Programozási tételek szerepe a gyakorlati programozásban

Szlávi Péter

ELTE TTK Informatika Szakmódszertani Csoport, 1996

Programozási tételek szerepe a gyakorlati programozásban

Gondolatok egy feladat megoldása kapcsán

Egy konkrét feladat megoldása közben arról töprengünk, hogy miként tehető mechanikussá a programkészítés tevékenysége a biztonságosság, helyesség megőrzése mellett. Más oldalról megközelítve a kérdést: mi a szerepe a lexikális ismereteknek a programozásban? Megint másként: rutinszerű tevékenység vagy alkotás (már amennyire szembeállíthatók ezek egymással) a programírás?

Három lépésen (gondolatsoron) keresztül jutunk el a feladatból a megoldó program kódjáig. Azt vizsgáljuk, hogy a három lépés: a feladatspecifikáció, az algoritmizálás és a kódolás hogyan kapcsolódnak egymáshoz. Igyekszünk beláttatni, hogy az első lépés a leginkább „kreatív" lépés, amelynek jó vagy rossz döntései tükröződnek az algoritmuson és a kódon is. Mind az algoritmizálás, mind a kódolás nagyban mechanizálható lépések láncolatából tehető össze.

Az első gondolatsor „mottója" ez lehet: mi a tételek szerepe a specifikáció elkészítésekor? A másodiké: hogyan kapcsolódnak össze a specifikáció egyes részei az algoritmus egyes részleteivel? S végül a harmadiké: milyen információk használhatók föl kódolás során az algoritmusból és az algoritmusból?

0. BEVEZETÉS

Írásomban egy programozási feladat megoldása kapcsán hangosan gondolkodom ilyesfajta kérdéseken, mint „Mennyire tehető mechanikussá a programkészítés?", vagy „Mi a szerepe a lexikális ismereteknek a programozásban?", ill. „Rutin vagy alkotás a programírás?"

Azt gondolom, ezek a kérdések nagyban befolyásolják a programozás oktatás „filozófiáját". Hisz egy tevékenység rutinszerűen elvégezhető lépéseit lehet/kell gyakoroltatni, egy ismeretnek lexikális részei „bemagoltatható" még a legbigottabb számítógép-kerülő tanulóval is; mindezt persze azért, hogy mindenki birtokába juthasson azoknak az eszközöknek, amelyekkel azután szárnyalni, alkotni képes.

Nos, az alábbiakból –remélem– kiolvasható lesz, hogy igen nagy részben mechanizálható lépések sorozatából áll a programozás, amelyek –megfelelően leegyszerűsített formalizmus esetén– korán taníthatók minden, „számítógéppel nem fertőzött gondolkodású" gyermekkel.

Tudom, hogy e gondolatokat megfogalmazni mai, túlracionalizált „felfedezés orientált" felfogású világunkban szinte *szent*ségtörés, de mivel *szent*ül meg vagyok győződve igazamról, leírom.

Előzetesen jeleznem kell, hogy az ELTE Informatika Tanár Szakán szokásosnak mondható felfogást követem, sok dolgot nem magyarázok meg, hanem utalásokkal intézem el. Például nem magyarázom a specifikáció részeit, mit jelentenek a formalizmusok, ugyancsak magyarázatlanul maradnak az algoritmikus nyelv elemei, ill. a célnyelvnek tekintett Turbo Pascal szintaxisa, szemantikája, ismertnek tételezem föl a programozási tételek fogalmát, sőt

magukat a tételeket is. Ezen ismereteket az ajánlott irodalmakból össze lehet szedni. A függelékben összegyűjtöttük a feladat megoldása során „alapvetőként” fölhasznált ismereteket. (Tételspecifikációkat, -algoritmusokat stb.)

1. A FELADAT

Adott egy repülőgépről fölvett magassági méréssorozat. A repülő Európából indult, ahol az első mérést megtette, és Amerikában szállt le, ahol szintén egy szárazföldi méréssel fejezte be útját. Tudjuk, hogy legalább egy mérést a tenger felett végzett. Ott, ahol tenger van (s csak ott) a mérés 0-nak adódott. Gyűjtsük ki, azon mérés-sorszámokat, ahol sziget kezdődött, ill. ahol sziget fejeződött be.

A feladat „eredeti” szövege ennyi. Mint általában e leírás titkokat rejt, nem egyértelmű. De hát az Élet is ilyen. A feladatmegoldó első dolga ezeket az „információs hézagokat/ disszonanciákat” betömni/feltárni. Ezt végezzük el a specifikáció elkészítése során.

2. AZ ELSŐ GONDOLATSOR

a feladat és a feladatspecifikáció

Tisztázandó tehát, milyen információk állnak rendelkezésünkre, amikből ki lehet indulni (*Bemenet*), ill. miféle információk, amiket a programnak szolgáltatnia kell (*Kimenet*). Ennek megfogalmazása során érdemes *csak néhány alaphalmaz*t fölhasználni, mégpedig *olyénhány*at, amelyeknek könnyen megfeleltethetünk egy, az algoritmikus nyelvben előforduló típust.[1] Ezek lehetnek: Z (egész számok), N (természetes számok), R (valós számok), Szöveg, Karakter, L (logikai értékek).

A rendelkezésre álló információk nemcsak adatok lehetnek, hanem valami, a feladat szempontjából kulcsfontosságú „adatosztályozó” elv: *tulajdonság*. Így célszerű a bemeneti részben megemlíteni az ilyen információkat hordozó logikai értékű függvényeket.

Az előrelátható, hogy az adatok és a függvények a leírás további részeiben is szerepelnek majd, ezért nevet kell hozzájuk rendelni.

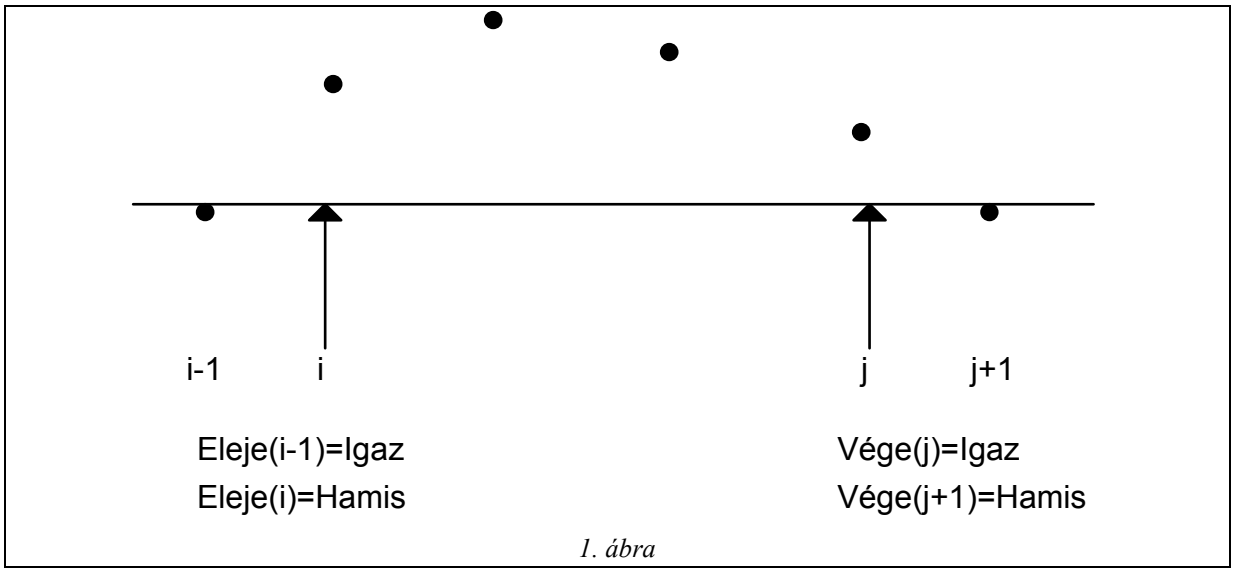
Tehát a konkrét esetben:

Bemenet: $N \in N$, $Mag \in R^N$, $Eleje: N \to L$,

$Eleje(z) := Mag_{z-1}=0$ és $Mag_z>0$, $Vége: N \to L$, $Vége(z) := Mag_z>0$ és $Mag_{z+1}=0$

Magyarázatra talán csak a két tulajdonság függvénye szorul. Ez is egy ábrával kellően illusztrálható. (L. 1. ábrát.)

[1] Típus = értékhalmaz + műveletek együttese.



1. ábra

Az eredmény miféle? A feladat szerint egy adatpárokból álló sorozatot várunk, ami „rossz" esetben nullahosszúságú. A leírásban nehézséget legfeljebb a „párság" okozhat. A „tisztességes" megoldás megkapható a halmazok között alkalmazható *direktszorzat* segítségével; bár –most– megkerülhető: önálló sorozatokként való föltüntetésével. Azaz

Kimenet: $Db \in N$, $Sziget \in (Első \times Utolsó)^{Db}$, $Első, Utolsó = N$

vagy

Kimenet: $Db \in N$, $Első, Utolsó \in N^{Db}$

A két leírás szemlélete más, ami a későbbi lépésekben (algoritmus és kód) is tükröződni fog.

Ez az a pillanat, amikor már érdemes kacsingatnunk a megoldás során felhasználható tételek garnitúrája felé.[2] Ugyanis, ha megsejtjük, hogy mely tétel alkalmazható a feladatra, akkor ezt az információt fölhasználhatjuk fogódzóként a specifikáció további részeinek elkészítéséhez, és természetesen az algoritmizálást már –úgy'szólván– mechanikusan végezhetjük el.

A leggyakrabban használt tételeinket az alábbi kategóriákba csoportosíthatjuk. A csoportosítás alapja, hogy –mint egy leképezésnek tekintve– mi az „értelmezési tartománya" (bemenet) és mi az „értékkészlete" (kimenet). Eszerint a csoportok:

a. sorozat[3] $\longrightarrow$ érték

b. sorozat $\longrightarrow$ sorozat

c. sorozat $\longrightarrow$ sorozatok

[2] Az F1 függelékben fölsoroljuk a „bevethető" tételeket.

[3] *Sorozat (S)*: egy alaphalmaz (H) *iteráltjá*nak (H^*), azaz véges számú alaphalmazbeli elemeknek, együttese; ennek i. elemére s_i-vel hivatkozunk.

d. sorozatok $\longrightarrow$ sorozat

Látszik, hogy ez az osztályozás „ránézésre” elvégezhető, s eredményeként a közel 20 tétel helyett már csak 4-6 tételt kell alaposabban szemügyre venni. (Például az a. esetben az „érték” mifélesége teljesen egyértelműen eligazít.) Esetünkben a b. csoport jön szóba, illetve az alternatív kimenetet vizsgálva a c. csoportra gondolhatunk. Az elsőt vizsgáljuk az alábbi A.-val jelölt részben, az utóbbit a B.-vel jelöltben.

A. esetben:

A szóba jöhető tételek: *másolás*, *kiválogatás*, *rendezés*(ek). Pusztán formális meggondolással kizárható a másolás és a rendezés (hisz a kimeneti sorozatok hossza ezeknél a bemenetiével feltétlenül megegyező, ami most nem igaz). A megmaradt egyetlen *kiválogatás* tétellel egy aprócska probléma van: az eredetiben egy tulajdonság helyett itt kettő is van. Ebből arra következhetünk, hogy a feladat valójában nem egy, hanem kettő kiválogatás tételt rejt. De azt már is konstatálnunk kell, hogy a két kiválogatás eredmény sorozatai azonos (Db) hosszúságúak.

A kiválogatás tétel specifikációját mintaként használjuk az előfeltétel és utófeltétel összeállításához. Az előfeltétel írja le (logikai formulaként[4]), hogy milyen szűkítő feltételek érvényesek a feladat szerint a bemeneti adatokra az előzőkben definiálthoz képest; jelen esetben N-re (N-hez képest), Mag-ra (R^N-hez képest). A sablonként alkalmazott tétel előfeltétele „üres” (azaz *azonosan igaz*), a feladat ezen szűkít: $N\geq 3$, hiszen azt állítja, hogy az első és az utolsó mérés pozitív, sőt legalább egy a tengerbeli; következésképpen legalább ennyi mérés volt. Az R^N helyett $R_+\cup\{0\}$ az alkalmazandó. Vagyis:

Előfeltétel: $N\geq 3$ és $\forall i\in[1,N]: Mag_i\geq 0$ és $Mag_1, Mag_N>0$ és $\exists i\in[2,N]: Mag_i=0$

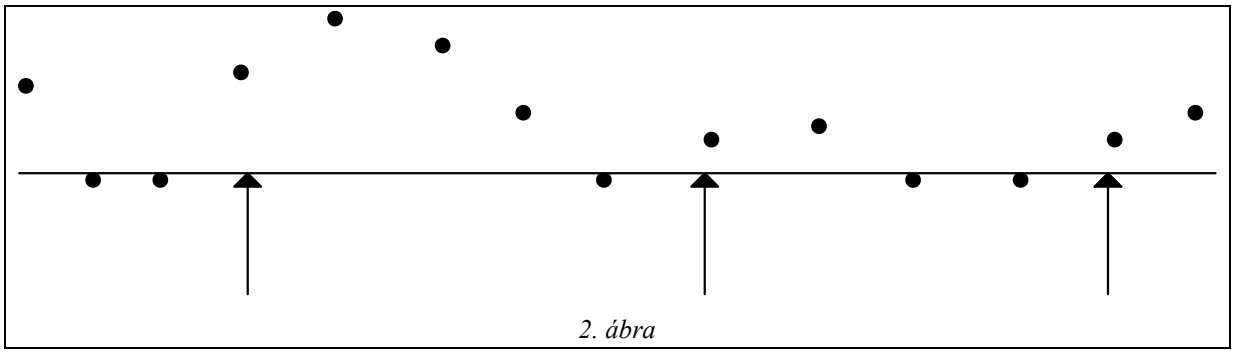
Az utófeltétel összeállításánál jelentősebb segítséget ad a tétel. A sablon szerint Db-re kell mondanunk valamit: „éppen akkora ..., ahány olyan ...”, és az eredmény sorozat elemeiről: „éppen azok indexei, amelyek olyanok ...”. Ahogy azt előbb megsejtettük, itt két kiválogatás van, amelyek egyikéhez az *Eleje*-tulajdonság és a *Mag* sorozat tartozik, a másikhoz a *Vége*-tulajdonság és az *Utolsó* komponens.

Ebből arra gondolunk, hogy az utófeltételnek két hasonló része lesz: az egyikben az egyik kiválogatás, a másikban a másik „mondanivalója”. Így kapjuk:

Utófeltétel: $Db=\Sigma_{(i=2..N)}\ \chi_{Eleje}(i)$ [5] és
($\forall i\in[2,N]$: Eleje(Sziget.Első$_i$) [6] és HalmazFölsorolás(Sziget.Első) [7] és
($\forall i\in[2,N]$: Vége(Sziget.Utolsó$_i$) és HalmazFölsorolás(Sziget.Utolsó)

[4] a *predikátum kalkulus* eszköztárának fölhasználásával
[5] $\chi_{Eleje}(z):=1$, ha Eleje(z)=Igaz; $\chi_{Eleje}(z):=0$, ha Eleje(z)=Hamis
[6] *Sziget.Első* jelölésről: a Sziget elempár-sorozat Első-komponenseiből álló sorozat.

A feladathoz egy „átlagos” állapotot tükröző ábrát készítve némi rossz sejtelmünk ébredhet az utófeltételt illetően. (L. 2. ábrát!)



2. ábra

Az utófeltétel szerint olyan mérés, amely teljesíti az Eleje predikátumot három van, jól lehet sziget csak kettő. Hol a hiba? Válasz: az Eleje predikátumot leegyszerűsítettük két pont vizsgálatára, s nem foglalkoztunk azzal, hogy ezek sziget pontjai valójában vagy sem. A probléma megoldása lehet: 1) *kiegészíteni* a hiányolt vizsgálattal, de ez –előreláthatólag– meglehetősen hosszadalmassá tenné az eldöntést, vagy 2) a predikátumot *korlátozzuk* csak olyan szakaszra, ahol nem keveredhet a szigetek közé kontinens. Azaz, ahol az index a [2,N]-ben „fut” (a Σ-ban, illetve a $\forall i\in[2,N]$ hatáskörben) az utófeltételben cseréljük ki az alábbira: [e,u]-ra, ahol

(*) $e,u\in[1,N]: \forall i\in[1,e-1]\cup[u+1,N]: Mag_i>0$

Ezzel a kiegészítéssel a végső változat így alakul:

Utófeltétel: $e,u\in[1,N]: \forall i\in[1,e-1]\cup[u+1,N]: Mag_i>0$ és
$Db=\Sigma_{(i=e..u)}\ \chi_{Eleje}(i)$ [8] és
$(\forall i\in[e,u]:$ Eleje(Sziget.Első$_i$) [9] és HalmazFölsorolás(Sziget.Első) [10] és
$(\forall i\in[e,u]:$ Vége(Sziget.Utolsó$_i$) és HalmazFölsorolás(Sziget.Utolsó)

B. esetben:

A szóba hozható tétel mindössze egy: a *szétválogatás.* Az elő- és utófeltétel összeszedését már nem részletezzük. Az előfeltétel ugyanaz lesz, mint az előbbi. Az utófeltétel is azonos, a különbség a „levezetésben” lesz, ui. most nem kell rájönni arra, hogy két tétel bújik meg benne.

[7] $z=(z_1,...,z_N)$: *HalmazFölsorolás*(Z)=Igaz, ha $\forall i,j\in[1,N]: z_i\neq z_j$; *HalmazFölsorolás*(Z)=Hamis, máskülönben.
[8] $\chi_{Eleje}(z):=1$, ha Eleje(z)=Igaz; $\chi_{Eleje}(z):=0$, ha Eleje(z)=Hamis
[9] *Sziget.Első* jelölésről: a Sziget elempár-sorozat Első-komponenseiből álló sorozat.
[10] $z=(z_1,...,z_N)$: *HalmazFölsorolás*(Z)=Igaz, ha $\forall i,j\in[1,N]: z_i\neq z_j$; *HalmazFölsorolás*(Z)=Hamis, máskülönben.

3. A MÁSODIK GONDOLATSOR

a specifikáció és az algoritmus

Túlvagyunk a dolog legkreatívabb lépésén. Az elkövetkezőkben a specifikációból és a megsejtett tételből kiindulva a megoldó algoritmust „generáljuk”. Az algoritmuskészítés során csak a lényegi részre koncentrálunk; s nem foglalkozunk pl. az adatok beolvasásával és az eredmény megjelenítésével (amik természetesen nélkülözhetetlenek lennének a megoldó programhoz, de ezeket majd a kódolás során fogjuk meggondolni).

A be- és a kimenet figyelembevétele

A be- és kimeneti részből származtatjuk a leendő programunk globális deklarációs részét. Alkalmazzuk a következő „halmaz → típus” egymáshozrendelési konvenciót: N, Z→Egész; L→Logikai; R→Valós; ...

A halmazokon végzett műveletekkel képzett összetett adathalmazokhoz egy-egy új típust rendelünk, ahogy ezt az alábbi példa mutatója:

```
valami∈AxB, ahol A=C, B=D → Típus ABTip=Rekord(A:CTip, B:DTip)
```

Értelemszerűen a CTip, DTip a C, D halmazok típusmegfelelőit jelölik[11]. Egy másik tipikus példa:

```
valami∈A^N → Típus ANTömbTip=Tömb(1..N:ATip)
```
[12]

Abból indulunk ki, hogy a bemenő adatok értékeit „meg kell szerezni” – mégha ezt nem is algoritmizáljuk–, ezért a bemeneti adatobjektumokat *változó*ként deklaráljuk az algoritmusban. Így:

```
Változó N:Egész
Típus   ValósNTömb=Tömb(1..N:Valós)
Változó Mag:ValósNTömb
```

A specifikációban szereplő függvények definícióját is itt helyezzük el az algoritmikus nyelvi szabályok szerint, amihez minden definíciós „kellék” megvan:

```
Függvény Eleje(z:Egész):Logikai
  Eleje:=Mag(z-1)=0 és Mag(z)>0
Függvény vége.

Függvény Vége(z:Egész):Logikai
  Vége:=Mag(z)>0 és Mag(z+1)=0
Függvény vége.
```

[11] Hogy miféle műveleteket rendelünk az egyes típusokhoz, abba most nem bonyolódunk bele.

[12] Megjegyezzük, hogy az A^N sorozatnak nem csak *Tömb* feleltethető meg elvileg, de most a hangsúlyt a tételek alkalmazására helyezzük, s ekkor nincs különösebb ok arra, hogy meggondoljuk az egyéb lehetőségeket.

Majd a kimenet megfelelője következik:

```
Típus   ElsőUtolsóTip=Rekord(Első:Egész,Utolsó:Egész)
        ElsőUtolsóNTömb=Tömb(1..N:ElsoUtolsóTip) [13]
Változó Db:Egész
        Sziget:ElsőUtolsóNTömb
```

Az elő- és az utófeltétel figyelembevétele

Az előfeltételnek most nincs szerepe. Ott használjuk föl a benne levő információt, ahol a bemenő adatok „születnek”, vagyis –a majdan meggondolandó– *beolvasó* eljárás kódolásánál.

Talán a legizgalmasabb részhez értünk. Most jön létre a megoldás lényegi része, amelyet egy –feladatra utaló nevű– eljárásba foglaljuk (most: Szigetek). Két utófeltétel-variánsunk is van, mivel a második egyetlen tételhez (*szétválogatás*) kapcsolódik, így egyszerűbbnek ígérkezik, ezért válasszuk azt. (Az első változatra is vissza fogunk térni.)

A szétválogatás tétel algoritmusát, mint sablont fölhasználva, a konkrétumok behelyettesítés után az alábbi algoritmusdarabhoz jutunk:

```
Db:=0
Ciklus i=e-től u-ig
  Elágazás
    Eleje(i) esetén Db:=Db+1; Sziget(Db).Első:=i
    Vége(i)  esetén Sziget(Db).Utolsó:=i
  Elgázás vége
Ciklus vége
```

Be kell vallani, két, egymással összefüggő problémát kellett áthidalni az átírás közben. Az első mindjárt az, hogy a tétel szerint két eredménysorozathoz külön-külön tartozik egy-egy számláló, márpedig erről „nem tud” a specifikáció. Mivel itt a két sorozat elemei párt alkotnak –éppen ezt fejezi ki a specifikáció A-változata–, ezek hossza szükségképpen megegyezik. Továbbá a feldolgozás során mindig az „Első” sorozatbeli jön, amelyet a párja (egy „Utolsó”-beli) követ, így egyetlen számláló elegendő.

A másik probléma: az elágazás második ágánál –a sablon szerint– épp az előbbiekben kiselejtezett számlálót kellene növelni. Innen a számlálónövelés most már nyilvánvalóan okok miatt minden gond nélkül elhagyható.

Az algoritmusban használtunk két deklarálatlan, sőt értéket sem kapott e-t és u-t. Szerepükre világít rá az utófeltételben szereplő és eddig fel sem használt (a korábban `(*)`-gal jelölt) többlet-információ.

Az e és u meghatározására ismét valamilyen tételt kell alkalmaznunk. Ha a `(*)`-állítást utófeltételként értelmezzük, csak olyan tétel jöhet szóba, amely *egy* sorozatból egy *valamilyen*

[13] Itt nem foglalkozunk azzal, hogy sok programnyelvben *dinamikus tömbdeklaráció* nincs, s ezért az indextípus csak *statikus* lehet, azaz megadásánál változó nem szerepelhet.

tulajdonságú ($Mag_i>0$) *elem index*ét állítja elő. Egyetlen ilyen van: a *kiválasztás*, amelynek előfeltételeként viszontláthatjuk a mostani előfeltételünk egy részletét: $Mag_i=0$. Azaz minden kétséget kizáróan erről van szó. Így a hiányzó algoritmusdarabbal kiegészített megoldó eljárás az alábbi:

```
Eljárás Szigetek:
  Változó i,e,u:Egész
  i:=2                                  |
  Ciklus amíg Mag(i)>0                  |
    i:=i+1                              |
  Ciklus vége                           |
  e:=i                                  |          az „újdonság”
  i:=N-1                                |
  Ciklus amíg Mag(i)>0                  |
    i:=i-1                              |
  Ciklus vége                           |
  u:=i                                  |
  Db:=0
  Ciklus i=e-tõl u-ig
    Elágazás
      Eleje(i) esetén Db:=Db+1; Sziget(Db).Első:=i
      Vége(i)  esetén Sziget(Db).Utolsó:=i
    Elgázás vége
  Ciklus vége
```

Vegyük észre, hogy a specifikációbeli feltételek „futó” (a kvantorok mellett szereplő individuum) változói az eljárás lokális változóiként bukkannak elő.

Ami az A-változatot illeti: két független kiválogatással operáló algoritmushoz jutunk első lépésben. Szemet kell, szúrjon, hogy ezek nagyban azonosak (mindössze a cikusmagbeli feltételek különböznek). S józan megfontolásokkal[14] a fenti algoritmusokhoz jutunk.

4. A HARMADIK GONDOLATSOR

az algoritmus + a specifikáció és a kódolás

Eljutottunk a programozás legmechanikusabban[15] elvégezhető lépéséhez: a kódoláshoz. Ennél a lépésnél pontosan ismernünk kell a célnyelv szintaktikáját, szemantikáját és más természetű „korlátait”. Ezek tudatában elkészíthető jó előre az előzőekben használt algoritmikus nyelv és a célnyelv közötti ún. *kódolási szabályok*. Ilyeneket alkalmazunk most a Turbo Pascal nyelv esetére, néhány általánosabb megjegyzéssel kísérve.

[14] Megjegyzem: ún. *kódtranszformáció*kkal mechanikusan is. (Erről azonban most nem szólunk.)

[15] Szándékosan nem a *programkészítése* szót használtunk, ui. az utóbbi egy lényegesen szélesebben értelmezendő folyamat.

A Pascal programnak van néhány olyan jellemzője, amely alaposan rányomja bélyegét a kódolási lépésünkre. Ezek:

1. A program *felülről lefelé tervezés*sel szemben a Pascal programban a használatot mindig meg kell, előzze a definiálásnak, azaz *alulról felfelé építkezés* jellemzi.
2. A tömbök méretének ismertnek kell lennie már fordításkor, azaz *statikus tömbfogalm*at ismer a Pascal.

Kódolásnál követni fogjuk azt az elvet, hogy a programban három funkció: a beolvasás, a kiszámítás és az eredménymegjelenítés önálló eljárások formájában ölt testet. Azaz a program szerkezete nagy vonalakban:

```
Program ProgNév;
  ... definíciós és deklarációs rész ...
  Procedure Beolvas;
  Begin
    ... eljárástörzs ...
  End;
  Procedure Kiszámol{valami lényegre utaló név};
  Begin
    ... eljárástörzs ...
  End;
  Procedure EredménytMegjelenít;
  Begin
    ... eljárástörzs ...
  End;
Begin
  Beolvas;
  Kiszámol{valami lényegre utaló név};
  EredménytMegjelenít
End.
```

Ezután a kódolásban a legkreatívabb feladat: a Beolvas, illetve az EredménytMegjelenít eljárások megalkotása. Ehhez szükséges ismereteket az algoritmikus deklarációból származó globális adatleíró, illetve a specifikáció előfeltétel része szolgáltatja.

A globális definíciós és deklarációs részhez –most– mindössze egy többlet konstans bevezetéséről kell gondoskodnunk a 2. Pascal jellemző miatt.[16] Így ez a része a programunknak így alakul:

[16] Általában valami effélét kell tenni:

Az algoritmusban		*A kódban*
`Típus Atip=Tömb(1..N:BTip)`	⟶	`Const MaxN=`*`valamilyen konkrét szám`*`;` `Type Atip=Array [1..MaxN] of BTip;`

```
Var   N:Integer;
Const MaxN=100; {ide írandó az elvileg előforduló
                 legnagyobb elemszám}
Type  ValosNTomb=Array[1..MaxN] of Real;
Var   Mag:ValoNTomb;

Function  Eleje...;
      ...
Var   Db:Integer;
Type  ElsoUtolsTip=Record Els,Utolso:Integer End;
      ElsoUtolsNTomb=Array[1..MaxN] of ElsoUtolsoTip;
Var   Sziget:ElsoUtolsNTomb;
```

Szándékosan megtartottuk az algoritmusbeli sorrendet, bár más ott is és természetesen itt is átcsoportosíthatók az egyes deklarációs részek (figyelembe véve persze a korábban említett Pascal korlátozásokat).

Az utolsó (részletesebb) megemlítésre érdemes gondolat a beolvasás kódolása. A vezérgondolat itt az, hogy az adatok értékeinek beolvasásánál garantálnunk kell az előfeltételben előírtakat. Ez „többféle igényességgel” is elvégezhető. Mi most csak jelzésszerűen, vázlatosan oldjuk meg a beolvasás ellenőrzését: feltételezzük, hogy a felhasználó *típussértés*t nem, „legfeljebb” *értékhatár-átlépés*t követ el. Így kapjuk meg a Beolvas eljárást:

```
Procedure Beolvas;
  Var i,beI:Integer;
      beR:Real;
      jo,van:Boolean;

  Begin
    Repeat
      Write('N:'); Readln(beI);
    Until (beI>=3) and (beI<=MaxN);
              ↑              ↑
       az elõfeltételbõl   a Pascal kódolási elvbõl

    N:=beI;
    Van:=False;
    Repeat
      For i:=1 to N do
      Begin
        Repeat
          Write(i,'. mérés:'); Readln(beR);
          Case i of
            1,N: jo:=beR>0;
            else jo:=beR>=0;
          End;
          Van:=Van or (beR=0);
        Until jo;
        Mag[i]:=beR;
      End;
    Until Van;
  End;
```

Megjegyzések:

1. A beolvasott értékek ellenőrzésére –most– a célváltozóval azonos típusú segéd változókat vezettünk be. Minden elem beolvasása egy ciklus, aminek feltétele: az előfeltételben rögzített, illetve a tömb maximális méretéből adódó értékek közé esés.
2. Ha a segéd változó típusát „bővebbre” változtatnánk, akkor nagyobb teret engednénk a felhasználói hibázásnak. (Pl. Integer helyett Real vagy String.)
3. A tömbbeolvasás relatíve nagy bonyolultságának[17] az az oka, hogy van az aktuális tömbnek egy olyan jellemzője is, ami „globális” jellegű, azaz a tömb teljes beolvasása után derül csak ki ennek teljesülése vagy nem teljesülése.
4. Most a „küllemre” egyáltalán nem ügyelünk. Ez természetesen fontos lenne, de mostani témánk: a specifikáció-algoritmus-kódolás hármas kapcsolata szempontjából nem érdekes.

5. KÖVETKEZTETÉSEK

FÜGGELÉK - „ALAPISMERETEK”

F1. Néhány programozási tétel

Másolás($\mathbb{N}$,H^*,$H\to G$):G^*

Be: $N\in\mathbb{N}$, $X\in H^*$, $f:H\to G$
Ki: $Y\in G^*$
Ef: -
Uf: $\forall i(1\le i\le N): y_i=f(x_i)$

Az algoritmus:

```
Eljárás Másolás(Konstans N:PozEgész,X:Tömb(1..N:H_elemTíp),
                Változó Y:Tömb(1..N:G_elemTíp)):
  Ciklus I=1-től N-ig
    Y(I):=f(X(I))
  Ciklus vége
Eljárás vége.
```

[17] továbbá annak, hogy a későbbiekben (az F2 függelékben) példaként hozott kódolási szabálytól eltérünk

Kiválogatás($\mathbb{N}$,H^*,$H\to\mathbb{L}$):($\mathbb{N}$,$\mathbb{N}^*$)

Be: N∈$\mathbb{N}$, X∈H^*, T:H→$\mathbb{L}$

Ki: DB∈$\mathbb{N}$, Y∈$\mathbb{N}^*$

Ef: -

Uf: $DB=\sum_{i=1}^{N}\chi(T(x_i))$ ∧ Y∈[1..N]^{DB} ∧ HalmazFölsorolás(Y)
∧ ∀i∈[1..DB]: T(x_{y_i})

Az algoritmus:

```
Eljárás Kiválogatás(Konstans N:PozEgész,
                              X:Tömb(1..N:H_elemTip),
                    Változó   M:PozEgész,
                              Y:Tömb(1..M:NemNegEgész)):
  M:=0
  Ciklus I=1-től N-ig
    Ha T(X(I)) akkor M:+1; Y(M):=I
  Ciklus vége
Eljárás vége.
```

F2. Példák a kódolási szabály-gyűjteményből

Az alábbi kiragadott kódolási példáknál a deklarációknak csupán információ-nyújtás szerepe van, nem tekinthető teljesnek.

Skaláradat (pl. Integer, Real, ...) beolvasása:

```
Változó X:Egész
        ...

...
Be: X [X∈[A,B]]
...
```

```
Var X:Integer; beX:Real;
    ...

...
Repeat
  Write('Kérdés:'); Readln(beX);
Until (beX>=A) and (beX<=B) and
      (Int(beX)=beX);
X:=Trunc(beX);

...
```

Tömb (pl. Integer, Real, ...) beolvasása:

```
Változó X:Tömb(1..N:Egész)
        ...
...
Be: X [X(i)∈[A,B] i=1..N]
...
```

```
Const MaxN=1000;
Var X:Array[1..MaxN] of Integer;
    beX:Real;

...
For i:=1 to N do
Begin
  Repeat
    Write(i:3,'.elem:');Readln(beX);
  Until (beX>=A) and (beX<=B) and
        (Int(beX)=beX);
  X[i]:=Trunc(beX);
End;
...
```

IRODALOM

[]

[]

[]

[]

[]