c)      { c az a és b lnko-ja }
m := a
n := b
Amíg n > 0 végezd el
    r := m-[m/n] n
    m := n
    n := r
(Amíg)vége
c := n
ELJÁRÁS VÉGE
```

Itt a és b bemeneti, míg c kimeneti paraméter. Algoritmusunk a következő:

```
Adottak n, xi, i:=1,2,...,n
d := x1
Minden i:=2,n -re végezd el
   lnko (d,xi,c)          { d és xi lnko-ja lesz d új értéke }
   d := c
(Minden)vége
Eredmény d
```

A d:=c értékadásra azért van szükség, hogy az Inko eljárás újabb hívásakor az első paraméter az addig lefutott számok legnagyobb közös osztója legyen. Természetesen a ciklus két utasítása ebben az esetben eggyé ötvözhető. Tehát

```
Minden i:=2,n -re végezd el
    Inko (d,xi,d)
(Minden)vége
```

Bizonyos programrészt akkor is érdemes külön eljárásként megírni, ha ezzel érthetőbbé tesszük a leírást. Lássunk erre példát cikksorozatunk első részéből! Ott egyik feladatunk az volt, hogy sorba kellett rendeznünk egy adott számsorozatot. Ezt az egymás mellett lévő számok felcserélésével, az ún. buborékos rendezés segítségével oldottuk meg. Ha két szám felcserélését külön eljárásként írjuk meg, akkor a leírás a következő lesz:

```
ELJÁRÁS CSERE (a,b)      { fölcseréli a-t b-vel }
t := a
a := b
b := t
ELJÁRÁS VÉGE
```

Az algoritmus pedig a következő:

```
Adatok n, xi, i=1,2,...,n
Ismételd
   jel := 0
   Minden i := 1,n-1 -re végezd el
      Ha xi > xi+1 akkor
         CSERE (xi, xi+1)
         jel := 1
      (Ha)vége
   (Minden)vége
ameddig jel = 0
Eredmény xi, i=1,2,...,n
```

Előző cikkünkben, amikor az algoritmusok tervezéséről írtunk, tulajdonképpen hallgatólagosan már használtunk eljárás típusú elemeket. Az eljárások és függvények használata nagyon megkönnyíti az algoritmusok tervezését és leírását. Ezért használatuk igen ajánlott.

dr. Kása Zoltán