Az Algol programozási nyelv

BENEDEK PÁL*
MEZEI MIHÁLY*

Az előző közleményben egy feladatot ismertettünk [1] és kidolgoztuk ennek két megoldásmenetét, amelyek közül az egyik speciális (első változat), a másik kissé általánosabb (második változat) volt. Ez utóbbit számítási folyamatábrában rögzítettük. A számítási folyamatábra a program terve. Most ezt a tervet valósítjuk meg: programot írunk, mégpedig az Algol programozási nyelv GIER gépre írt variánsát a GIER—ALGOL-4 reprezentációt használjuk. E cikk függelékében mindkét változat programját közöljük. Valahányszor az Algol programozási nyelv ismertetésében új fogalom tárgyalására kerül sor, e programokra fogunk hivatkozni, oly módon, hogy megadjuk melyik változat, hányadik sorában található a példa.

Egy Algol program három jeltípusból épül fel: Algol alapjelekből, számokból és azonosítókból.

Alapjelek

Az alapjelek lehetnek műveleti jelek, zárójelek, elválasztójelek, valamint bizonyos aláhúzott angol szavak. Mindegyiknek megvan a speciális jelentése, van amelyik magától értetődő és közismert, (pl. a zárójel, +, −, ·-jelek) mások viszont csak az Algol nyelvben használtak (az elsőre furcsának tűnő ↑ jel pl. a hatványozás jele). Az alapjeleket és jelentésüket folyamatosan vezetjük be. A zárójel nyitva, zárójel zárva megkülönböztetett jelek. Figyeljük meg a zárójel használatát az 1. változat 32. sorában *(Függelék)*.

Itt jegyezzük meg, hogy a programban legépeléskor a program áttekinthetőségének növelése céljából gyakorlatilag tetszés szerinti helyen kezdhetünk új sort, és nagy szabadsággal helyezhetünk el szóközt, ami a program áttekinthetőségét elősegíti.

Számok

Számításaink során gyakran szerepelnek rögzített konstansok, pl. feladatunkban az egyensúlyi állandó hőmérsékletfüggését kifejező függvényben előforduló konstansok, melyeknek pontos értékét a programban kell megadni.

Az Algol programban szereplő számok nem sokban térnek el az általában használt számoktól, előjelük lehet tizedespont (de nem tizedes vessző) szerepelhet bennük (pl. a programban a K függvényben szereplő —0.07685). Újdonság viszont például a programban szereplő $1.4052_{10}-10$. A_{10} jel mögött álló egész szám azt jelenti, hogy az előtte álló számot 10 hányadik hatványával kell megszorozni. (Tehát pl. 1.23 azonos $123_{10}-2$-vel.)

Algol alapjelekre tehát máris láthatunk néhány példát:

+, −, ·, $_{10}$ jelek.

Azonosítók

Amikor a feladatunk megoldására szolgáló algoritmust kidolgoztuk, az egyes mennyiségeket valamilyen névvel láttuk el, és ezekkel a nevekkel „végeztük el", pontosabban jelöltük ki az elvégzendő műveleteket. A programban az egyes mennyiségeket ugyancsak névvel kell ellátnunk, hogy a velük elvégzendő műveleteket elő tudjuk írni. Ezt a nevet hívják azonosítónak. Azonosító lehet minden betűvel kezdődő, betűkből és számokból álló jelsorozat. Ilyen azonosítókat olvashatunk pl. mindkét programváltozat harmadik sorában. Általában az egyes változókat olyan azonosítóval látják el, ami utal az illető változó nevére (pl. az egyensúlyi koncentrációt számító programrészben t, ill. T-vel a Celsius, ill. Kelvin fokban kifejezett hőmérsékletet jelenti, de a program ugyanúgy működne, ha t helyett mindenütt temp-et írtunk volna).

Azonosítóval látjuk el a változókon kívül az utasításokat jelölő címkéket, és az eljárásokat (szubrutin). Ezekről azonban később lesz szó.

Változók típusai

A változók szerepét már láttuk: egy konkrét matematikai mennyiséget jelentenek. Az Algol nyelvben három típusú változó van: egész, valós és logikai. Az egész típusú változó értéke mindig egész szám, a valós típusé viszont tetszőleges szám.

Az első programunk 3. sorában a következő valós változókat definiáljuk:

t, delta, mCO, mCO2, mH2, mH2O, MIn

Második programunk 4. sorában a következő integer változókat definiáljuk:

i, n

A megkülönböztetést azért teszi a nyelv, mert az egész számokat fixpontosan, a nem egészeket pedig lebegőpontosan tárolja a gép, és így értelemszerűen más gépi utasítással lehet hozzájuk jutni és műveleteket végezni velük.

A logikai változóknak két értéke lehet:

true vagy false (a true és false algol alapjelek, és az igaz, ill. a hamis értéket jelentik). Szerepüket majd a később tárgyalandó feltételes utasításokban részletezik.

Műveleti jelek, zárójelek és kifejezések

Konstansokból és változókból műveleti jelek segítségével különböző kifejezéseket építhetünk fel.

* Magyar Vegyipari Egyesülés Mérnöki Iroda, Budapest.

Először felsoroljuk a műveleti jeleket:

↑ : hatványozás,

×, /, :, : szorzás, osztás, ill. egész-osztás,

+, − : összeadás, kivonás.

Magyarázatra csak a : jel szorul, az egész-osztás műveleti jel. i:j eredménye az i/j hányados abszolútértékének egész része, a hányadossal megegyező előjellel, tehát lényegében a hányados „lekerekítése". (Egy szám egész részén azt a legnagyobb egész számot értjük, amelyik az illető számnál még nem nagyobb, pl. a 2.3 egészrésze 2, de 5 egészrésze 5). Az egész-osztás csak egész típusú változók között végezhető el.

Meg kell ezenkívül jegyeznünk, hogy a / jelet *csak* osztás jelként használhatjuk, zárójelként viszont nem.

Egyszerű aritmetikai kifejezésre példa mindkét programunkban az iners gázok móltörtjét jelentő kifejezés (az 1. változat 36. sorában):

$$1 - mCO - mH2O - mCO2 - mH2$$

Bonyolultabb kifejezésnél, ahol pl. összeadás mellett szorzás is szerepel, a kijelölt műveletek elvégzési sorrendjét is meg kell határozni, hiszen az összeadás és szorzás nem cserélhető fel (pl. $2 \times (3 + 5)$ nem egyenlő $(2 \times 3) + 5$-tel). Ezt a célt szolgálják az értékelési (precedencia) szabályok és a zárójelek.

Az értékelési szabályok alapján a műveletek csoportokba oszthatók. A leírásuknál egy sorba írtuk az egy csoportba tartozó műveleteket. A csoportok között precedencia áll fenn, amin azt értjük, hogy az értékelésben először a legnagyobb precedenciájú műveletet, a hatványozást kell elvégezni, azután a hatványozásnál alacsonyabb precedenciájú osztásokat és szorszásokat, végül a legalacsonyabb predenciájú műveleteket, az összeadásokat és a kivonásokat. Az olyan kifejezéseket, amelyekben azonos precedenciájú műveletek szerepelnek, balról jobbra haladva értékeli a program. Tehát pl. a/b×c nem a/(b×c)-t, hanem (a×c)/b-t jelent). Ezt a természetes precedenciát zárójelezéssel viszont tetszőleges módon változtathatjuk, mert alapvető szabály az, hogy a program a zárójelekbe tett kifejezéseket teljesen önállóan értékeli ki.

Bonyolultabb aritmetikai kifejezésre példa mindkét programunkban delta kifejezése (pl. az 1. változat 51 − 52. sora):

$$(K \times mCO \times mH2O - mCO2 - mH2)/(K \times (mCO + mH2O) + mCO2 + mH2)$$

Jól láthatjuk, hogy a műveletek elvégzési sorrendjét hogyan lehet zárójelek segítségével befolyásolni.

A hatványozást tartalmazó kifejezésre példa lehet egy R sugarú gömb térfogata:

$$4 \times R \uparrow 3/3 \times 3.141592$$

Látjuk, hogy az R ↑ 3-t nem kell zárójelbe tenni, mert a hatványozás a szorzásnál nagyobb precedenciájú.

Az Algol program felépítése. Blokk-szerkezet, deklaráció

Egy Algol program blokkokból, vagy speciálisan egy blokkból épül fel.

Egy blokk a begin alapjellel kezdődik és az end alapjellel végződik.

A begin után deklarációk következnek.

A deklarációk nem végrehajtandó utasítások, hanem a fordítóprogramnak szóló közlések arra vonatkozóan, hogy az illető blokkban milyen nevű és típusú mennyiségeket fogunk használni.

A korábbiakban lerögzítettük, hogy három típusú változót ismer az algol nyelv. A típust azonban az azonosítójuk alapján nem lehet megállapítani, tehát külön kell azt a programban megjelölni. Ezt a célt szolgálja a deklaráció.

A változók deklarálásakor a real (valós), integer (egész) és boolean (logikai) alapjelek után az illető blokkban szereplő típusú változók azonosítóit soroljuk fel.

Az azonosítókat vesszővel választjuk el egymástól, és a felsorolást pontosvesszővel zárjuk. Pl. második programunk fő blokkjában (azaz az első begin után) csak valós majd egész változókat deklaráltunk:

```
begin
  real t,MCO, MCO2, MH2, MH2O;
  integer i, n;
```

Deklarálhatunk a változók mellett tömböket (array) eljárásokat (procedure) és kapcsolókat (switch), ezekről azonban később lesz szó.

A deklarációk után következnek az utasítások, amelyeket a gép a leírásuk sorrendjében hajt végre.

Minden utasítást pontosvesszővel vagy az end, ill. else alapjelek egyikével (utasítás terminátor) kell lezárni. Megjegyezzük még, hogy az egyes utasításokat az áttekinthetőség fokozására külön sorba szokták írni, noha a nyelv erre nem kötelez. További hasznos szokás, hogy minden begin után a sorkezdés beljebb megy, és az end után pedig kijjebb. Ily módon a blokk-szerkezet jól láthatóvá válik.

Értékadás

A legegyszerűbb utasítás az értékadás:

Az értékadó utasításban a:=algol alapjel egy változó azonosítója (ez kapja az új értéket), és egy kifejezés (amely persze lehet, hogy csak egy egyszerű változóból, vagy egy konstansból áll) között áll. Ennek a kifejezésnek az aktuális értéke lesz a baloldalon levő változó új értéke:

$$MIn := 1 - mCO - mH2O - mCO2 - mH2;$$

A fenti utasítás hatására MIn értéke az elegyben levő iners gáz koncentrációja lesz (1. változat 36. sor).

Egyetlen értékadó utasítással több változó is kaphat egyszerre értéket a következő módon:

$$i := j := 0;$$

Ilyen utasítás hatására az i és j nevű változók értéke nulla lesz.

Fontos megjegyezni azt, hogy azoknak a változóknak, melyeknek ily módon tudunk egyszerre értéket adni azonos típusúnak kell lenniök.

Az értékadó utasítás másik érdekessége az, hogy a baloldalon szereplő változó szerepelhet a jobboldalon is (1. változat 53. sor):

```
mCO:=mCO-delta;
```

Ilyenkor a jobboldalon mCO eredeti értékével számol a program, tehát az utasítás végrehajtása előtti értékkel. Ebből vonja le deltát, és az így kapott szám lesz mCO új értéke.

Komment

A comment alapjel és az utána következő első pontosvessző közötti szöveget a program figyelmen kívül hagyja, így lehetőségünk van arra, hogy a program szövegébe magyarázatokat, emlékeztető megjegyzéseket iktathatunk.

Programjainkban az első komment a program feladatát mondja meg az olvasónak:

```
comment VÍZGÁZ EGYENSÚLY SZÁMÍTÓ
PROGRAM 1. változat;
```

Kommentet a begin valamint a ; alapjelek után írhatunk. Ugyancsak figyelmen kívül hagyja a program az end és az azt követő utasításterminátor (;, end, v. else alapjelek) közötti szöveget (pl. a 2. változat 66. sor).

```
end EGYENSÚLY;
```

Így könnyű észrevenni, hogy ez az end az EGYENSÚLY eljárás blokkjának lezáró end-je. (Hogy mit jelent az eljárás, azt később tárgyaljuk.)

Címkék

Minden utasítást címkével láthatunk el. Címke tetszőleges, az illető blokkban még nem szereplő azonosító, vagy előjel nélküli egész szám lehet.

Erre példa első programunkban az iterációs formula előtt álló címke (50. sor):

```
ISM:
    delta:=(K×mCO×mH2O-mCO2×mH2)/
           K×(mCO+mH2O)+mCO2+mH2).
```

Láthatjuk, hogy a címke azonosítója és az illető címkéhez tartozó utasítás közé kettőspontot kell írni.

Ha egy utasításnak címkét is adunk, akkor lehetőség nyílik arra, hogy a program más utasítások végrehajtása után ennél az utasításnál folytassa a számolást (a később tárgyalandó ugrató utasítások segítségével). Ily módon nem is az utasításra, hanem az utasítás *helyére* vonatkozik a címke.

Kapcsolódó deklarációk

A cimkéket — a változókkal ellentétben — nem kell (sőt nem szabad) deklarálni. Deklarálhatunk viszont, mint már említettük, kapcsolókat (switch).

Egy kapcsoló lényegében egy azonosítóval ellátott cimkelista. A deklarálást a következő, második programunkban levő példa mutatja (67. sor):

```
begin
switch KAPCS:=L1,L2,END;
```

ahol KAPCS a kapcsoló azonosítója, L1, L2, END, pedig címkék. Ezekre a cimkékre azután nemcsak a nevükkel hanem mint a KAPCS kapcsoló elős második, ill. harmadik elemeként is hivatkozhatunk. Adott esetben ez a program szervező részének megírását kényelmessé teszi:

Példánkban:

```
L1   egyenértékű KAPCS[1]-gyel
L2   egyenértékű KAPCS[2]-vel
END  egyenértékű KAPCS[3]-mal
```

Látjuk, hogy a kapcsoló azonosítója után szögletes zárójelben levő kifejezés értéke (a jelenesetben az 1, 2, 3, konstansok) mondja meg, hogy a KAPCS kapcsoló hanyadik eleméről van szó.

A szögletes zárójel a második fajta zárójel, amivel eddig az Algol nyelvben találkoztunk.

Ugrató utasítás

Mint említettük, a program az utasításokat leírásuk sorrendjében hajtja végre. Ezt a természetes sorrendet ugrató utasítással lehet megváltoztatni.

Egy ugrató utasítás a go to alapjelből és egy utána álló cimkéből vagy kapcsolóelemből áll.

Pl: első programunk az eredmények kiírása után nem éri el az utolsó end-et, mert előtte szerepel a

```
go to INPUT;
```

utasítás, aminek hatására az eredmény kiírása után a következő végrehajtandó utasítás az lesz ismét, aminek cimkéje a program 11. sorában áll:

```
writetext(*<
Homeerseklet:*);
```

lesz.

A második programunkban kapcsolót is deklarálunk, így ugyanarra a helyre való ugratást kétféleképpen is megoldhatjuk, pl. hibás input adatok esetén (50. sor):

```
go to KAPCS[3];
```

utasítással ugrattunk a program végére, más helyen a

```
go to END;
```

utasítást alkalmaztuk.

Fontos megjegyezni, hogy a go to alapjel után nem állhat olyan címke azonosítója, amelyik az illető utasítás helyéhez képest egy belső blokkban ban, tehát pl. az első programunkban a go to INPUT; utasítás helyén nem állhat a go to ISM; utasítás, mert az ISM címke egy belső blokkban van.

Utasítás-zárójel

A begin és end alapjelek közé írt utasításokat az Algol nyelv egyetlen, összetett utasításként kezeli, és hajtja végre, tehát egy blokk is összetett utasítás. A begin és end alapjelek ebben a vonatkozásban utasítás-zárójelek. Nem szabad azonban elfelejteni, hogy a blokk és az összetett utasítás *nem* azonos, mert a blokk elején deklarációk is vannak. Az utasítás-zárójel szerepét a feltételes utasításokban és a ciklus-utasításokban fogjuk látni.

Logikai kifejezések

Logikai kifejezésben szerepelhetnek logikai változók, valamint relációk. Reláció két, relációjellel összekapcsolt aritmetikai kifejezés pl.:

x = 1,
2 × y + 5 ≧ b ↑ 2,
stb.

Relációjelek a következők:

= egyenlő
≠ nem egyenlő
> nagyobb
≧ nagyobb v. egyenlő
< kisebb
≦ kisebb v. egyenlő

A reláció értéke true vagy false aszerint, hogy a reláció a benne szereplő változók aktuális értékével teljesül-e, vagy nem.

A logikai kifejezések a logikai változókból és relációkból logikai műveletek segítségével épülnek fel. A következő logikai műveletek szerepelnek az Algol nyelvben:

—	negáció	(tagadás)
∧	konjunkció	(és)
∨	diszjunkció	(vagy)
=>	implikáció	(ha, akkor...)
≡	azonosság	(azonosan egyenlő)

Ezek a műveletek jól ismertek a logikai ítélet kalkulusból: —, A értéke true, ha A értéke false, viszont false ha A értéke true,

A ∧ B értéke csak akkor true, ha A és B értéke egyaránt true, egyébként false,

A ∨ B értéke csak akkor *false*, ha A *és* B értéke false, egyébként true,

A => B értéke csak akkor false, ha A true de B false, egyébként true,

A ≡ B értéke true, ha A és B értéke ugyanaz, egyébként false.

Hasonlóan az aritmetikai kifejezésekhez, is vannak értékelési (precedencia) szabályok. A műveleteket precedenciájuk sorrendjében írtuk fel. Az értékelési sorrendet (gömbölyű) zárójellel meg lehet változtatni.

Mindkét programunkban szerepel (pl. 1. változat 39. sor) a

MIn < 0 ∨ MIn > 1 ∨ T ≦ 0

logikai kifejezés. Mint látjuk, itt három reláció diszjunkciója szerepel. A kifejezés értéke akkor true, ha legalább egy reláció true, (tehát az iners gázok móltörtje vagy kisebb 1-nél vagy nagyobb nullánál, vagy pedig a megadott hőmérséklet negatív) és az utasítás (hibajelzés) csak ebben az esetben hajtódik végre. Látjuk, hogy bármelyik reláció teljesülése elvi hibát jelent az adatrendszerben, a hibajelzés tehát jogos.

Feltételes utasítás, feltételes kifejezés

A feltételes utasítás, ill. kifejezés lényege az, hogy egy logikai kifejezés aktuális értéke alapján dönt a program két lehetősége között.

Például az első programunkban az iteráció folytatását, ill. abbahagyását az

if abs (delta) > 0.0001 then go to ISM;

feltételes utasítás vezérel (60. sor).
Mint látjuk az if és then alapjelek között egy logikai kifejezés áll, utána pedig egy utasítás. Ha ennek a logikai kifejezésnek értéke true, vagyis delta abszolút értéke 0.0001-nél nagyobb, akkor végrehajtódik a then után levő utasítás:

go to ISM;

vagyis a program új iterációs lépést kezd. Ha viszont a logikai kifejezés értéke false, vagyis delta értéke legfeljebb 0.0001, akkor az ugrató utasítás nem hajtódik végre, és a program a következő utasítás végrehajtásával a végeredményt kezdi kiírni. Itt tehát az egyik lehetőség a then utáni utasítás végrehajtása, a másik pedig a végre nem hajtása.

A feltételes utasítás második típusában a logikai kifejezésen kívül két utasítás szerepel, és a logikai kifejezés értéke szerint az elsőt vagy a másodikat hajtsa végre a program.
A 2. változat 86. sorában pl. a

go to KAPCS [i];

utasítás helyett írhattuk volna az

if i = 1 then go to L1 else go to L2;

utasítást.

Mint látjuk, megint az if és a then lapjelek között van a logikai kifejezés. Ha ennek értéke true akkor a then és else alapjelek közötti utasítás, ha pedig false, akkor az else utáni utasítás hajtódik végre. (Mivel programunkban ezen a helyen csak i = 1 és i = 2-nek van értelme, nyilvánvaló hogy ez az utasítás egyenértékű a programban használttal.) Meg kell jegyeznünk, hogy az else utáni utasítás lehet feltételes utasítás is (így tehát feltételláncokat tudunk felépíteni), a then után viszont nem. (A then után álló if bizonyos esetekben kétértelműséget eredményezne, ezért itt nem szabad használni),

Összetett utasítás (vagyis begin és end utasítás zárójelek közé zárt utasítások) mindkét helyen szerepelhet.

A feltételes kifejezések a második típusú feltételes utasításokkal analógok, tehát a kifejezés értékét eldöntő logikai kifejezés mellett két kifejezést kell megadni, melyeket az if, then és else alapjelek választanak el egymástól. Programunkban ilyen nem szerepel. Ha azonban az egyensúlyi állandó értékének hőmérsékletfüggése úgy lenne megadva, hogy mondjuk 400 Kelvin fok felett érvényes a programban szereplő formula, alacsonyabb hőmérsékleten viszont egy másik függvényt (kifejezést) használnánk, akkor a K értékét megadó kifejezés a következőképpen alakulna:

K := if T > 400 then (A PROGRAMBAN LEVŐ KIFEJEZÉS)
else (egy másik függvény)

Nemcsak aritmetikai, hanem logikai kifejezések is lehetnek feltételesek, sőt, cimkét is megadhatunk

feltételesen, például, a második típusú feltételes kifejezésnél leírt utasítással teljesen azonos hatású a

go to if i=1 then L1 else L2;

utasítás.

Ezzel arra is mutattunk példát, hogy ugyanazt a célt többféle utasítással is el lehet érni, hiszen ez már a harmadik megoldás ugyanarra a feladatra. Láthatjuk tehát, hogy a programozó bizonyos szabadságot élvez munkája során. Ennek eredményeképpen ugyanarra a feladatra lehet gyorsabb vagy lassabb, áttekinthetőbb vagy komplikáltabb, könnyen, ill. nehezen módosítható programot írni.

Ciklusutasítások

Gyakran előfordult olyan feladat, melyben valamilyen számítást sokszor kell megismételni, úgy hogy egyetlen változó értéke szisztematikusan változik. Ezt ciklusutasításokkal érjük el.

Háromféle ciklusutasítás van:

1. A legegyszerűbb eset a következő példából érthető meg:

```
for x:=0.01, 0.5, 0.75, 1, 2, 10 do
begin
   y:=2×x ↑ 2+1;
   write(≮ddd.dddd≯,y)
end
```

E programrész hatására x értéke rendre felveszi a := és a do alapjelek között álló értékeket, vagyis a 0.01, 0.5, stb értékeket, rendre kiszámítja y értékét a megadott algebrai kifejezéssel (2×x ↑ 2+1), és a (továbbiakban részletezett) write eljárásutasítás segítségével kiírja y értékét 7 jegy pontossággal.

Megjegyezzük, hogy az a lista, melyen x értéke végigfut, nemcsak konstansokból, hanem tetszőleges aritmetikai kifejezésekből is állhat.

2. Ha ki akarjuk számítani n faktoriálisának értékét, azt a következő programrésszel tehetjük meg:

```
i:=1;
for j:=1 step 1 until n do i:=i×j;
```

Miután i értéke felvette a := utáni kifejezés értékét, 1-et (egyszerű értékadás), j értékét egytől a step utáni kifejezéssel, vagyis jelen esetben egyesével lépeti a program addig, amíg i értéke el nem éri az until után álló kifejezés, vagyis esetünkben az n értékét, és minden lépésben i-t megszorozza j-vel (tehát végrehajtja a do után álló utasítást). Így a ciklus végrehajtása után i értéke n faktoriálisa lesz. Ha pl. n=4, akkor a következő négy lépésben jutunk eredményhez:

$1\times1=1$
$1\times2=2$
$2\times3=6$
$6\times4=24$

Ez esetben is igaz az, hogy mind a ciklusváltozó kezdeti értéke, mind a léptetéshez megadott kifejezés, mind pedig a felső korlát tetszőleges aritmetikai kifejezés lehet.

3. A programunkban az egyensúlyi koncentrációkat iterációval számítottuk ki. Az iterációt a program akkor fejezi be, ha az abs(delta)>0.0001 reláció értéke false lett, mert az

```
if abs(delta)>0.0001 then
go to ISM;
```

utasítás ez esetben már nem hajtódik végre, a program tovább megy.

Ezt a feladatot a következő ciklusutasítással is megoldhatjuk:

```
for delta:=K×mCO×mH2O-mCO2×mH2)/
(K×(mCO+mH2O)+mCO2+mH2) while abs
(delta)>10-4 do begin
   mCO:=mCO-delta;
   mH2O:=mH2O+delta;
   mCO2:=mCO2:=mCO2+delta;
   mH2:=mH2+delta;
end delta>0.0001.
```

Ez a ciklusutasítás azt eredményezi, hogy delta felveszi a neki megadott aritmetikai kifejezés értékét, ezután a program kiszámítja a while utáni logikai kifejezést, és ha az értéke true, akkor a program végrehajtja a do utáni utasítást, majd delta értéke ismét felveszi az aritmetikai kifejezés értékét, s. i. t.

Ez általában azért nem egyszerű ismétlés, mert, mint esetünkben is, az aritmetikai kifejezésben szereplő változók értékei változnak a do utáni utasítás hatására.

A három típusú ciklusutasítást kombinálni is lehet, a := és a do alapjelek között vesszővel elválasztva mindhárom ciklustípus előfordulhat, például:

```
for y:=1,2×x+5,3 step 2 until n ↑ 2,x ↑ 2
while y<105 do begin
.....
```

Tömbök

Gyakran fordul elő, hogy egy feladatban sok, azonos jellegű mennyiséggel kell számolni. Tegyük fel, hogy valamilyen számolás során nemcsak n faktoriálisára van szükségünk, amint azt a ciklusutasításnál kiszámítottuk, hanem az 1 és n közötti számok faktoriálisára is. Ha tudjuk, hogy a feladatunkban legfeljebb n=4, akkor egyszerű változókkal is elérhetjük célunkat:

```
f1:=1;
f2:=f1×2;
f3:=f2×3;
f4:=f3×4.
```

A fenti négy utasítás hatására f1, f2,...f4 értékei rendre 1, 2, 3, 4 faktoriálisai lesznek.

Látjuk azonban, hogy ezek az f1, f2, f3, f4 változók lényegében ugyanolyan mennyiségeket jelentenek, és csak a sorszámuk tesz közöttük különbséget.

Az Algol nyelvben lehetőség van arra, hogy egy változó sorozatot közös azonosítóval lássunk el, és az egyes változókra a sorszámokkal hivatkozzunk. Ha a faktoriálisokat 1-től n-ig egy f azonosítójú *tömbben* kívánjuk tárolni, akkor ezt a következő programrésszel érhetjük el:

```
f[1]:=1;
for i:=2 step 1 until n do f[i]:=f[i-1]×i.
```

Látjuk, hogy a tömb azonosítója után szögletes zárójelbe kell tenni azt az aritmetikai kifejezést, amelyik megmondja, hogy a tömb hányadik elemé-ről van szó (mint a kapcsoló elemeknél). Ez a kifejezés a tömb indexe. Ugyancsak látható ezen a példán, hogy a ciklusutasítások használhatóságát a tömbök nagymértékben elősegítik. A második programunkban, ahol egyszerre több adatrendszert olvastunk be (ciklus utasítással), tömbökben tároltuk a beolvasott értékeket.

Egy tömbnek nemcsak egy, hanem több indexe is lehet. Ha pl. egy n. változós lineáris egyenletrendszert kívánunk megoldani, akkor az A együtthatókat célszerű egy kétindexes tömbben tárolni, úgy, hogy az i-ik sor j-ik elemét az A[i, j] elem tartalmazza. (A a tömb azonosítója ebben az esetben.)

Több indexes tömb használatakor az egyes indexek értékét megadó aritmetikai kifejezéseket vesszővel kell elválasztani.

A tömböket a változókhoz hasonlóan deklarálni kell. A deklaráció során meg kell adni, hogy a tömb milyen típusú változókat tartalmaz (egész, valós, ill. logikai), hány indexe van, és az egyes indexek milyen korlátok között változhatnak a programon belül. (Az utóbbira azért van szükség, hogy a program tudja, mekkora helyet kell a memóriában fenntartani az illető tömb számára).

Tömbdeklaráció a következő alakú:

```
begin
    integer array fakt [0:50];
    real array coef1, coef2[1:n, 1:n];
```

A tömbdeklaráló alapjel az array, az előtte szereplő alapjel a tömb típusát adja meg. Ha csak az array alapjelet írjuk le, az real array-t jelent. Ezután a tömb azonosítóját kell leírni, majd utána szögletes zárójelbe az alsó és a felső korlátokat, kettőspontal elválasztva: [0:50]

A fakt tömbben tehát tárolhatjuk 0, 1, 2,... 49, 50 faktoriálisait.

A második példában két kétindexes tömböt deklaráltunk, mindkettő azonos méretű, ezért a méretet csak egyszer kellett megadni. Látjuk, hogy ebben az esetben a tömb mérete egy változó értékétől függ. Fontos, hogy ennek a változónak már a deklarálása előtt adjon a program értéket, mert ellenkező esetben a program ezt a véletlen bitkombinációt tekintené a tömb kívánt méretének, mely az n változó tárolására a fordítóprogram által kijelölt helyen van, ami akár $-_{10}58$ is lehet. (Ez a megjegyzés persze nemcsak a deklarációknál használt változókra érvényes, hanem bármely egyéb utasításban szereplőre is.)

Eljárások

Az eddigiekben is találkoztunk már olyan esettel, amikor ugyanaz a programrész többször működik, például a ciklusoknál.

A második programunkban egy ennél valamivel általánosabb algoritmust kell két különböző módon végrehajtani, különböző bemenő adatokkal — az egyensúlyi koncentrációk számítását. Ahhoz, hogy ne kelljen ugyanazt a programrészt kétszer leírni, vagy esetleg különböző feltételes utasításokkal visszavezérelni a programot a már egyszer végrehajtott részre, *eljárást* kell használni.

Az eljárás olyan blokk, melynek van egy azonosítója, és lehetnek *formális paraméterei*. Az eljárás formális paraméterei olyan azonosítók, melyek csak blokk leírása (azaz a deklarációja) során szerepelnek. Amikor a blokkot végre kívánjuk hajtani a programmal, akkor meg kell adni, hogy az egyes formális paraméterek helyére milyen konkrét változó vagy kifejezés kerüljön. Az eljárások használata nagy flexibilitást jelent, különböző változókkal, mint bemenő adatokkal hajtathatjuk végre a blokkot, a kimenő adatokat is tetszőleges változóknak adhatjuk át. A második programban az egyensúlyi koncentrációkat számító eljárás deklarációja a következő (31. sor).

```
procedure EGYENSÚLY (K, mCO, mCO2, mH2, mH2O);
    value K;
    real K, mCO, CO2, mH2, mH2O;
    begin
```

Az eljárás deklaráció alapjele, mint látjuk a procedure. Ennek az eljárásnak EGYENSÚLY az azonosítója. Az azonosító után zárójelbe zárva, vesszővel elválasztva a formális paraméterek listája következik:

(K, mCO, mCO2, mH2, mH2O)

A deklaráció során meg kell adni, hogy az eljárás paraméterei milyen típusúak, hiszen ezek nem deklarált változók, és értelmük csak az eljárásdeklaráción belül van. Ezt az információt a specifikáció adja, mely formailag a deklarációra emlékeztet, de nem szabad vele összetéveszteni:

real K, mCO, mCO2, mH2, mH2O;

Ez azt jelenti, hogy az eljárás valamennyi paramétere egyszerű valós mennyiség.

A value K; információ külön jelentéssel bír: azt jelenti, hogy a K paramétert *értékkel hívtuk*.

Ha valamelyik paramétert értékkel hívjuk, akkor az eljárásba való belépéskor a program deklarál magának egy változót, annak értéke felveszi az értékkel hívott változó (esetünkben K) aktuális értékét, és a továbbiakban az eljárás során a program ezzel számol tovább, ezt a változót viszont amelyet K helyére az eljárás hívásakor beírtunk, nem változtatja. Ha tehát a programunkban az EGYENSÚLY eljárásban pl. az mCO2 formális paramétert is értékkel hívtuk volna, akkor az eljárás ugyan kiszámolta volna az egyensúlyi CO2 koncentrációt, de MCO2 értéke (amit az eljárás első hívásakor mCO2 helyére írtunk, tehát azt akartuk, hogy az MCO2 változóval hajtódjanak végre azok az utasítások melyeket a deklaráció során mCO2-re előírtunk), a beolvasott, kiindulási érték marad.

Felmerül az a kérdés, hogy milyen típusú mennyiségek szerepelhetnek eljárás paramétereként: Minden, ami deklarálható, szerepelhet eljárás paramétereként, (tehát szerepelhet akár eljárás is). Ezenkívül megadhatunk címkét (label) és szöveget (string) is.

Szövegen tetszőleges jelsorozatot értünk, amelyet speciális zárójel közé kell tenni. A Gier-Algol a

⪡< nyitó és ⪢ záró zárójelet használja e célra. Mivel az Algol nyelvben nincs olyan egyszerű utasítás, amelyben szövegek szerepelnek, szöveg paramétert közvetlenül csak gépi kódban írt eljárás tud kezelni.

Függvény eljárások

Ha egy eljárásnál a procedure alapjel elé a real, integer vagy boolean azonosítót írjuk, akkor függvényeljárást deklarálunk. A függvényeljárás az eljárástól abban különbözik, hogy az eljárásnak is van értéke, tehát szerepelhet kifejezésekben egyszerű változók helyén. Ily módon tetszőleges függvényeket is bevehetünk az aritmetikai kifejezéseinkbe. A következő részben erre is látunk majd példát.

Standard eljárások

Standard eljárás olyan eljárás, amit nem kell deklarálni, mert már a fordítóprogramba be van építve. Ilyenek a leggyakrabban használt matematikai függvények (standard eljárás függvényeljárás is lehet) pl. az ln, exp, sin, cos, arctan, abs, entier (valamennyien egyparaméteres eljárások, értékük rendre a paraméterként kapott érték természetes logaritmusa, természetes alapú exponenciálisa, sinusa, cosinusa, arcustangensének főértéke, abszolútértéke, egész része). Pl. a

K:=exp(−0.7685×ln(T)+((((1.4752$_{10}$−10×
T×−9.6605$_{10}$−7)×T+3.0102$_{10}$−3)×T1.5065)×
×T+4.9433$_{10}$3)/T);

utasításban aritmetikai kifejezésben szerepel az ln és az exp standard függvényeljárás. (Ez egyben példa arra is, hogyan szerepel függvényeljárás aritmetikai kifejezésben.)

A most felsorolt standard függvényeljárások valamennyi Algol fordítóprogramba be vannak építve

Input—Output

Az inputot és outputot ugyancsak standard eljárások végzik, ezek azonban már gépről gépre változnak.

Példaképpen ismertetünk néhány, programunkban előforduló Gier-Algol4 standard eljárást, olymódon, hogy vázlatosan leírjuk azt az eljárás deklarációt, amelyikkel deklarálni kellene az illető eljárást.

integer procedure select(k);
value k;
integer k;
begin

select:=Előző select utasítás paraméterének értéke; A beolvasó és kiíró egységet (perifériát) k értékétől függően választja ki (szelektálja) a program, a select(17) utasítás például írógép inputot és outputot jelent;

end select;
real procedure real read;
begin

read real:=A választott perifériáról az utasítás végrehajtásakor beolvasott szám,

end read real;

például

mCO:=read real;

Ebből a példából az is látható, hogy függvényeljárás deklarációnál szerepelnie kell egy olyan értékadó utasításnak, melynek bal oldalán az eljárás azonosítója áll.

A programban a select(17) utasítás önmagában áll, noha a select eljárás függvényeljárás. Általános szabály az, hogy függvényeljárást lehet úgy használni, mint egy eljárást, ebben az esetben az eljárás értékét ugyan kiszámítja a program, de nem őrzi meg.

procedure writetext (TEXT);
string TEXT;
begin

A program a választott perifériára kiírja a TEXT helyén megadott szöveget.

end;

Például a

writetext ⪡<
HIBÁS INPUT ADATOT KAPTAM;
⪢);

utasítás hatására a hibaüzemet kerül kiírásra.

A writecr; eljárás hívás egyenértékű egy

writetext ⪡<
⪢);

utasítással, azaz kocsivisszát csinál a kiíró egységen.

Az egyes számokat kiíró eljárás, a write eljárás, valamivel bonyolultabb, egyrészt tetszőleges számú paramétere lehet (Ha egyszer egy eljárást *mi* deklaráltunk Algolban, annak mindig ugyanannyi paramétere kell hogy legyen, mint ahány formális paramétere volt), másrészt a kiírási formátumot egy speciálisan adott logikai változóval lehet megadni. Például a

write (⪡−d.dddd⪢, mCO, mCO2, mH2, mG2O);

utasítás hatására a program kiíratja mCO, mCO2, mH2, mH2O értékeit 5 tizedes pontossággal.

Záró megjegyzés

Vázlatosan ismertettük az Algol nyelv alapfogalmait. Most már a Programunkban minden utasítást értelmezni tudunk, javasoljuk az olvasónak, hogy ezt tegye is meg. Ebben segíteni fogják az olvasót a mindkét programban sűrűn előforduló kommentek.

Szeretnénk mégegyszer hangsúlyozni, hogy nem programozni akartuk tanítani az olvasót (erre a célra több, cikkünknél lényegesen részletesebb könyv, kiadvány jelent meg), hanem csupán annyit szerettünk volna elérni, hogy az olvasó meg tudjon érteni egy Algol nyelven írt programot (amire sorozatunk további tagjaiban többször szüksége lesz).

FÜGGELÉK

1. változat programja

begin
comment VÍZGÁZ EGYENSÚLY SZÁMÍTÓ PROGRAM, 1. változat;

```
    real t, delta, mCO, mCO2, mH2O, mHX, MIn;
    comment t a hőmérséklet Celsius fokokban,
    delta a móltört változás az iteráció során, MIn
    az inert gázok móltörtje, mCO, mCO2, mH2,
    mH2O pedig a szénmonoxid, széndioxid, hid-
    rogén, ill. víz móltörtjei;
    select(17);
  comment Írógép input-output;
    INPUT:
    writetext(*<
Homerseklet: *);
  t:=read integer;
  comment Beolvassuk a hőmérséklet értékét;
  writetext (*<
CO, H2O, H2, CO2 moltortek: *);
  mCO:=read real;
  mH2O:=read real;
  mH2:=read real;
  mCO2:=read real;
  comment Beolvastuk a kiindulási móltörteket;
  begin
    comment Ebben a blokkban végezzük el a
    tulajdonképpeni számolást;
    real T, K;
    comment T lesz az abszolút hőmérséklet, K
    pedig az egyensúlyi állandó;
    T:=t+273.16;
    comment Átszámítottuk a hőmérsékletet Cel-
    siusról Kelvin fokra;
    K:=exp(-0.7685+ln(T)+((((1.4752_10-
    -10×T
    -9.6605_10-7)×T+3.0102_10-3)×T-1.5065)
    ×T+4.9433_10 3)/T);
    comment Kiszámítottuk az egyensúlyi állandó
    értékét; MIn:=1-mCO-mH2O-mCO2-
    -mH2;
    comment MIn értéke felvette az inert gázok
    móltörtjének értékét;
    if MIn<0∨MIm>1∨T<0 then
    begin
      comment Valamelyik adat fizikailag értel-
      metlen, ezért a program új adatokat kér;
      writetext(*<
HIBÁS INPUT ADATOKAT KAPTAM
*);
      go to INPUT;
      comment A következő végrehajtott utasítás a
      t:=read real lesz;
    end;
    ISM:
    delta:=(K×mCO×mH2O-mCO2×mH2)/
    (K×(mCO+mH2O)+mCO2+mH2);
    mCO:=mCO-delta;
    mH2O:=mH2O-delta;
    mCO2:=mCO2+delta;
    mH2:=mH2+delta;
    comment Kiszámítottuk a móltörtváltoztatás
    értékét (delta), és ezzel korrigáltuk a móltörte-
    ket;
    if abs(delta)>0.0001 then go to ISM;
    comment Megvizsgáltuk delta értékét. Ha ez
    még nagyobb 0.0001-nél, akkor az iteráció
    folytatódik azaz a következő utasítás megint
    a delta:= ... lesz;
  end számoló blokk;
  writetext(*<
Az egyensúlyi móltörtek:
  CO     CO2     J2     H2O
  *);
  comment Az eredményközlés előtt fejlécet írtunk
  ki, hogy az adatok könnyen értelmezhetők legye-
  nek;
  write(<-d.ddddd<, mCO, mCO2, mH2,
  mH2O);
  comment Kiírtuk mCO, mCO2, mH2, mH2O ér-
  tékeit, melyek most már nem az eredeti, hanem
  az egyensúlyi állapotot írják le;
  go to INPUT;
  comment A program új adatokat fog kérni;
end
program.
```

2. változat programja

```
begin
  comment VÍZGÁZ EGYENSÚLY PROGRAM.
  2. változat;
  real t, MCO, MCO2, MH2, MH2O;
  integer i, n;
  real procedure K(t);
  value t;
  real t;
  begin
    comment Az eljárás a K egyensúlyi állandót
    számítja t hőmérsékleten;
    real T;
    T:=t+273.16;
    comment Átszámítottuk Celsius fokról Kelvin
    fokra a hőmérsékletet;
    K:=exp(-0.7685×ln(T)+(((1.4752_10-10×T
    -9.6605_10-7)×T+3.0102_10-3)×T-1.5065)
    ×T+4.9433_10 3)/T);
  end K;
  procedure KIIR (a, b, c, d);
  value a, b, c, d;
  real a, b, c, d;
  begin
    comment Ez az eljárás a négy móltörtet írja ki,
    fejléccel ellátva;
    writetext(*<
    CO     CO2     H2     H2O
*);
    write (<,,-d.dddd*, a, b, c, d);
    writecr;
  end KIIR;
  procedure EGYENSÚLY (K, mCO, mCO2,
  mH2, mH2O);
  value K;
  real K, mCO, mCO2, mH2, mH2O;
  begin
    comment Az EGYENSÚLY eljárás az egyen-
```

```
súlyi móltörteket számítja ki;
real Min, delta;
Min:=1-mCO-mCO2-mH2-mH2O;
comment Min értéke a gázelegyben levő közömbös (inert) gázok móltörtje lesz;
if Min<0∨Min≥1 ∨ K≤0 then
begin
    select(17);
    comment Irógép input-output;
    writetext(*<
VALAMELYIK INPUT ADAT HIBÁS:*);
    KIIR(mCO, mCO2, mH2, mH2O);
    comment Amennyiben Min vagy K értéke fizikailag értelmetlen volna, a program hibajelzést ad;
    goto KAPCS[3];
    comment A program befejezi a futást, ugyanis ez az utasítás a goto END utasítással egyenértékű.;
end hibajelzés;
for delta:=(K×mCO×mH2O-mCO2×mH2)/
(K×(mCO+mH2O)+mCO2+mH2)
while abs (delta)>10-4 do
begin
    comment abs (delta) értéke delta abszolút értéke;
    mCO:=mCO-delta;
    mCO2:=mCO2+delta;
    mH2:=mH2+delta;
    mH2O:=mH2O-delta;
    comment Valahányszor a program a ciklust végrehajtja,
      a móltörteket a ciklus elején kiszámított deltával korrigálja;
  end delta ciklus;
end EGYENSÚLY;
switch KAPCS:=L1, L2, END;
comment Hangsúlyozni szeretnénk, hogy a deklarációk sorrendje tetszőleges, a program ugyanúgy dolgozna, ha a begin után először az EGYENSÚLY eljárást, vagy akár a KAPCS kapcsolót deklaráltuk volna;
INPUT:
select(17);
writetext(*<
A futás módja: <);
comment Ha az írógépen beírt szám 1, akkor még beolvassa az adatrendszerek számát az írógépről (select(17) utasítással kezdtük a programot), majd annyi adatrendszert olvas be szalagról és azokat feldolgozza sorban, ha viszont 2, akkor az írógépről csak egy koncentráció adatrendszert olvas be, és 400, 425, 450, 475, 500 Celsius fok mellett kiszámítja az egyensúlyi elegy koncentrációit.
i:=read integer;
if i≒1 < i≒2 then goto INPUT;
comment ÉRTELMETLEN SZÁM BEÍRÁSA ESETÉN ISMÉTLI A BEOLVASÁST;
goto KAPCS[i];
comment A program futást i értékétől függően az L1, ill. L2 címkénél folytatja;
L1:
writetext(*<
Adatrendszerek száma: *);
n:=read integer;
if n<1 then goto L1;
begin
  array MCO, MH2, MCO2, MH2O, t[1:n];
  comment Új blokkot nyitottunk, ahol deklaráltunk négy n elemű tömböt. Ezeknek az azonosítói ugyanazok, amelyek a külső blokkban valós változóknak vannak deklarálva, ezért ebben a blokkban azokhoz a változókhoz nem tudunk nyúlni (nem is akarunk). A blokkot lezáró end után viszont ismét visszanyerik az azonosítók az eredeti értelmüket.;
  select(8);
  comment Sornyomtató output, szalag input;
  for i:=1 step 1 until n do
  begin
    MCO[i]:=read real;
    MCO2[i]:=read real;
    MH2[i]:=read real;
    MH2O[i]:=read real;
    t[i]:=read real;
    comment A ciklus minden lépése egy adatrendszert olvas be a megfelelő töbökbe.;
  end i;
  for i:=1 step 1 until n do
  begin
    comment Ebben a ciklusban egymásután feldolgozzuk a beolvasott adatokat;
    printtext(*<
A kiindulási adatok:
A hőmérséklet Celsius fokokban: *);
    comment A printtext eljárás funkciója lényegében a writetext-évl azonos, de sornyomtatóra való írás esetén az ékezeketek is megfelelően tudja kezelni;
    write (*<dddd.dd*>, t[i]);
    KIIR(MCO[i], MCO2[i], MH2[i], MH2O[i]);
    comment Visszaírtuk a beolvasott adatokat;
    EGYENSÚLY(K(t[i]), MCO[i], MCO2[i],
    MH2[i], MH2O[i]);
    comment Látjuk, hogy az egyensúlyi állandót jelölő formális paraméter helyére a K(t[i]) függvényeljárás hívást írtuk. Mivel a K paramétert az EGYENSÚLY eljárás értékkel hívja, a program csak az EGYENSÚLY eljárás hívásakor fogja a K(t[i]) függvényt kiszámítani. Ha értékkel hívta volna az EGYENSÚLY eljárás a K paraméterét, akkor a program az EGYENSÚLY eljárás futása során K értékét mindig kiszámította volna, valahányszor a K azonosító benne előfordul. Ez hibás eredményre ugyan nem vezetett volna, csupán a számítási időt növelte volna meg.;
    printtext(*<
```

```
Az egyensúlyi móltörtek:≯);
    KIIR(MCO[i], MCO2[i], MH2[i], MH2O[i]);
    comment Kiírtuk az eredményül kapott mól-
    törteket;
  end i;
  end Tömbdeklarációk blokkja;
  goto END;
  L2:
  writetext(≮<
CO, CO2, H2, H2O móltörtekE ≯);
  MCO:=read real;
  MCO2:=read real;
  MH2:=read real;
  MH2O:=read real;
  comment Beolvastuk a móltörteket az írógép-
  ről;
  select(8);
  printtext(≮<
A kiindulási móltörtek:≯);
  KIIR(MCO, MCO2, MH2, MH2O);
  for t:=400 step 25 until 500 do
  begin
    comment Ebben a ciklusban a különböző hő-
    mérsékletekre számolunk;
    writetext(≮<
A hőmérséklet:≮);
    write(≮dddd.dd≯,t);
    EGYENSÚLY(K(t), MCO, MCO2, MH2,
    MH2O);
    comment Mint látjuk, a móltörteket nem ál-
    lítjuk vissza minden számítás előtt az induló
    értékre, hanem az előző számítás eredményből
    számolunk tovább. Az eredménynek azonban
    ugyanolyannak kell lennie, mintha az eredeti
    értékekből indultunk volna ki, hiszen a szén,
    hidrogén és oxigén tartalma az elegynek min-
    dig ugyanaz. Ugyanakkor viszont az iteráció
    során kevesebb műveletre lesz szükségünk, ha
    így számolunk, mert a kiinduló állapot az
    egyensúlyihoz közel van.
    Ezzel ismét láttunk egy példát a számítás-
    technikai gyakorlatban gyakran előforduló fo-
    gásokra, melyek megfelelő használatával a
    végrehajtandó műveletek számát jelentősen le
    tudjuk csökkenteni, ami rövidíti a futási időt
    és növeli a számítás megbízhatóságát;
    printtext(≮<
Az egyensúlyi móltörtek: ≯);
    KIIR (MCO, MCO2, MH2, MH2O);
  end t;
  END:
end
program;
```


IRODALOM

[1] *Esztergár Zs.—Mezei M.:* Magyar Kém. Lapja 28,99 (1973).

РЕЗЮМЕ

В сообщении с описан вариант программного языка АЛГОЛ 60, разработанного для машины ГИЕР. В качестве примера приводится решена задачи, фигурирующей в 11 сообщении.

SUMMARY

The Gier version of the Algol 60 programming language is presented and to illustrate its application the solution of a problem which will be discussed in the preceeding paper is given.