PROGRAMOZÁSMÓDSZERTAN
2. ELŐADÁS'2005
(VÁZLAT)

1. A MODUL MINT A TÍPUSMEGVALÓSÍTÁS KERETE

1.1 Mi a típus?

Amit már tudunk – „statikus kellékek"

- ***Értékhalmaz***
- ***Műveletgarnitúra***

Miket kell megadni ahhoz, hogy mondhassuk: a típust „teljesen" ismerjük?

A statikus kellékeken, ismereteken túl:

- ***Szignatúra*** – milyen dolgokkal manipulál a művelet (*értelmezési tartomány*), s mit állít elő (*értékkészlet*)?
- A ***műveletek „(együtt)működése"*** – hogyan függnek össze a műveletek transzformációi (*axiómák*)?

Mit értünk a típus megvalósítása alatt?

Annak definiálását, hogy milyen leképezést valósítanak meg az egyes műveletek (***szemantika-definíció***), és hogyan lehet „megszólítani" őket (***szintaktika-definíció***):

1. Definiálhatjuk ***csak a kapcsolatukat***, egymásra hatásukat, axiómák megadásával. (***Algebrai specifikáció***)
2. Definiálhatjuk az egyes műveletek „***egyedi***" ***leképezés***ének megadásával. Ez az *export modul* feladata, amely tartalmazza a *szignatúrákat* és –esetleg– az *elő-utófeltételeket*. *(**Algoritmikus specifikáció***)
3. Rögzíthetjük az egyes műveleteket „***egyedi***" ***megvalósításuk***kal, amelyben már figyelembe vesszük a típus már rögzített ábrázolását. Ez a ***reprezentációs-implementációs modul*** feladata.

A fenti 2-3.-at egyaránt vonatkoztathatjuk ***algoritmus***ra és konkrét ***programozási nyelv***re. A következő („1.3. A modul") fejezetben mi az algoritmikus nyelvünket bővítjük ki úgy, hogy kezelhető legyen benne e kérdéskör. Mivel nem állunk meg a puszta elmélet síkján tisztázzuk azt is („2. A modulfogalom és a Pascal nyelv" fejezetben), hogy az általunk használt lingua franca, a Pascal programozási nyelv, milyen lehetőséget ad mindehhez.

1.2. A típus algebrai specifikációjáról

Röviden utalunk a nyelv azon kiegészítésére, amelyben tisztázni fogjuk „globális" elvárásainkat az adott típussal szemben. Ezt a legabsztraktabb elképzelést formálisan egy algebrai alapú nyelven írjuk le, az alábbi keretben:

Szintaktikai „minta”	Magyarázat
Típus *TípusKonstrukció*(*paraméterek*): **Asszociált műveletek:** *Operáció(értelmezési tartomány): Értékkészlet* …	*A definiálandó típuskonstrukció „alkalmazási sablona” (feje).* *Operációinak szignatúrája (értelmezési tartománya, értékkészlete)*
Axiómák: **Jelölések:** • *objektum: Típus(paraméterek) =* *értelmező megjegyzés* • … **1°** … az axióma informális szövege ... *a formalizált axióma* **2°** … …	*Az operációk kapcsolatát leíró axiómák, és a bennük fölhasznált objektumok „deklarációi”.* *Axióma-állítások esetleg kibővítve fontos következményállításokkal.*

Mint a példából is kiderül alkalmazunk néhány konvenciót, amelyet most –az idő rövidsége miatt– nem taglalunk. (A részletekhez lásd a Típus-specifikációkban.)

Visszatérő példa legyen a hét napjainak típusa! Nézzük ennek egy lehetséges algebrai leírását! Megjegyzem: a példa nem „tipikus” bizonyos értelemben, mivel a napok típusa egy *konkrét típus*, így nem tudunk élni a fenti típus-specifikálási „nyelvezet” paraméterezési szolgáltatásaival.

Példa:

Típus TNap:

Asszociált műveletek:

Hétfő,Kedd,Szerda,Csütörtök,
Péntek,Szombat,Vasárnap,
Min,Max:TNap
Számosság:Egész[1]

Létrehoz:TNap
Lerombol(TNap)
Következő(TNap):TNap ∪ {NemDef}
Előző(TNap):TNap ∪ {NemDef}
Sorszám(TNap):Egész
TNap(Egész):TNap ∪ {NemDef}
<(TNap,TNap):Logikai

Axiómák:

• *a:TNap*
• *n:Egész*
0° A minimális érték a Hétfő, a maximális a Vasárnap, a típus számossága 7.

[1] A konstansokat 0-változós függvényeknek tekintjük.

```
   Számosság=7  ∧  Min=Hétfő  ∧  Max=Vasárnap
1° Létrehozás utáni értéke Hétfő.
   a=Létrehoz ⇒ a=Hétfő
2° Lerombolás után nincs értelme a műveletek (a Létrehoz
   kivételével) alkalmazásának.
   Következő(Lerombol(a))=NemDef  ∧  …
3° Nem létezik Hétfőt megelőző, s Vasárnapot követő.
   Előző(Hétfő)=NemDef ∧ Következő(Vasárnap)=NemDef
4° A Következő és Előző műveletek egymás inverzei.
   ∃Előző(a) ⇒ Következő(Előző(a))=a  ∧
   ∃Következő(a) ⇒ Előző(Következő(a))=a
5° A Sorszám és TNap műveletek egymás inverzei.
   n∈[0..Számosság) ⇒ Sorszám(TNap(n))=n  ∧
                       TNap(Sorszám(a))=a  ∧
    n∉[0..Számosság) ⇒ TNap(n)=NemDef
6° A fenti felsorolás sorrendjében növekvően rendezettek.
   Következő(Hétfő)=Kedd  ∧  Következő(Kedd)=Szerda …
7° A Hétfő sorszáma 0, a Vasárnapé 6.
   Sorszám(Hétfő)=0  ∧  Sorszám(Vasárnap)=6
Állítás: Sorszám(a)∈[0..Számosság); és a
  Sorszám(a)=a fenti felsorolásbeli pozíciója + 1.
Bizonyítás:
  Nyilvánvaló.
8° A felsorolásban előbb szereplő kisebb, mint bármely
   későbbi.
   a<b  ⇔  Sorszám(a)<Sorszám(b)[2]…
```

Megjegyzések:

- Figyeljük meg a következőket! A ***Létrehoz*** *értelmezési tartomány nélküli* függvény, amely a „semmiből” kreálja meg az adott típusú adatot; a ***Lerombol*** pedig *értékkészlet nélküli*, amivel épp a többi, „normális” függvény ezután történő alkalmazhatatlanságát fejeztük ki. Ezt nyomatékosítjuk a 2^{o} axiómával is.
- Lehetséges, hogy a ***Létrehoz*** és ***Lerombol*** műveletekre a későbbiek során ***explicite nem*** lesz szükségünk. Ez a helyzet az ún. *statikus* adatoknál, amelyre az a jellemző, hogy a deklaráció által *automatikusan* jön létre, s így a programozónak nincs „gondja” vele. (Érdemes meggondolni, mikor kerül sor e két művelet végrehajtására a *blokk-struktúrájú nyelvek –pl. a Pascal–* esetén.)

[2] <:TNap**x**TNap→Logikai rendezés visszavezetése <:Egész**x**Egész→Logikai rendezésre.

- A ***NemDef*** egy típusfüggetlen konstans, ami akkor képződik, amikor valamilyen tiltott művelet végrehajtására kerülne sor. Ha egyszer *NemDef* érték jött ki, akkor arra alkalmazva bármely operációt, az eredmény *NemDef* marad (***implicit error-axióma***).
- Nyilvánvaló, hogy a **2-8°** axiómák feltételéül meg kellett volna fogalmaznunk, hogy az ott szereplő ***objektum létezik*** (objektumok léteznek), azaz létrehoztuk korábban. Ezt *implicite* föltételeztük.
- A **6°** axióma párjaként megfogalmazható „Előző-rendezési axiómát” következmény-állításként illeszthetjük a rendszerbe. (Most nem tesszük meg.)

1.3. A modul

Elsőként a programozói felületet leíró, ún. ***export modul*** szintaxisát adjuk meg:

```
ExportModul TípusNév(InputParaméterek):
    [most következnek az exportálandó fogalmak „értelmes” csoportosításban:]
  Típus   [ritkán van ilyenre szükség]
    Tip1,Tip2,...  [csak azonosítók felsorolása!]
  Konstans   [ritkán van ilyenre szükség]
    Kons1,Kons2,...[a TípusNév típus néhány kiemelendő konstansa]
  [az alábbi részben soroljuk föl a típushoz tartozó tevékenységek „fejsorait”
   és működésüket leíró elő-utófeltételeket]
  Függvény Fv1(FormParam): FvTip1
    Ef: …
    Uf: …
  Függvény Fv2(FormParam): FvTip2
    Ef: …
    Uf: …
  …
  Eljárás Elj1(FormParam)
    Ef: …
    Uf: …
  Eljárás  Elj2(FormParam)
    Ef: …
    Uf: …
  …
  Operátor Op1(FormParam): OpTip1
    Ef: …
    Uf: …
  Operátor Op2(FormParam): OpTip2
    Ef: …
    Uf: …
   …
Modul vége.
```

Megjegyzések:

- A „fő fogalom”, a ***TípusNév***, külön megadás híján is látszódó fogalom lesz.

- Az ***InputParaméterek*** lehetnek típust jellemző konstansok, de –típuskonstrukció esetén– típusok is. (Szokás e paramétereket 'generic parameter'-nek nevezni.)
- Az ***elő-utófeltételek*** időnként *elhagyhatók*. (Pl. ha ezeket csak az ábrázolás ismeretében lehet megfogalmazni, ami e pillanatban még nem ismert.)
- A ***modulfej*** egyben a felhasználás „*mintá*jául" is szolgál (l. az alábbi példát).

Példa:

```
ExportModul Tömb(Típus TIndex: Típus TElem):
…
Modul vége.
```

A fenti definíció után a felhasználás szintaxisa:

```
Típus  Vektor=Tömb(1..9:Egész)
…
```

- Az egyes ***exportált fogalmak csoportosítás***ra semmilyen megkötés nincs: sem sorrendjére, sem arra, hogy hány „darabba" különítjük el őket. (Tehát lehet több konstans, típus vagy más csoport is.)
- ***Kerülni kell*** a változók exportálását. Ui. veszélyes, ha a rátámaszkodó programnak lehetősége van a reprezentált adatokkal *közvetlen* kapcsolatba kerülni. Ha ilyen igény merülne föl, akkor *definiálni kell* külön erre a célra *tevékenységeket* (eljárásokat, függvényeket, operátorokat).

Ki fogjuk egészíteni az algebrai specifikáció műveletkészletét továbbiakkal is: ***egyenlőség-vizsgálat***tal, ***értékadás***sal, ***beolvasás*** és ***kiírás*** műveletekkel, valamint a hibás elvégzés tényét rögzítő ***hiba-flag-et kezelő függvénny***el. (Ezek nélkül ugyanis felhasználhatatlan eszköz maradna a legtöbb típus, vagy mindenegyes esetben a programozónak külön kellene jól-rosszul ezen műveletekkel kiegészítenie.) Mindezt annak ellenére tesszük, hogy sok esetben a programozási nyelvek nem adnak mindegyikre (pl. a „típusos" be/ki-re) módot.

Egy-két típusnál szokatlan lehet valamely ***létrehoz***ó, ill. ***lerombol***ó művelet használatának explicit felkínálása, mivel a programozási nyelvek legtöbb implementációja *automatikusan* gondoskodik a létrehozásról. (Pl. `Tömb`, `Táblázat`, `Statikus gráf`.) A leírás teljességére való törekvés, illetve az esetleges „hagyományostól" eltérő ábrázolás lehetővé tétele mégis indokolttá tehetik ezt.

Nézzük visszatérő példánk egy lehetséges export modulját!

Példa:

```
ExportModul TNap: [kiindulási alap az algebrai leírás]
  Konstans
    Hétfő, Kedd, Szerda, Csütörtök, Péntek, Szombat,
    Vasárnap:TNap
  Operátor Min:TNap
    Másnéven Min'TNap
    Ef: –
```

```
  Uf: Min=Hétfő
Operátor Max:TNap
  Másnéven Max'TNap
  Ef: -
  Uf: Max=Vasárnap
Operátor Számosság:Egész
  Másnéven Számosság'TNap
  Ef: -
  Uf: Számosság=7
Függvény Következő(Változó x:TNap):TNap
  Ef: x≠Vasárnap [v.ö. a 3. axiómával]
  Uf: Következő(Hétfő)=Kedd ∧ … [v.ö. a 6. axiómával]
Függvény Előző(Változó x:TNap):TNap
  Ef: x≠Hétfő [v.ö. a 3. axiómával]
  Uf: Előző(Kedd)=Hétfő ∧ … [v.ö. a 6.&4. axiómával]
Függvény Sorszám(Konstans x:TNap):Egész
  Ef: -
  Uf: Sorszám(Hétfő)=0 ∧ … [v.ö. a 7. axiómával]
Függvény TNap(Változó x:Egész):TNap
  Ef: x∈[0..6]
  Uf: TNap(0)=Hétfő ∧ …
Infix Operátor Egyenlő(Konstans x,y:TNap):Logikai
  Másnéven x=y
  [ezt a függvényt nem specifikáljuk, mert az „alapokat" firtatja, és
   elkerülhetetlenül circulum viciosus-ba ütköznénk]
Infix Operátor LegyenEgyenlő(Változó x:TNap
                              Konstans y:TNap)
  Másnéven x:=y
  [ezt az eljárást nem specifikáljuk, mert az „alapokat" firtatja, és
   elkerülhetetlenül circulum viciosus-ba ütköznénk]
Infix Operátor Kisebb(Konstans x,y:TNap):Logikai
  Másnéven x<y
  Ef: -
  Uf: Kisebb(Hétfő,y) ⇔ y≠Hétfő ∧
      …
Operátor Be(Változó x:TNap)
  Másnéven Be:x
  Ef: Input∈{'hétfő','kedd','szerda'…}
  Uf: Input='hétfő' ⇒ x=Hétfő ∧
Operátor Ki(Konstans x:TNap)
  Másnéven Ki:x
  Ef: -
  Uf: x=Hétfő ⇒ Output='hétfő' ∧
Függvény Hibás?(Változó x:TNap):Logikai
  [teszteli, hogy volt-e hibásan elvégzett művelet (pl. amelynek
   az előfeltétele nem teljesült) az adott TNap típusú (x) adatra;
```

Megjegyzés [SzP1]: Lehetne így is:
Függvény Következő(**Konstans** x:TNap): TNap
És ha hiba történt, akkor e tényt a TNap típusú **eredmény**-adat árulja el, az ő hiba-flag-je Igaz. Ez azonban sok gondot vet föl. Pl. egy kifejezés eredményeként születik egy hibás TNap-konstants?!?

```
        majd inicializálódik a hiba-flag]
Modul vége.
```

Megjegyzések:

- A **Létrehoz** és **Lerombol** műveleteket ***kihagytuk*** az algoritmikus specifikációból. Ezzel eltértünk az algebrai specifikációtól. Ok: ***statikusan*** kívánjuk megvalósítani e felsorolástípust, építünk a *blokk-struktúrájú nyelvek* adatlétrehozó, -leromboló mechanizmusainak létére. (Gondoljon az eljárás-/függvény-paraméterekre és a lokális adatok kezelésére!) A *Létrehoz-axióma* persze még fel kell bukkanjon a megvalósításban a szükséges kezdőérték megadása miatt.
- A **Min**, **Max** és **Számosság** konstansokat (prefix) *0-változós operátor*ként definiáltuk, s nem konstansként, mert alkalmazásuk speciális szintaxisú (típusparaméterű). Vegyük észre, hogy ezek a konstansok minden rendezett, illetőleg véges típus esetén értelmesek. Így a használó program szintjén való megkülönböztetés érdekében olyan szintaxist kell hozzájuk rendelni, amely lehetővé teszi a típushoz kapcsolást. Ilyen probléma nem vetődik föl a többi (`Hétfő`, `Kedd` stb.), natív konstanssal kapcsolatban. (Ez a magyarázata a **Min'**Típus, **Max'**Típus, illetve a **Számosság'**Típus különleges szintaxisának.)

  Példa:

```
…
  Változó
    ma:TNap
    napDb:Egész
…
  Ciklus ma=Min'TNap-tól Max'TNap-ig
    …
  Ciklus vége
  napDb:=Számosság'TNap
…
```

- Látható, hogy akkurátusan törekedtünk –bármi áron is!–, hogy az ***utófeltételekben ne nyúljunk vissza más művelethez***, csak saját „hatáskörben" definiáltuk az elvárásokat. Ha ezt az elvet nem tartottuk volna tiszteletben, akkor egyszerűen meg lehetett volna fogalmazni a **:=** és **=** operátorok utófeltételét. Például:

  `Egyenlő.Uf: x=y ⇔ Sorszám(x)=Sorszám(y).`

  Ekkor azonban könnyen előadódhatna az a helyzet, hogy egymásra *kölcsönösen* hivatkozva specifikáljuk őket, ami egy logikailag *értelmezhetetlen* specifikációt eredményezne.
- Az ***értékadás*** és az ***értékazonosság*** operációkról feltesszük, hogy *bitről-bitre történő másolás*t, illetőleg *azonosságvizsgálat*ot jelent, ezért nem tartjuk fontosnak feltételekkel körülbástyázni. Másrészt, ha ezt formálisan is le akarnánk írni, vissza kellene nyúl-

ni (esetleg bit-szintű) reprezentációig, amit effajta specifikációnál elkerülendőnek tekintünk.

- Az időnként felbukkanó „**…**” folytatás jelzése formálisan ***megengedhetetlen***, azonban itt csupán egy példáról van szó, így megelégedtünk a kitalálható folytatás jelzésével.
- Fölvethető kérdés: mennyire *ugyanarról szól* a kétféle specifikáció. Valóban ez külön vizsgálandó, sőt bizonyítandó probléma.
- A „**Be:**” és „**Ki:**” operátorok működésénél két problémát kell tudni formalizálni:
  1. a ***külvilággal való kapcsolat***ét (amely karakteres alapú),
  2. ***Szöveg↔TNap transzformáció***ét.

  Az 1. általános megoldására bevezettünk két függvényt. Az **Input** megadja azt a *szöveg*et, amelyet a beolvasó utasítás talál az input-pufferben; az **Output** szimbolizálja az output-puffer állapotát (azt a *szöveget*, amely kiíródik a végrehajtás során, pl. a képernyőre). Így a 2. probléma már kezelhetővé vált, hisz az adat és transzformáltja is egy-egy memóriabeli objektumban csücsül. Fel kell figyelnünk a szöveges és a „belsőállapotú” konstansok merev megkülönböztetésére (`'hétfő'≠Hétfő`…)!
- **Operátor**`nak` azt az alprogramot nevezzük, amely az eljárástól vagy függvénytől *eltérő szintaxis*sal építhető be a programba (azaz hívható). Lehet **Infix**, ekkor a két operandus ***közé*** illeszkedik, lehet **Prefix**, ekkor ***megelőzi*** az egyetlen paraméterét, ill. **Postfix** esetben ***követi***. S lehet „egyéni” szintaxisa, ekkor a **Másnéven** kulcs-szónál adhatjuk meg ezt. A leggyakoribb eset a **Prefix**, ezért ezt a minősítőt elhagyhatjuk.
- Egyik-másik operátornál „meglepő” a paraméter ***hozzáférési jog***a. A várható konstans helyett változó lett azért, mert a végrehajtás során hiba következhet be, amely által az „állapota” kényszerűen és szándékunk ellenére megváltozik. (**Következő**, **Előző**… **Hibás?**. Nagyjából azoknál, amelyeknél nem üres az előfeltétel, tehát van „esélye” a hibás használatnak. Pontosan azoknál, amelyeknek bármi köze is van a hiba-flag-hez.)

Folytassuk a megvalósítást betetéző fogalommal, a ***reprezentációs-implementációs modul***lal, röviden a ***modul***lal! Ennek feladata, hogy amit eddig elterveztünk, azt kidolgozza: azaz tisztázza, hogy

1. miként ábrázolható a típusbeli érték a memóriában, vagy bárhol (*reprezentáció*) és
2. hogyan valósíthatók meg az egyes műveletek, figyelembe véve a választott ábrázolást (*implementáció*).

A modul-szintaxis a következő:

```
Modul TípusNév(InputParaméterek):
 Reprezentáció
  …
  [a típusábrázoláshoz szükséges adatleíró „kellékek”:
   konstansok, saját típusok,
   a változó-deklarációs rész már magának a típusnak a „mezőit” határozza meg]
```

```
Implementáció
  [itt szerepelnek az export-modulban „megjósolt”
   tevékenységek működésének részletezése…]
 Függvény Fv1(FormParam): FvTip1
   Ef: …
   Uf: …
   …
 Függvény Fv2(FormParam): FvTip2
   Ef: …
   Uf: …
   …
 Eljárás Elj1(FormParam)
   Ef: …
   Uf: …
   …
 Eljárás Elj2(FormParam)
   Ef: …
   Uf: …
   …
 Operátor Op1(FormParam): OpTip1
   Ef: …
   Uf: …
   …
 Operátor Op2(FormParam): OpTip2
   Ef: …
   Uf: …
   …
 Inicializálás
   [egy ilyen típusú adat létrehozása az itt leírt utasításokkal történik]
Modul vége.
```

Megjegyzések:

- Az egyes műveletek egyedi specifikációját jelentő elő-utófeltétel párok, amennyiben megegyeznek az exportmodulbelivel, nyilván elhagyhatók. Természetesen „értelmesen” igazítva a reprezentációhoz, újabb biztonsági támpontul szolgálhat a fejlesztéshez. (Ekkor persze újabb feladatként járul az eddigiek mellé, hogy ezek következését az „elődjükből” be kell látni:
  $\text{operáció}_{\text{Exportmodul}}.\text{Ef} \Rightarrow \text{operáció}_{\text{Modul}}.\text{Ef} \wedge \text{operáció}_{\text{Exportmodul}}.\text{Uf} \Rightarrow \text{operáció}_{\text{Modul}}.\text{Uf}$.)

A visszatérő példánk egy lehetséges moduljának töredékét tervezzük meg az alábbiakban. A reprezentációs döntésünk, hogy legyen minden TNap adatnak legyen egy hiba-flag-je.

Példa:

```
Modul TNap: [kiindulási alap az export modul]
 Reprezentáció
   Változó
     felsKód:Egész
     hiba:Logikai
```

```
  Konstans
    Hétfő:TNap(0,Hamis)
    Kedd:TNap(1,Hamis)
    …
    Vasárnap:TNap(6,Hamis)
Implementáció
 Operátor Min:TNap
   Másnéven Min'TNap
   Ef: -
   Uf: Min=Hétfő
   Min:=Hétfő
 Operátor vége.
 Operátor Max:TNap
   Másnéven Min'TNap
   Ef: -
   Uf: Max=Vasárnap
   Max:=Vasárnap
 Operátor vége.
 Operátor Számosság:Egész
   Másnéven Számosság'TNap
   Ef: -
   Uf: Számosság=7
   Számosság:=7
 Operátor vége.
 Függvény Következő(Változó x:TNap):TNap
   Ef: x≠Vasárnap
   Uf: x=Hétfő  ⇒ Következő(x)=Kedd  ∧ …
   Ha felsKód=Vasárnap.felsKód
      akkor    hiba:=Igaz
      különben Következő.felsKód:=felsKód+1
 Függvény vége.
 Függvény Előző(Változó x:TNap):TNap
   Ef: felsKód≠Hétfő
   Uf: felsKód=Kedd ⇒ Előző(x).felsKód=Hétfő  ∧ …
   …
 Függvény vége.
 Függvény Sorszám(Konstans x:TNap):Egész
   Ef: -
   Uf: x.felsKód=Hétfő.felsKód ⇒ Sorszám(x)=0  ∧ …
   …
 Függvény vége.
 Függvény TNap(Változó x:Egész):TNap
   Ef: x∈[0..6]
   Uf: TNap(0).felsKód=Hétfő.felsKód  ∧  …
   …
 Függvény vége.
 Infix Operátor Egyenlő(Konstans x,y:TNap):Logikai
   Másnéven x=y
```

Megjegyzés [SzP2]: Típus-konstrukciós függvény

```
    [ezt a függvényt nem specifikáljuk, mert az „alapokat” firtatja,
     és elkerülhetetlenül circulum viciosus-ba ütköznénk]
    …
  Operátor vége.
  Infix Operátor LegyenEgyenlő(Változó x:TNap
                               Konstans y:TNap)
    Másnéven x:=y
    [ezt a függvényt nem specifikáljuk, mert az „alapokat” firtatja,
     és elkerülhetetlenül circulum viciosus-ba ütköznénk]
    …
  Operátor vége.
  Infix Operátor Kisebb(Konstans x,y:TNap):Logikai
    Másnéven x<y
    Ef: –
    Uf: x.felsKód=Hétfő.felsKód ⇒ Kisebb(x,y)  ⇔
        y.felsKód≠Hétfő.felsKód  ∧  …
        …
    …
  Operátor vége.
  Operátor Be(Változó x:TNap):
    Másnéven Be:x
    Ef: Input∈{'hétfő','kedd','szerda'…}
    Uf: Input='hétfő'  ⇒  x=Hétfő  ∧  …
    …
  Operátor vége.
  Operátor Ki(Konstans x:TNap):
    Másnéven Ki:x
    Ef: –
    Uf: x=Hétfő  ⇒  Output='hétfő'  ∧
    …
  Operátor vége.
  Függvény Hibás?(Változó x:TNap):Logikai
    Hibás?:=hiba; hiba:=Hamis
  Függvény vége.
  …
 Inicializálás
   hiba:=Hamis; felsKód:=Hétfő
Modul vége.
```

Megjegyzés:

- Figyeljünk föl rá, hogy az ***utófeltételben*** sok esetben ***nem rögzítjük a hiba-flag értékét***! Ez szándékos: ha hibás volt a művelet előtt, maradjon is az (implicit error-axióma), ha hibás ereményre vezet, akkor persze állítódjon át garantáltan Igaz-ra.
- Vegyük észre: az ***inicializálás***ban éppen az **1°** axióma szerint jártunk el. A többieknél is, csak azoknál nem ennyire „látványosan”.
- Időnként attól a ***konvenciónktól eltérünk***, hogy a típus reprezentációjában szereplő komponensekre (`felsKód`, `hiba`) nem a rekordnál szokásos mezőhivatkozásos jelöléssel tesszük meg (`x.felsKód` helyett csupán `felsKód`), az a magyarázata, hogy

az adott operációnál *nem* lenne *egyértelmű*, hogy melyik ilyen típusú adat valamely komponenséről van éppen szó. Pl. a **Következő** operációban „szóba kerül" két TNap típusú objektum is: a kiindulásul szolgáló x paraméter és az eredményt jelentő.

2. A MODULFOGALOM ÉS A PASCAL NYELV

Amire mód van: *algoritmikus specifikáció* és a *megvalósítás*; amire nincs: az algebrai specifikáció. Többféle Pascal-nyelvi lehetőséget lehet a típusgyártás szolgálatába állítani:

- **Unit** – *önálló fordítási egység*, amely lehetne a típus „zárt", lefordított kerete. Zárt és lefordított ⇒ *paraméterezni* már *nem lehet*, többször *fordítani* viszont *nem kell*.
- **Include-állomány** – *forrásprogram-töredék*, amelyben a megvalósított típus „dolgai" találhatók, de bizonyos részek nem definiáltak benne, pl. a paraméterei, így „részt vesz" minden fordításban annak ellenére, hogy benne esetleg nincs változás;
- **Objektum** – olyan rekord-szerű képződmény, amely egységbe zárja a típus két kellékét: az értékhalmazt és a műveleteit; mindezt úgy teszi, hogy akár része lehet a fejlesztendő programnak, akár kiemelhető unit-ba vagy include-állományba (azok minden előnyével és hátrányával).

Az nyilvánvaló, hogy a Pascal-ban

- nincs *operátordefiniálás*i lehetőség, helyette vagy **function**-t vagy **procedure**-t kell használni;
- *csak egyszerű* értéket szolgáltató *függvények*et lehet definiálni (használni), ezért ilyenek gyanánt alkalmasan átalakított **procedure**-kat kell difiniálni. Az átalakítás persze igencsak mechanikus, ahogy az alábbi példa mutatja:

Algoritmikus	⇒	**Pascal**

`Operátor Min:TNap`
`  Másnéven Min'TNap`
`  Min:=Hétfő`
`Operátor vége.`
	⇒	`Procedure Min(Var minKi:TNap);`
		`Begin`
		`  minKi:=Hetfo`
		`End;`

2.1. Unit

- A unit szintaxisa igazodik a Pascal nyelvhez, így túl sok magyarázatra nincs szükség. Annyit előljáróban: a unit egyesíti az exportmodul (**Interface**-rész) és a modul (**Implementation**-, inicializáló rész) fogalmat.

```
Unit TipusNev;{InputParaméterek}
 Interface
  …
  {a kívülről láttatandó fogalmak:
   unit-ok, konstansok, típusok, változók;  eljárások és függvények fejsorai}
  …
```

```
Implementation
  …
  {az elrejthető reprezentációs dolgok:
   unit-ok, konstansok, változók, típusok, „saját” eljárások és függvények;
   az exportált eljárások és függvények törzsükkel együtt}
  …
Begin
  …
  {a típus adatainak inicializálását végző utasítások}
  …
End.
```

Megjegyzések:

- A program ***végrehajtása*** azzal ***kezdődik***, hogy az összes hozzászerkesztett unit (**Begin** és **End**.-je közé tett) inicializáló utasításai *egyszer* s mindenkorra végrehajtódnak. Azaz nincs mód arra, hogy ha több adott (valamely unit-ban megvalósított) típusú adat van a programban, akkor azok mindegyike külön-külön inicializálódjék.
- Mivel ***nincs mód paraméterezésre***, ezért ellenáll mindenféle „általánosításnak”. A fenti kommentben levő InputParaméterek csak a fejlesztőnek szolgálnak információt arról, hogy mik azok a fogalmak, amelyek, ha lehetséges volna, paraméterként funkcionálhatnának.
- A ***unit neve*** és ***a befoglaló file neve azonos*** kell, legyen!
- Felhasználása: a „hívó” program **Uses** utasítása sorolja föl a felhasznált összes unitot.
- Ha ugyanaz az azonosító több hivatkozott modulban is előfordul (vagy magában a hivatkozó programban), akkor fölvethető a kérdés: melyikre mi módon lehet hivatkozni. A válasz: ***minősített azonosító*** alkalmazása, azaz a **unitnév** + **pont** + **azonosító**.
  Például: írható saját `ClrScr`-eljárás, amelyben építhetünk a `Crt` unit hasonló nevű eljárására:

```
…
  Uses …,Crt,…;
  …
  Procedure ClrScr(Const cím:String);
  Begin
    Crt.ClrScr;
    Writeln(cím:40+(Length(cím) Div 2));
    …
  End;
…
Begin
  ClrScr('Ügyes kis program');
…
```

Az alábbi példa bemutatja, hogyan lehet a TNap típust unit-tal megvalósítani. A Pascal nyelv fentebb említett kellemetlen vonásai miatt a TNap-értékű függvényeket procedure-ákkal kell helyettesíteni.

Példa:

```
Unit TNapUnit;{InputParaméterek}
 Interface
  Type
    TNapK=(Hetfo,Kedd,Szerda,Csutortok,
           Pentek,Szombat,Vasarnap);
    TNap=Record felsKod:TNapK; hiba:Boolean End;
  Procedure Min(Var minKi:TNap);
    {Ef: -
     Uf: minKi=TNap(Hetfo,Hamis)}
  …
  Function Szamossag:Word;
    {Ef: -
     Uf: Szamossag=7}
  Procedure Kovetkezo(Var x:TNap; Var kovetkezoKi:TNap);
    {Ef: x.felsKod<>Vasarnap
     Uf: x.felsKod=Hetfo ⇒ KovetkezoKi.felsKod=Kedd  ES…}
  …
  {erre nincs szükség, ezt tudja a Pascal (:=):
  Procedure LegyenEgyenlő(Var x:TNap
                          Const y:TNap);
  }
  Function Egyenlo(Const x,y:TNap):Boolean;
    {Ef: …
     Uf: …}
  Function Kisebb(Const x,y:TNap):Boolean;
    {Ef: …
     Uf: …}
  Procedure Be(Var x:TNap);
    {Ef: Input ELEME {'HÉTFŐ','KEDD','SZERDA'…}
     Uf: Input='HÉTFŐ'  ⇒  x=Hetfo  ES  …}
  Procedure Ki(Const x:TNap);
    {Ef: -
     Uf: x.felsKod=Hetfo  ⇒  Output='HÉTFŐ'  ES  …}
  Function HibasE(Var x:TNap):Boolean;
 Implementation
  Type
    TNapS=Array [1..7] of String[9];
  Const
    NapS:TNapS=('HÉTFŐ','KEDD','SZERDA',
                'CSÜTÖRTÖK','PÉNTEK',
                'SZOMBAT','VASÁRNAP');
```

```
  Procedure Min(Var minKi:TNap);
    {Ef: -
     Uf: minKi=TNap(Hetfo,Hamis)}
  Begin
    minKi.felsKod:=Hetfo; minKi.hiba:=False
  End;
  …
  Function Szamossag:Word;
    {Ef: -
     Uf: Szamossag=7}
  Begin
    Szamossag:=7
  End;
  Procedure Kovetkezo(Var x:TNap; Var kovetkezoKi:TNap);
    {Ef: x.felsKod<>Vasarnap
     Uf: x.felsKod=Hetfo  ⇒ KovetkezoKi.felsKod=Kedd  ES…}
  Begin
    With x do
    Begin
      {Ef-ellenőrzés:}
      If felsKod<>Vasarnap then
      Begin
        hiba:=True; Exit
      End;
      {művelet-törzs:}
      KovetkezoKi.felsKod:=Succ(felsKod)
    End;
  End;
  …
  Function HibasE(Var x:TNap):Boolean;
  Begin
    HibasE:=x.hiba; x.hiba:=False
  End;
Begin
End.
```

Megjegyzések:

- Sajnálatos módon a sajátos Pascal logika miatt olyanok is belekerülnek az **Interface** részbe, amelyeket ténylegesen nem szeretnénk exportálni. (`TNapK`, `TNap` „belső struktúrája”…)
- A `TNap` típus felhasználását a következő példa mutatja.

  Példa:

```
Program AProgram;
  Uses …,TNapUnit,….;
  Var
    tegnap,ma,holnap:TNap;
```

```
  …
Begin
  …
  Repeat
    Write('Milyen nap van ma?:'); Be(ma);
  Until not HibasE(ma);
  …
  Writeln('Holnap:');
  Kovetkezo(ma,holnap); Ki(holnap);
  …
End.
```

Egy más elvű megoldást is találhatunk, ami azonban nem a modulból, hanem az exportmodulból indul ki. Ötlete a következő: a napok absztrakt felsorolását egészítsük ki egy további, pl. NemDef-fel elkeresztelt értékkel. Az előbbi `TNapK` és `TNap` kettőse helyett ábrázoljuk így:

```
Típus TNapK=(Hétfő, Kedd, Szerda, Csütörtök, Péntek,
             Szombat, Vasárnap, NemDef)
Változó érték:TNapK
```

Ekkor egyszerű értéktípusúak lesznek az eleddig rekord-típusú értéket szolgáltató függvények. Pl.:

```
Függvény Következő(Változó x:TNap):TNap
  Ef: x≠Vasárnap
  Uf: x=Hétfő ⇒ Következő(x)=Kedd  ∧ …
  Ha érték=Vasárnap akkor   x:=NemDef
                    különben Következő:=Következő(érték)
Függvény vége.
```

Megjegyzés [SzP3]: Ez a TNapK nem pedig az éppen definiálandó TNap Következő művelete.

Így a Pascal-megfeleltetés úgyszólván problémamentes:

```
Unit TNapUnit;{InputParaméterek}
 Interface
   Type
     TNapK=(Hetfo,Kedd,Szerda,Csutortok,
            Pentek,Szombat,Vasarnap, NemDef);
    {a TNap=Record ertek:TNapK End helyett:}
     TNap=TNapK;
   …
   Function Kovetkezo(Var x:TNap):TNap;
   …
 Implementation
    …
    Function Kovetkezo(Var x:TNap):TNap;
    {Ef: x<>Vasarnap
     Uf: x=Hetfo  ⇒ Kovetkezo(x)=Kedd  ES …}
    Begin
      If x=Vasarnap then x:=NemDef
                    else Kovetkezo:=Succ(x)
    End;
```

```
  …
```

2.2. Kódtöredék automatikus beillesztése

A unitok nyilvánvaló hátrányát igyekszik kiküszöbölni az „állomány automatikus beillesztés” lehetősége. A lényeg: a Pascal fordítóprogramot „rávesszük”, hogy a kódtöredéket tartalmazó file-t a befoglaló program adott pontján illessze be. Ennek módja: **{$i** `töredék-file-név` **}** direktíva az adott helyen.

Példa:

Legyen az alábbi töredék a `TNap.inc` nevű file-ban:

```
{TNap
  InputParaméterek - ha lennének. Most nincs.
}
  Type
    TNapK=(Hetfo,Kedd,Szerda,Csutortok,
           Pentek,Szombat,Vasarnap);
    TNap=Record felsKod:TNapK; hiba:Boolean End;
    TNapS=Array [1..7] of String[9];
  Const
    NapS:TNapS=('HÉTFŐ','KEDD','SZERDA',
                'CSÜTÖRTÖK','PÉNTEK',
                'SZOMBAT','VASÁRNAP');
  …
  Procedure Kovetkezo(Var x:TNap; Var kovetkezoKi:TNap);
    {Ef: x.felsKod<>Vasarnap
     Uf: x.felsKod=Hetfo ⇒ KovetkezoKi.felsKod=Kedd  ES…}
  Begin
    With x do
    Begin
      {Ef-ellenőrzés:}
      If felsKod<>Vasarnap then
      Begin
        hiba:=True; Exit
      End;
      {művelet-törzs:}
      KovetkezoKi.felsKod:=Succ(felsKod)
    End;
  End;
  …
```

A felhasználás:

```
Program AProgram;
  Uses …;
  …
  {$i TNap.inc - a TNap típus beillesztése}
  Var
    tegnap,ma,holnap:TNap;
  …
```

```
Begin
  …
End.
```

Megjegyzés:

- A fenti példából sajnos nem látszik a leglényegesebb különbség: a „kvázi paraméterezhetőség”. A későbbiek során azonban számos példa lesz a unit paraméter-problémájára.

2.3. Objektum-osztály

Objektum, helyesebben objektum-osztály fogalma számunkra azzal az előnnyel jár, hogy a programon belül megvalósíthatjuk a megfelelő típusfogalmat, azaz az értékhalmaz és művelethalmaz egységét, egységbezárását (encapsulation).[3] A szintaxisáról elegendő ennyit tudni:

```
…
  Type
    TipusNev=Object
      …
      {mező-deklarációk, amit itt az objektum
       attribútumainak vagy tulajdonságainak hívnak}
      …
      {a típus műveleteinek, metódusainak fejsorai következnek:}
      Constructor EljC(…); {elhagyható}
      Destructor EljD(…); {elhagyható}
      Procedure Elj1(…);
      …
      Function Fv1(…):TFv1;
      …
    End;
    {a típus műveleteinek kifejtése:}
    Constructor TipusNev.EljC(…);
    …
    Begin
      …
    End;
    Destructor TipusNev.EljD(…);
    …
    Begin
      …
    End;
    Procedure TipusNev.Elj1(…);
    …
    Begin
      …
    End;
```

[3] Persze elismerjük, hogy ennél jóval több újdonságot rejt az objektumok fogalma, számunkra most ennyi pontosan elegendő.

```
    …
    Function TipusNev.Fv1(…):TFv1;
    …
    Begin
      …
    End;
  …
  Var
    a,b,c:TipusNev;
  …
Begin
  …
  a.EljC(…); {az a objektum konstruktorának „hívása” a létrehozás kedvéért}
  a.Elj1(…); {az a objektum Elj1 műveletének „hívása”}
  …
```

Megjegyzések:

- Az objektum-***osztály*** (értsd: objektum-típus) definíciójában felhasználható minden korábban definiált típus, így akár „***paraméterezhető***” is.
- Semmi akadálya annak, hogy az ***osztály definícióját*** külön ***include-állományba*** kiemeljük, s ezzel biztosítsuk a könnyű *újrafelhasználás*t (reuseability).

Példa:

Legyen a töredék az alábbi, `ONap.inc` nevű file-ban:

```
{Osztály ONap[4]

  InputParaméterek - ha lennének. Most nincs ilyen.
}
  Type
    TNapK=(Hetfo,Kedd,Szerda,Csutortok,
           Pentek,Szombat,Vasarnap);
    TNapS=Array [1..7] of String[9];
  Const
    NapS:TNapS=('HÉTFŐ','KEDD','SZERDA',
                'CSÜTÖRTÖK','PÉNTEK',
                'SZOMBAT','VASÁRNAP');
  Type
    ONap=Object
      felsKod:TNapK;
      hiba:Boolean;
      Procedure Min(Var minKi:ONap);
      Procedure Max(Var maxKi:ONap);
      Function Szamossag:Word;
      Procedure Kovetkezo(Var x:ONapkovetkezoKi:ONap);
      Procedure Elozo(Var x:ONap)elozoKi:ONap);
      Function Sorszam(Const x:ONap):Word;
```

[4] Az „O” a „T” helyett utalás, hogy „Objektum-Osztály”-ról van szó. Csak „helyi” konvenció!

```
      Procedure ONap(Var x:Word):ONap;
      Procedure Be(Var x:Word);
      Procedure Ki(Const x:Word);
      Function HibasE(Var x:ONap):Boolean;
    End;
    Procedure ONap.Min(Var minKi:ONap);
      {Ef: -
       Uf: minKi=ONap(Hetfo,Hamis)}
    Begin
      minKi.felsKod:=Hetfo; minKi.hiba:=False
    End;
    …
    Function ONap.Szamossag:Word;
      {Ef: -
       Uf: Szamossag=7}
    Begin
      Szamossag:=7
    End;
    Procedure ONap.Kovetkezo(Var kovetkezoKi:ONap);
      {Ef: felsKod<>Vasarnap
       Uf: felsKod=Hetfo ⇒ KovetkezoKi.felsKod=Kedd  ES…}
    Begin
      {Ef-ellenőrzés:}
      If felsKod<>Vasarnap then
      Begin
        hiba:=True; Exit
      End;
      {művelet-törzs:}
      kovetkezoKi.felsKod:=Succ(felsKod)
    End;
  …
```

A felhasználás:

```
Program AProgram;
  Uses …;
  …
  {$i ONap.inc - az ONap osztály beillesztése}
  Var
    tegnap,ma,holnap:ONap;
  …
Begin
  …
  Repeat
    Write('Milyen nap van ma?:'); ma.Be;
  Until not ma.HibasE;
  …
  Writeln('Holnap:'); ma.Kovetkezo(holnap); holnap.Ki;
  …
End.
```

TARTALOM