# „ADATSZERKEZETEK” TANTÁRGY , 1. FÉLÉV
# 1. ELŐADÁS'2009
## (VÁZLAT)

## 1. ADATOK JELLEMZŐI

### 1.1. Azonosító

Az a ***jelsorozat***, *amellyel hivatkozhatunk a tartalmára, amely által módosíthatjuk tartalmát.*
Például:

| | |
|---|---|
| i, j, n, a | – *változó*k nevei (szimbolikus nevek) |
| π, -128, 3.14, "Eredmény=", IGAZ | – *konstans*okat azonosító jelsorozat |

Megjegyzés:

**x := x+1**

*címhivatkozás* *értékhivatkozás*

vagy

Feldolgoz(x) – *eljáráshívásnál nem látszik, hogy x az adat címét vagy értékét jelenti*

**Eljárás** Feldolgoz(**Változó** x:TX)*:* – *Ez már világos beszéd!*

### 1.2. Hozzáférési jog

Adatokat ***módosítani***, illetve értéküket ***lekérdezni***, használni lehet; eszerint egy adat hozzáférés szempontjából négyféle lehet:

| | módosítható | lekérdezhető | |
|---|---|---|---|
| ***Független*** | *Nem* | *Nem* | ***Független*** |
| **Input** | **Nem** | **Igen** | **Konstans** |
| **Output** | **Igen** | **Nem** | **Virgin** |
| **I/O** | **Igen** | **Igen** | **Változó** |
| **Háttértár-adat** | | | **Memória-adat** |

### 1.3. Kezdőérték

A ***születéskor hozzárendelt érték***. Változóknál deklarációban kap*hat* értéket, vagy eleve *van* típushoz rendelt kezdőérték, esetleg speciális '*nem definiált*' érték[1], s így akkor mód van hivatkozás ellenőrzésre is (*virgin*)!

[1] Ez a helyzet az interpretált nyelvek (pl. az ún. szkript-nyelvek) esetében.

### 1.4. Hatáskör

A ***programszöveg azon tartománya, amelyben az adathoz a hozzáférés lehetséges***.
Gondoljunk a blokkstruktúrájú programozási nyelvek egymásba ágyazódó adatdeklarációira! Így beszélhetünk: *globális* és *lokális* (sőt *saját*) adatokról.
A hatáskört „színezi” a **láthatóság** kérdése: ha egy adat azonosítója megegyezik egy olyan adatéval, amelynek hatáskörébe esik, akkor azt elérhetetlenné teszi a saját hatáskörében, hiszen az adott név alatt neki van elsősége. Ez az ún. „**luk a hatáskörben**” jelenség.

Példa:

```
…
Változó
  x:TX
Konstans
  y:TY(…)
…
  [itt az x TX, az y TY típusú]
…
Eljárás E1(Változó x:TX2):
  Változó
    y:TY2
  [itt az x TX2, az y TY2 típusú]
  …
Eljárás vége.
…
  [itt az x TX, az y TY típusú]
…
```

### 1.5. Élettartam

A ***futási időnek az az intervalluma, amelyben az adat azonosítója mindvégig ugyanazt az objektumot jelöli***.
Meggondolandó a az élettartam hatáskörrel való kapcsolata! Globális –azaz a legfelsőbb szinten deklarált–, lokális –pl. egy eljárásban vagy függvényben deklarált– adatok hatásköre és élettartama bizonyos értelemben azonos, amíg a „dinamikus” –azaz futásközben, a programozó döntése szerint születő és megszűnő– adatok esetében már nincs meg az előbbi pontos párhuzam. L. később!)

### 1.6. Értéktípus

Az adatoknak az a tulajdonsága, hogy ***értékei*** mely **halmaz***ból* származnak (*értékhalmaz*) és **tevékenységek***nek* (eljárások, függvények, operátorok) *mely „**készlete**, amely létrehozza, felépíti, lerombolja* és *részekre bontja*”, alkalmazható rá (*asszociált műveletek*).

# 2. AZ ÉRTÉKTÍPUS

## 2.1. Az értéktípusról általánosságban

Összetettség (strukturáltság) szempontjából:

- *skalár* (vagy *strukturálatlan*)
- *összetett* (más szóval *strukturált*)

A*datstrukturálási módok*:

| Strukturálási mód | | → | Adatszerkezet |
|---|---|---|---|
| Keresztszorzat | A×B… | → | Rekord |
| (Megkülönböztetett) egyesítés | A×B×…(C×D∪E×F∪…) | → | Alternatív rekord |
| Halmazképzés | $2^A$ | → | Halmaz |
| Iterált-képzés | $A^*$ | → | Sorozatfélék |

## 2.2. Az asszociált műveletek osztályozása

Az asszociálható műveleteket csoportosítjuk:

- *értékadás* (azonos típusúak közötti adatmozgatás: másolatkészítés, értékmegosztás),

Példa:

```
…
Típus
  TPont=Rekord(x,y,z:Valós)
Változó
  Origó,Egys,Q:TPont
…
  Eltol(Origó,Egys)
  Q:=Origó
…
Eljárás Eltol(Változó p:TPont;Konstans Δ:TPont):
  [értékmegosztás jön létre a formális és az
   aktuális paraméterpárok között]
  p.x:+Δ.x; p.y:+Δ.y; p.z:+Δ.z [2]
Eljárás vége.
…
[értékmásolások az értékadások két oldala között]
```

[2] A $Valami_1$:+$Valami_2$ utasítás szemantikusan ekvivalens a $Valami_1$:=$Valami_1$+$Valami_2$ utasítással.

- *típusátviteli függvények:*
  - *Konstrukciós műveletek* (strukturált érték létrehozása)

    Példa:

```
…
Típus
  TPont=Rekord(x,y,z:Valós)
  TVárakozók=Sor(TEmber)
  TPMut=TPont'Mutató    [l. később]
Változó
  p,q:TPont
  v:TVárakozók
  pm:TPMut
…
  p:=TPont(1.0,0.0,0.0)
  Üres(v)
  Létrehoz(pm,p); q:=TPMut(pm)    [l. később]
…
```

  - *Szelekciós operációk*

    Példa:

```
…
  hossz:=SQRT(p.x^2+p.y^2+ p.z^2)
  vevő:=Első(v)
 …
```

  - *Azonosság* és más *relációk*

    Példa:

```
…
Típus
  TNap=(hétfő,kedd,szerda,csütörtök,péntek,
        szombat,vasárnap) [l. később]
Konstans
  napSzám:Egész(Számosság'TNap) [l. később]
Típus
  TNapNév=Tömb(1..napSzám:Szöveg)
Konstans
  SNapHu:TNapNév=
         ('hétfő','kedd','szerda','csütörtök',
          'péntek','szombat','vasárnap')
  SNapEn=
         ('Monday','Tuesday','Wednesday',
          'Thursday','Friday',
          'Saturday','Sunday')
Változó
  nN:TNapNév
  mn:TNap
  c:Karakter
…
```

```
  Ha c='E' és nN=SNapHu akkor nN:=SNapEn
…
  Ciklus amíg mn≤péntek
…
```

- *Számosság-függvény*

  Példa:

```
…
Típus
  TNap=(hétfő,kedd,szerda,csütörtök,péntek,
        szombat,vasárnap)
Konstans
  napSzám:Egész(Számosság'TNap)
  SNap:Tömb(1..napSzám:Szöveg)=
       ('hétfő','kedd','szerda','csütörtök',
        'péntek','szombat','vasárnap')
…
```

- *Min és Max típusoperátor*, amelyek értelemszerűen csak *rendezett típusok* esetén léteznek. Nem keverendő össze az adatokra (nem típusokra!) alkalmazandó függvényekkel!

  Példa:

```
…
Típus
  TNap=(hétfő,kedd,szerda,csütörtök,péntek,
        szombat,vasárnap)
Konstans
  SNap:Tömb(Min'TNap..Max'TNap:Szöveg)=
       ('hétfő','kedd','szerda','csütörtök',
        'péntek','szombat','vasárnap')
…
```

- *Sorszám- (vagy Rend-) függvény*

  Példa:

```
…
  első:=Sorszám(hétfő) [első:=0]
  utsó:=Sorszám(vasárnap) [utsó:=6]
  A_Kód:=Sorszám('A') [A_Kód:=65]
…
```

- *Be/Ki műveletek* (konverzió az input/output és belsőábrázolás között)

  Példa:

```
…
Változó
  nap:TNap
…
  Be: nap   [nap<Max'TNap]
  Ki: Következő(nap)
…
```

- *Transzformációs* (a típuson értelmezett, a típusra képező függvények)

  Példa:

```
…
Típus
  TPont=Rekord(x,y,z:Valós)
Változó
  kövNap,nap:TNap
  p,q:TPont
…
  kövNap:=Következő(nap)
  táv:=Hossz(p)-Hossz(q)   [Hossz a programozó
                            által definiált fv.]
…
```

## 3. EGYSZERŰ ADATTÍPUSOK

Alábbiakban definiáljuk az „eredendően” szerkezetnélküli ún. ***skalár*** típusokat, megadva ezek *értékhalmaz*át, a hozzátapadó *művelet*ek, *reláció*k halmazát. Két típus „lóg ki” ezek közül. A szöveg (string) és a mutató típus. A ***szöveg típus***t azért soroljuk ide, mert a programozási nyelvek majd mindegyike fölkínálja, s így hozzátartozik az ún. *elemi* (natív) *típusok*hoz, másrészt abban jócskán eltér a későbbiekben tárgyalt összetettektől (s így oda még kevésbé illik), hogy elemeinek típusa nem választható meg szabadon. A ***mutató*** típus valóban szerkezet nélküli, hisz értékhalmaza a memóriacímek egy részhalmaza (ezért tárgyaljuk itt), de az is kétségtelen, hogy oly szorosan tapad valamely összetett típushoz, amely címét tartalmazza, hogy anélkül értelme sem lenne bevezetni.

A tárgyalt típusokról a következőket adjuk meg:

- ***Értékhalmaz***
- ***Kezdőérték***, amit az ilyen típusú objektum létrejövetelekor kap.[3]
- ***Asszociált műveletek***
- (s esetleg) tipikus ***ábrázolása***(i)
- (s néhány esetben) ***példa.***

A rövid leírás kedvéért időnként alkalmazni fogjuk a ***Típ*** jelölést az éppen tárgyalt típusra való hivatkozásra. (Értelemszerűen akkor, amikor a típus nevét a programozó feladata megadni.) Származtatott típus esetén pedig gyakorta a ***TB*** az ún. *bázistípus*ra (amelyből kiindulunk) utal.

[3] És most legkevésbé sem hagyjuk magunkat befolyásolni olyan programozási nyelvektől, amelyben a változók létrejöttét nem kíséri automatikus kezdőértékadás.

## 3.1. Elemi adattípusok

### 3.1.1. Egész

| | |
|---|---|
| *Értékhalmaz* | -32768**..**32767 ⊂ **Z**<br>(*Min'Egész***..***Max'Egész*) |
| *Kezdőérték* | **0** |
| *Műveletek* | **+**, **–**, *****, **Div** (egészosztás), **Mod**, **-** (unáris mínusz), **^** (pozitív egészkitevős hatványozás)<br>**:=**<br>**=**, **<**, **≤**, **≥**, **>**, **≠**<br>**Be:**, **Ki:** |
| *Ábrázolás* | ún. (16-bites) *kettes komplemens kódú* |

Megjegyzés:

A programozási nyelvek többféle értékhalmazzal is felkínálnak efféle típusokat. Pl. a Turbo Pascal megkülönböztet BYTE, SHORTINT (8 bites); INTEGER, WORD (16-bites); LONGINT (24 bites)… Ennek ellenére nem tartjuk fontosnak az algoritmikus nyelvben ezeket megkülönböztetni, annál is kevésbé, mert ha kell, a „szélsőséges értékeire" tudunk hivatkozni a Min'Egész és Max'Egész típusfüggvénnyel anélkül, hogy letennénk a voksunkat bármelyik konkrét értékhalmaz mellett.

### 3.1.2. Valós

| | |
|---|---|
| *Értékhalmaz* | ???..??? ⊂ **R**<br>(*Min'Valós***..***Max'Valós*) |
| *Kezdőérték* | **0.0** |
| *Műveletek* | **+** , **–** , ***** , **/**, **-** (unáris mínusz), **^**<br>**:=**<br>**=**, **<**, ≤, ≥, **>**, ≠<br>**Be:**, **Ki:** |
| *Ábrázolás* | ún. *lebegőpontos ábrázolás* (pontosabb lenne, ha e típust *racionális*nak nevezzük, mert csak racionális számot képes ábrázolni), vagy<br>ún. *pakolt decimális* ábrázolás |

Megjegyzések:

Vegyük észre, hogy ugyanazok a műveleti jelek most –ha hasonló jelentéssel is, de– mégsem egészen ugyanazzal a jelentéssel bírnak. (Polimorfizmus.)

Itt már föl sem vállaltuk az értékhalmaz pontosítását, mivel ez sokkal inkább implementációfüggő, mint az egész számoké.

### 3.1.3. Logikai

| | |
|---|---|
| *Értékhalmaz* | *Hamis*..*Igaz* ⊆ **L**<br>(*Min'Logikai*..*Max'Logikai*) |
| *Kezdőérték* | **Hamis** |
| *Műveletek* | **nem**, **és**, **vagy**<br>**:=**<br>=, <, ≤, ≥, >, ≠<br>**Be:**, **Ki:** |
| *Ábrázolás* | 0 ≡ *Hamis*, –1 ≡ *Igaz* – azaz (valamely) egész típusra visszavezetés;<br>néha 1 ≡ *Igaz* – ekkor 1 bites az ábrázolás |

### 3.1.4. Karakter

| | |
|---|---|
| *Értékhalmaz* | 0..255 -*kód*ú jelek<br>(*Min'Karakter*..*Max'Karakter*) |
| *Kezdőérték* | **Min'Karakter** |
| *Műveletek* | **Karakter(.) – Karakter**: Egész→Karakter<br>**Sorszám(.) – Sorszám**: Karakter→Egész (belső kód)<br>**:=**<br>=, <, ≤, ≥, >, ≠<br>**Be:**, **Ki:** |
| *Ábrázolás* | Valamely kódrendszer szerinti kód, mint előjelnélküli szám. (*Fix bitszámú kód*.) |

Megjegyzés:

Sokfajta kód rendszer létezik (legismertebbek: ASCII, UNICODE). Többségük fix bitszámú (ASCII 8 bites, UNICODE 16 bites), de elképzelhető változó bitszámmal dolgozó is. (L. Huffman-kódolás.)

### 3.1.5. (Absztrakt)Felsorolástípus

| | |
|---|---|
| *Értékhalmaz* | **(**konstans$_1$**,** konstans$_2$**, ... ,** konstans$_N$**)** (*Típ* jelölje a típus azonosítóját)<br>(*Min'Típ*=konstans$_1$..*Max'Típ*=konstans$_N$) |
| *Kezdőérték* | **Min'Típ** vagy **NemDef** |
| *Műveletek* | **Következő(**Típ-*típusbeli érték*), – **Következő**: Típ→Típ **U** {NemDef}<br>**Előző(**Típ-*típusbeli érték*), – **Előző**: Típ→Típ **U** {NemDef}<br>**Sorszám(**Típ-*típusbeli érték*), – **Sorszám**: Típ→Egész<br>**Típ(***egész típusú érték*), – **Típ**: [0..**Sorszám(Max'**Típ)]→Tip<br>**:=**<br>=, <, ≤, ≥, >, ≠ (a *felsorolás sorrendje* egyben *rendezés* is)<br>**Be:**, **Ki:** |

| | |
|---|---|
| *Ábrázolás* | A minimálisan szükséges bitszámú (≈$\log_2$(**Számosság**'Típ)) kód, mint előjelnélküli szám. |

Megjegyzések:

Az értékhalmazban szereplő, ismétlődés nélküli, szimbolikus nevek (a típus konstansai) ***nem lehetnek más típus*** (pl. egy másik felsorolástípus) ***értékhalmazában***, ez a feltétel amiatt szükséges, mert a fordítóprogram így mindig egyértelműen el tudja dönteni a konstans „hovatartozását". Természetesen „…" nem szerepelhet a felsorolásban, mivel a fordítónak nincsenek „előzetes elképzelései" arról, hogy mik lehetnének ott a felsorolásban.

Mindazon típusokat, amelyek értékkészletét konstansainak egyszerű fölsorolásával adhatunk vagy adhatnánk meg ***diszkrét típus***nak hívjuk. Tehát ilyen az `Egész`, a `Logikai`, a `Karakter`, de lehet bármilyen –a program írója által kreált– *absztrakt konstans*okat tartalmazó fenti *absztrakt felsorolástípus* is.

Példa:

```
Típus
  TNap=(hétfő, kedd, szerda, csütörtök, péntek, szombat,
        vasárnap)
Változó
  tegnap,ma,holnap: TNap
Konstans
  ünnepnap: TNap(vasárnap)
…
  Ha ma=Min'TNap akkor  tegnap:=Max'TNap
                 különben tegnap:=Előző(ma)
  Ha ma=Max'TNap akkor  holnap:=Min'TNap
                 különben holnap:=Következő(ma)
  első:=Sorszám(hétfő) [első:=0]
  utsó:=Sorszám(vasárnap) [utsó:=6]
…
```

### 3.1.6. (Rész)Intervallumtípus

Csak ***diszkrét*** típusból származtatható egyszerű típus. Jelöljük a bázistípust TB-vel.

| | |
|---|---|
| *Értékhalmaz* | konstans$_1$..konstans$_2$ ⊆ TB<br>(*Min'Típ*=konstans$_1$..*Max'Típ*=konstans$_2$) |
| *Kezdőérték* | **Min'Típ** vagy **NemDef** |
| *Műveletek* | TB-vel megegyező műveletek (korlátozva persze az értékhalmazra) |
| *Ábrázolás* | TB-vel megegyező ábrázolás |

Példa:

```
Típus
  TNap=(hétfő, kedd, szerda, csütörtök, péntek, szombat,
        vasárnap)
  TMunkanap=hétfő..péntek
  THétvége=szombat..vasárnap
Változó
  tegnap,ma,holnap: TNap
Konstans
  ünnepnap: TNap(vasárnap)
…
  Ha ma=Min'TNap akkor  tegnap:=Max'TNap
                különben holnap:=Előző(ma)
  Ha ma=Max'TNap akkor  holnap:=Min'TNap
                különben holnap:=Következő(ma)
  első:=Sorszám(hétfő) [első:=0]
  utsó:=Sorszám(vasárnap) [utsó:=6???]
…
```

Megjegyzések:

1. Érdekes anomáliára vezet a `Sorszám` és `Típ` típuskonstrukciós függvény következetes bevezetése. Miért?
2. Ha a diszkrétségtől eltekintünk, kiterjeszthető a `Valós`akra is az intervallumtípusképzés. Ez persze csak azzal a többlettel képzelhető el, hogy egy lépésközt is megadunk a származtatáshoz (amelyre vannak persze elvárások).
3. További általánosítás lehetséges: nem első és utolsó elem által meghatározott része egy értékhalmaznak, hanem valamilyen (*predikátum*mal definiált) tulajdonságnak eleget tevő elemeinek részhalmaza.

Példa:

```
Típus
  TTermSzám=Egész
            [Típusinvariáns: i:Egész : i>0]
  TPrímek=TTermSzám
          [Típusinvariáns: n:TTermSzám : Prím?(n)]
          Függvény Prím?(Konstans x:Egész):Logikai
            …
          Függvény vége.
…
```

### 3.1.7. Szövegtípus

| | |
|---|---|
| *Értékhalmaz* | MaxHossz darabnyi jelből álló karakterláncok halmaza ⊂ **Karakter**$^*$ (*Min'Szöveg*..~~*Max'Szöveg*~~) – *alfabetikus* rendezés szerinti első (=üres szöveg); az utolsó erősen reprezentációfüggő, ezért nem definiáljuk. |
| *Kezdőérték* | **''** azaz üres-szöveg (**Min'Szöveg**) |
| *Műveletek* | **+**, **Hossz(.)**, **Balrész(.)**, **Jobbrész(.)**, **Jele(.)**<br>**:=**<br>=, <, ≤, ≥, >, ≠<br>**Be:**, **Ki:** |
| *Ábrázolás* | **Rekord**(hossz:Egész, jel:**Tömb**(1..MaxHossz:Karakter)) – Pascal-stílusú<br>Karakter$^*$ × SzövegVégJel – C-stílusú |

## 3.2. Mutató típusok

Az alcímbeli többes szám jogos, mert valójában tetszőleges (többnyire összetett) típushoz, mint bázistípushoz (`TB`) szervesen tartozhat egy-egy ilyen típus. Egy konkrét mutató típusú objektum ***csak egyfajta*** (nevezetesen `TB`-típusú) ***elemek kezdőcímei***t hordozhatja. E szigorúság oka: az ellenőrizhetőség, biztonságosság.

| | |
|---|---|
| *Értékhalmaz* | Memóriacím, amely valamely TB-típusú elem kezdőcíme, vagy **Sehova**<br>(rendezésnek nincs értelme ⇒ ¬∃*Min'Típ, Max'Típ*) |
| *Kezdőérték* | **Sehova** |
| *Műveletek* | **Lefoglal(Változó** m:Típ, **Konstans** e:TB**)** – az eljárás létrehoz a memóriában egy TB-elemet, amelynek értéke éppen 'e', és a *cím*ét teszi 'm'-be; ha nincs elegendő hely, akkor 'm'-be **Sehova** érték kerül. Elhagyható a kezdőértéket definiáló paraméter, ekkor a TB-beli *iniciális* értékű elem keletkezik.<br>**Típ(Konstans** m:Típ**)**:TB – a függvény az 'm'-beli címnél kezdődő TB-elemet adja vissza értékként<br>**Felszabadít(Változó** m:Típ**)** – az eljárás felszabadítja a memória 'm'-ben lévő címtől kezdődő TB-elemnyi tartományát, majd 'm'-be a **Sehova** érték kerül.<br>**:=**<br>**=, ≠, <, ≤, ≥, >**<br>**Be:, Ki:** |
| *Ábrázolás* | Memóriacím |

Megjegyzések:

A `Lefoglal` művelet fent taglalt szemantikája is indokolja a szigorú típusosság kívánalmát!

A Pascal-beli `Lefoglal` művelet, a `New`, nem foglalkozik kezdőértékadással, sőt a helyfoglalás sikertelenségét sem közli a `Sehova`, vagyis a `Nil` értékkel. A `Felszabadít`, `Dispose` művelet sem törli a mutatót.

Példa:

```
…
Típus
  TBlokk=Tömb(1..MaxM:TElem)
  TBMut=TBlokk'Mutató
  TBlokkok=Tömb(1..MaxN:TBMut)
Változó
  e:TElem;     i:Egész
  b:TBlokk;    bk:TBlokkok;    bm:TBMut
…
  Ciklus i=1-től MaxN Div 2-ig
    Lefoglal(bm)   [bm-be kerül a lefoglalt TBlokk-nyi
                    terület címe, és a bm-nél kezdődő
                    tömb elemei TElem-kezdőértékűek]
    bk(i):=bm      [bk i. eleme a dinamikusan lefoglalt
                    tömb címe; értékmegosztás]
    … [a bk(i) tömbbe értékek kerülnek] …
  Ciklus vége
  b:=TBlokk(bk(1))   [az első blokk b-be]
  bm:=bk(1)          [az első blokk címe bm-be]
  Ciklus i=1-től MaxM-ig
    TBlokk(bm)(i):=e  [TBlokk(bm) bm-nél kezdődő blokk
                       mint tömb,
                       TBlokk(bm)(i) a tömb i. eleme]
  Ciklus vége
  [vajon milyen értékek vannak a bk(2..MaxN) elemekben;
   és mit mondhatunk a TBlokk(bk(2..MaxN)) értékéről?]
…
```

## 4. ÖSSZETETT ADATTÍPUSOK – TÍPUSKONSTRUKCIÓK

Helyesebb összetett típusok helyett ***típuskonstrukciók***ról beszélni, mivel valahány összetevő típusból hozunk létre, konstruálunk egy újat; s az ilyeneket létrehozni képes eszközöket ***típuskonstrukciós eszközök***nek hívni.

Többnyire *nem* magától értetődő az ilyen struktúrák *rendezése*, ezért sem a `Min'`, sem a `Max'` típusfüggvényt nem jelezzük. (Megjegyezzük: természetesen nem elképzelhetetlen – sőt!–, hogy utólag valamilyen rendezést rájuk is definiáljuk. Erről később még szó esik.)

A korábbi szokásos mondanivalók mellé bekerül a „konstrukció”, amelyben tisztázzuk az alkalmazás szintaxisát.

Rövidítés miatt a bázistípusokat sorszámozzuk és így jelöljük: $TB_1$, $TB_2$, …

### 4.1. Összetett típusok osztályozásai

A bázistípusok sokfélesége szerint –

- *Heterogén* (rekord, alternatív rekord)
- *Homogén* (sorozatfélék: tömb, lista…, halmaz; rekurzív típusok: lista, fa; gráf…)

***Rákövetkezési reláció*** az elemei között –

- *Nincs* (értelmetlen: rekord, alternatív rekord; lehetne, de nincs: halmaz)
- *Egyértelmű*, kivéve a szélső elemeket (sorozatfélék: tömb, lista,…)
- *Többszörös* (rekurzív típusok közül a fa; gráf)

### 4.2. Rekord

| | |
|---|---|
| *Konstrukció* | **Rekord(**mező$_1$**:** TB$_1$**,** mező$_2$**:** TB$_2$ …**)** |
| *Értékhalmaz* | TB$_1$×TB$_2$×… |
| *Kezdőérték* | Az egyes komponensekhez tartozó kezdőérték. |
| *Műveletek* | **Típ(Konstans** m$_1$**:** TB$_1$**,** m$_2$**:** TB$_2$ …**)** – létrejön egy Típ típusú konstans m$_1$, m$_2$… mezőértékekből (konstrukciós operáció)<br>objTip**.mező$_i$**:TB$_i$ – a „szokatlan” szintaxisú művelet értéke az adott Típ típusú objektum (objTip) mező$_i$ mezőjének értéke… (szelekciós operáció)<br>objTip**.mező$_i$**:TB$_i$**:=**e$_i$ – a „szokatlan” szintaxisú értékadás eredménye az adott Típ típusú objektum (objTip) mező$_i$ mezőjének értékül adja e$_i$-t…<br>**:=**<br>**=**, **≠**<br>**Be**:, **Ki**: |
| *Ábrázolás* | A mezők folytonos memóriaterületre képezve, a felsorolás sorrendjében. |

### 4.3. Alternatív rekord

Olyan rekordféléről van szó, amelynek valamely mezőjétől (mezőitől) függ további mezőinek típusbesorolása.

| | |
|---|---|
| *Konstrukció* | **Rekord(**<br>mező$_1$**:** TB$_1$**,** … mező$_K$**:** TB$_K$**,**<br>**Alternatívák**<br>felt$_1$(mező$_1$,…) **esetén (**mező-k sorozata**),**<br>…<br>felt$_m$(mező$_1$,…) **esetén (**mező-k sorozata**)**<br>**Alternatívák vége)** |
| *Értékhalmaz* | TB$_1$×…TB$_K$×…∪$_{(i=1..m)}$TB$_{i,1}$×…TB$_{i,k_i}$ |
| *Kezdőérték* | A nem egyértelműsége miatt **NemDef**. |
| *Műveletek* | A rekordhoz hasonlóan. |
| *Ábrázolás* | A mezők folytonos memóriaterületre képezve, a felsorolás sorrendjében, a |

| | |
|---|---|
| | hosszat a leghosszabb alternatívával számolva. |

Példa:

```
…
Konstans
  ffi: Egész(1)
  nő : Egész(2)
Típus
  TDátum=...
  TNem=ffi..nő
  TKontroll=Függvény(TNem,Dátum,0..999):0..9
  Függvény Kontroll(Konstans n:TNem
                             d:TDátum
                             x:0..999):0..9
    [Kontroll: egy TKontroll típusú konkrét függvény]
    …
  Függvény vége.
Típus
  TSzemSzám=Rekord(
    nem:TNem
    szülidő:TDátum
    sorszám:Egész
    ellenőr:TKontroll [itt csak olyan fv-típus képzelhető
                       el, amely értelmezési tartománya:
                       TNem×TDátum×Egész; értékkészlete
                       most nincs megkötve])
  TSzemély=Rekord(
    szsz:TSzemSzám
    név:Szöveg
    Alternatívák
      nem=nő  esetén (lnév :Szöveg)
      nem=ffi esetén (katsz:Egész)
    Alternatívák vége)
Konstans
  Jézus:TSzemély((ffi,
                  1.12.25,
                  123,
                  Kontroll) [szsz mező tartalma],
                  'Jézus Krisztus' [név mező],
                  (123456) [nem=ffi⇒katsz mező])
…
```

### 4.4. (Hatvány-)Halmaz

Csak *diszkrét* (többnyire nagyon is korlátozott számosságú) típusnak definiálhatjuk a hatvány-halmaz-típusát.

| | |
|---|---|
| *Konstrukció* | **Halmaz(**TB**)**<br>(*Min'Típ*=∅**..***Max'Típ*=TB-értékhalmaz, a tartalmazás alapján rendezve) |
| *Értékhalmaz* | TB* |
| *Kezdőérték* | **Üres** halmaz |
| *Műveletek* | ∈ (eleme), ∩ (metszet), ∪ (egyesítés), \ (különbség), ∅ vagy **Üres** (üres halmaz létrehozás), **Üres?** (logikai értékű függvény)<br>**:=**<br>=, < (⊂), ≤ (⊆), ≥ (⊇), > (⊃), ≠<br>**Be**:, **Ki**: |
| *Ábrázolás* | **Tömb**(TB:Logikai) – azaz bitvektor az adott elem tartalmazása vagy nem tartalmazása szerint;<br>**Lista**(TB) – tartalmazott elemeinek felsorolása [l. később] |

Példa:

```
…
Típus
  TÓra=0..23
  TNap=Halmaz(TÓra)
Változó
  ma,holnap:TNap
Konstans
  munkanap:TNap(7..12,14..20)
…
  Üres(ma)   [ma nincs kötött program ⇒ ma „szabad”]
  Ha Üres?(holnap) akkor ma:=munkanap
…
```

### 4.5. Tömb

| | |
|---|---|
| *Konstrukció* | **Tömb(**TIndex **:** TElem**)** |
| *Értékhalmaz* | TElem$^{\text{Számosság(TIndex)}}$ |
| *Kezdőérték* | TElem típus kezdőértékei |
| *Műveletek* | objTip**(**i**)** (a Típ típusú objTip tömb i. TElem típusú eleme),<br>objTip**(**i**):=**e (a Típ típusú objTip tömb i. eleme legyen 'e' értékű)<br>**:=**<br>**=,** ≠<br>**Be**:, **Ki**: |
| *Ábrázolás* | Az elemek folytonos memóriaterületre képezve, növekvő index sorrendben. (L. később) |

# 5. SOROZATTÍPUSOK

***Sorozattípus***nak hívjuk azt a típust, amely

1. ***homogén*** (azaz elemei egyetlen bázistípushoz tartoznak),
2. ***egyértelmű rákövetkezési reláció*** van elemei között (az adott elemét legfeljebb egy elem követheti),
3. ***egyetlen*** olyan eleme van, amelyet ***nem előz meg*** másik, s ***egyetlen*** olyan, amelyet ***nem követ***.

## 5.1. Sorozatműveletek

Nagyon sokféle sorozattípus alkotható meg, mégha nem különböztetjük meg a különböző bázistípusúakat egymástól (azaz a *strukturálisan azonosak*at). Jellegzetes művelet-együttest rendelve hozzájuk kapjuk a későbbiekben tárgyalt egyes, fontos alosztályait.
A legfontosabb műveletei, halmozva, a következők:

| Művelet | Tevékenység-leírás |
|---|---|
| Üres | Létrehoz, elemek nélkül. |
| Létrehoz | Létrehoz, struktúrától függő elemekkel. |
| Üres?/Teli? | Ellenőrzi, hogy van-e eleme / bővíthető lenne-e? |
| ElemSzám | Hány eleme van? |
| Beilleszt | Struktúrától függő helyre új elemet illeszt. |
| Kihagy | Struktúrától függő helyről elemet hagy el. |
| Első/Utolsó | Első / utolsó elemének értékét adja. |
| Elejéről/Végéről | Leválasztja a sorozat első / utolsó elemét, értékét is visszaadja. |
| ElsőUtániak/UtolsóElőttiek | Eldobja az első / utolsó elemet. |
| Elejére/Végére | A sorozat első eleme elé / utolsó eleme mögé illeszt egy újat. |
| Elem | Struktúrától függően meghatározott elemének értékét adja vissza. |
| ElemMódosít | Struktúrától függően meghatározott elemének új értéket ad. |
| Elsőre/Utolsóra | A struktúra első / utolsó elem lesz az aktuális (ha volt ilyen). |
| Előzőre/Következőre | A struktúra aktuális eleme (ha volt ilyen) legyen az eddigit megelőző / követő. |

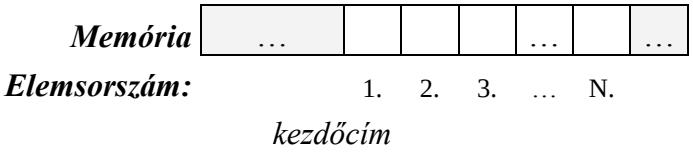
Aszerint, hogy mely műveleteket engedjük meg, egy sorozatfélénél beszélhetünk az alábbi típuskonstrukciókról. Nem jelöljük az „altípus-osztályokat”, továbbá a tevékenységeket nem a „hagyományos” nevükön említjük.

| Típuskonstrukció | Tevékenységhalmaz |
|---|---|
| Tömb | (Létrehoz, ElemSzám,) Elem, ElemMódosít |
| Lista | Üres, Üres?, Teli?, Beilleszt, Kihagy, Elsőre, Utolsóra, Előzőre, Következőre, Elem, ElemMódosít |
| Sor | Üres, Üres?, Teli?, ElemSzám, Első, Elejéről, Végére |

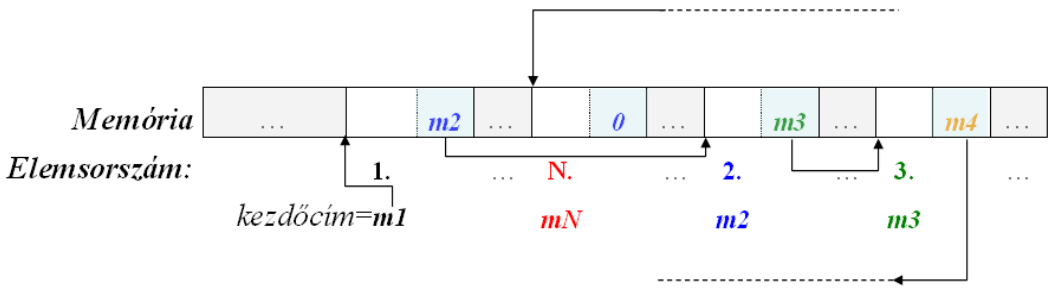
| | |
|---|---|
| Verem | Üres, Üres?, Teli?, ElemSzám, Első, Elejére, Elejéről |
| Táblázat | Üres, Üres?, Teli?, ElemSzám, Elem, ElemMódosít… |
| InputSzekvenciálisFile | Létrehoz, Üres?, Elejéről |
| OutputSzekvenciálisFile | Üres, Üres?, Teli?, Végére |
| DirektFile | Üres, Létrehoz, Üres?, Teli?, ElemSzám, Elem, ElemMódosít… |
| AsszociatívFile | Üres, Üres?, Teli?, ElemSzám, Elem, ElemMódosít… |

### 5.2. Sorozatok ábrázolásának lehetőségei

***Folytonos ábrázolás***, amikor az elemeket a memóriában a *kezdőcímtől szorosan egymásután* helyezzük el.




***Láncolt ábrázolás***, amelynél az elemek a memóriában elszórva helyezkednek el, a rákövetkezést mutatóval biztosítjuk.




***Blokkolt ábrázolás*** ötvözi az előbbi kettőt úgy, hogy láncot hoz létre az elemek egy adott számú elemet tartalmazó tömbjéből.

## 6. A MODUL MINT A TÍPUSMEGVALÓSÍTÁS KERETE

### 6.1 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***” ***megvalósítások***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 („6.3. 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 („7. A modulfogalom és a Pascal nyelv” fejezetben), hogy az általunk használt lingua franca, a Pascal programozási nyelv, milyen lehetőséget ad mindehhez.

### 6.2. 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*):<br>**Asszociált műveletek:**<br>*Operáció(értelmezési tartomány): Értékkészlet*<br>*…*<br>**Axiómák:**<br>**Jelölések:**<br>• *objektum: Típus(paraméterek) =*<br>*értelmező megjegyzés*<br>• *…*<br>**1°** *… az axióma informális szövege ...*<br>*a formalizált axióma*<br>**2°** *…* | *A definiálandó típuskonstrukció „alkalmazási sablona” (feje).*<br>*Operációinak szignatúrája (értelmezési tartománya, értékkészlete)*<br>*Az operációk kapcsolatát leíró axiómák, és a bennük fölhasznált objektumok „deklarációi”.*<br>*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[4]

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.

*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 és 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.*

[4] A konstansokat 0-változós függvényeknek tekintjük.

*Bizonyítás:*
*Nyilvánvaló.*
**8°** A felsorolásban előbb szereplő kisebb, mint bármely későbbi.
*a<b ⇔ Sorszám(a)<Sorszám(b)*[5]*…*

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°** axiómával is.
- Lehetséges, hogy a ***Létrehoz*** és ***Lerombol*** műveletekre a későbbiek során ***explicite nem*** lesz szükségünk. Ez a helyzet az ún. *statikus* adatoknál, amelyekre az a jellemző, hogy a deklarációk által *automatikusan* jönnek létre, s így a programozónak nincs „gondja” sem a létrehozással, sem a lerombolással. (Érdemes meggondolni, mikor kerül sor e két művelet végrehajtására a *blokk-struktúrájú nyelvek –pl. a Pascal–* esetén.)
- 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 feltételeül 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” következményállításként illeszthetjük a rendszerbe. (Most nem tesszük meg.)

### 6.3. 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]
```

[5] `<:TNap×TNap→Logikai` rendezés visszavezetése `<:Egész×Egész→Logikai` rendezésre.

```
  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 értelmesen 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 ezekkel a 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(Konstans x:TNap):TNap
    Ef: x≠Vasárnap [v.ö. a 3. axiómával]
    Uf: Következő(Hétfő)=Kedd ∧ … [v.ö. a 6. axiómával]
  Függvény Előző(Konstans 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(Konstans 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;
   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 nyúlni (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:
  $\text{operáció}_{\text{Exportmodul}}.\text{Ef} \Rightarrow \text{operáció}_{\text{Modul}}.\text{Ef} \wedge \text{operáció}_{\text{Modul}}.\text{Uf} \Rightarrow \text{operáció}_{\text{ExportModul}}.\text{Uf}$.)

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)  [v.ö. a 7. axiómával]
     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(Konstans x:TNap):TNap
  Ef: x.felsKód≠6 [6] [azaz a 7. axióma szerint nem „Vasárnap”]
  Uf: Következő.felsKód=x.felsKód+1 [v.ö. a 6. és 7. axiómával]
  Ha felsKód=6
     akkor    Következő.hiba:=Igaz
     különben Következő.felsKód:=felsKód+1
Függvény vége.
Függvény Előző(Konstans x:TNap):TNap
  Ef: felsKód≠0
  Uf: Előző.felsKód=x.felsKód-1
  …
Függvény vége.
Függvény Sorszám(Konstans x:TNap):Egész
  Ef: –
  Uf: Sorszám=x.felsKód [v.ö. a 7. axiómával és az Állítással]
  …
Függvény vége.
Függvény TNap(Konstans 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
  [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: Kisebb(x,y) ⇔ x.felsKód<y.felsKód [v.ö. a 8. axiómával]
  …
Operátor vége.
Operátor Be(Változó x:TNap):
  Másnéven Be:x
  Ef: Input∈{'hétfő','kedd','szerda'…}
  Uf: Input='hétfő' ⇒ x=Hétfő ∧ …
  …
Operátor vége.
```

[6] Persze szebb lenne így írni: …≠Vasarnap.felsKód.

```
  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
   felsKód:=0; hiba:=Hamis
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 eredményre vezet, akkor persze állítódjon át garantáltan Igaz-ra.
- A kisebb operátor utófeltétele az ExportModul-beli specifikáció „elvárt” párja. Ha azonban a specifikációhoz nyúlunk vissza ($8^o$ axióma), akkor jóval egyszerűbbet kapunk: `Kisebb(x,y)` ⇔ `x.felsKód<y.felsKód`.
- Vegyük észre: az ***inicializálás***ban éppen az **$1^o$** 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.

## 7. A MODULFOGALOM ÉS A PASCAL NYELV

Amire mód van: *algoritmikus specifikáció* és a *megvalósítás*; amire nincs: az algebrai specifikáció. Többféle (Turbo/Free) 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 fej-

lesztendő 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**-öket kell definiálni. Az átalakítás persze igencsak mechanikus, ahogy az alábbi példa mutatja:

| **Algoritmikus** | ⇒ | **Pascal** |
|---|---|---|
| **Operátor** Min:TNap<br>  **Másnéven** Min'TNap<br>  Min:=Hétfő<br>**Operátor vége.** | ⇒ | **Procedure** Min(**Var** minKi:TNap);<br>**Begin**<br>  minKi:=Hetfo<br>**End;** |

Érdemes tudni, hogy

1. a FreePascal-ban az előbbi korlátozást már feloldották, ui. record-értékű függvény definiálható;
2. a Min'TNap / Max'TNap implementálása elhagyható, ui. létezik a típusokhoz egy **Low** / **High** típusargumentumú függvény. Így a Min'TNap / Max'TNap kódolása történhet ily módon is:

| **Algoritmikus** | ⇒ | **FreePascal** |
|---|---|---|
| …Min'TNap…<br>…Max'TNap… | ⇒ | …Low(TNap)…<br>…High(TNap)… |

### 7.1. Unit

- A unit szintaxisa igazodik a Pascal nyelvhez, így túl sok magyarázatra nincs szükség. Annyit elöljáróban: 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. Ezért sokszor be kell vezetni egy további műveletet ***inicializálás*** céllal, és rá kell bízni a programozóra, hogy a deklaráció után és az első felhasználás előtt használja is.
- Mivel ***nincs mód paraméterezésre***, ezért ellenáll mindenféle „általánosításnak”. A fenti kommentben levő InputParaméterek csak a fejlesztőnek szolgáltatnak információt arról, hogy mik azok a fogalmak, amelyek, ha lehetséges volna, paraméterként funkcionálhatnának.
- A ***unit neve*** és ***a befoglaló fájl 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-ökkel helyettesítettük. (FreePascal-ban szebben is megvalósíthatjuk. Gyakorlásul tegyük is meg!)

Példa:

```
Unit TNapUnit;{InputParaméterek}
 Interface
  Type
    TNapK=0..6;
    TNap=Record felsKod:TNapK; hiba:Boolean End;
  Const
    Hetfo:TNap=(felsKod:0;hiba:False);
    Kedd:TNap=(felsKod:1;hiba:False);
    …
  {ki tudja, h. a Pascal a deklarált adatnak milyen kezdőértéket juttat, ezért plusz
   műveletként az alábbit az asszociált műveletekhöz hozzávesszük:}
  Procedure Inic(Var x:TNap);
    {Ef: –
     Uf: x=TNap(0,Hamis)}
  Procedure Min(Var minKi:TNap);
    {Ef: –
     Uf: minKi=TNap(0,Hamis)}
  …
  Function Szamossag:Word;
    {Ef: –
     Uf: Szamossag=7}
  Procedure Kovetkezo(Const x:TNap; Var kovetkezoKi:TNap);
    {Ef: x.felsKod<>6
     Uf: kovetkezoKi.felsKod=x.felsKod+1}
  …
  {az alábbira nincs szükség, hiszen az értékadást ismeri 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=0  ⇒  Output='HÉTFŐ'  ES  …}
  Function HibasE(Var x:TNap):Boolean;
 Implementation
  Type
    TNapS=Array [TNapK] of String[9];
```

```
  Const
    NapS:TNapS=('HÉTFŐ','KEDD','SZERDA',
                'CSÜTÖRTÖK','PÉNTEK',
                'SZOMBAT','VASÁRNAP');
  Procedure Inic(Var x:TNap);
    {Ef: -
     Uf: x=TNap(0,Hamis)}
  Begin
    x.felsKod:=0; x.hiba:=False
  End;
  Procedure Min(Var minKi:TNap);
    {Ef: -
     Uf: minKi=TNap(0,Hamis)}
  Begin
    minKi.felsKod:=0; minKi.hiba:=False
  End;
  …
  Function Szamossag:Word;
    {Ef: -
     Uf: Szamossag=7}
  Begin
    Szamossag:=7
  End;
  Procedure Kovetkezo(Const x:TNap; Var kovetkezoKi:TNap);
    {Ef: x.felsKod<>6
     Uf: kovetkezoKi.felsKod=x.felsKod+1}
  Begin
    With x do
    Begin
      {Ef-ellenőrzés:}
      If felsKod=6 then
      Begin
        kovetkezoKi.hiba:=True; Exit
      End;
      {művelet-törzs:}
      kovetkezoKi.felsKod:=felsKod+1
    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` a `TNap` „belső struktúrája”…)
- A `TNap` típus felhasználását a következő példa mutatja.

  Példa:

```
Program AProgram;
  Uses …,TNapUnit,….;
  Var
    tegnap,ma,holnap:TNap;
  …
Begin
  …
  Inic(ma); {most elhagyható lenne; miért?}
  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(Konstans x:TNap):TNap
  Ef: x≠Vasárnap
  Uf: Következő(x)= Következő(érték) [érték=TNapK(x)!]
  Ha érték=Vasárnap akkor   Következő:=NemDef
                    különben Következő:=Következő(érték)
Függvény vége.
```

Figyelem: itt semmi rekurzió nincs. Hiszen a definiálandó `Következő` függvény paraméterének a típusa `TNap`, amíg a felhasznált `Következő`é `TNapK`.

A Pascal-megfeleltetés úgyszólván problémamentes:

```
Unit TNapUnit;
```

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

### 7.2. Kódtöredék automatikus beillesztése

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

Példa:

Legyen az alábbi töredék a `TNap.inc` nevű fájlban:

```
{TNap
  InputParaméterek - ha lennének. Most nincs.
}
  Type
    TNapK=0..6;
    TNap=Record felsKod:TNapK; hiba:Boolean End;
    TNapS=Array [TNapK] of String[9];




  Const
```

```
    NapS:TNapS=('HÉTFŐ','KEDD','SZERDA',
                'CSÜTÖRTÖK','PÉNTEK',
                'SZOMBAT','VASÁRNAP');
  …
  Procedure Inic(Var x:TNap);
    {Ef: -
     Uf: x=TNap(0,Hamis)}
  Begin
    x.felsKod:=0; x.hiba:=False
  End;
  …
  Procedure Kovetkezo(Const x:TNap; Var kovetkezoKi:TNap);
    {Ef: x.felsKod<>6
     Uf: kovetkezoKi.felsKod=x.felsKod+1}
  Begin
    With x do
    Begin
      {Ef-ellenőrzés:}
      If felsKod<>6 then
      Begin
        kovetkezoKi.hiba:=True; Exit
      End;
      {művelet-törzs:}
      kovetkezoKi.felsKod:=x.felsKod+1
    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.

### 7.3. 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).[7] 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;
    …
    Function TipusNev.Fv1(…):TFv1;
    …
    Begin
      …
    End;
  …
  Var
    a,b,c:TipusNev;
  …
```

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

```
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 (reusability).

Példa:

Legyen a töredék az alábbi, `ONap.inc` nevű fájlban:

```
{Osztály ONap[8]

  InputParaméterek – ha lennének. Most nincs ilyen.
}
  Type
    TNapK=0..6;
    TNapS=Array [TNapK] of String[9];
  Const
    NapS:TNapS=('HÉTFŐ','KEDD','SZERDA',
                'CSÜTÖRTÖK','PÉNTEK',
                'SZOMBAT','VASÁRNAP');
  Type
    ONap=Object
      felsKod:TNapK;
      hiba:Boolean;
      {ki tudja, h. a Pascal a deklarált adatnak milyen kezdőértéket juttat,
        ezért plusz műveletként hozzávesszük:}
      Constructor Inic(Var x:TNap);
      Procedure Min(Var minKi:ONap);
      Procedure Max(Var maxKi:ONap);
      Function Szamossag:Word;
      Procedure Kovetkezo(Var x:ONap; kovetkezoKi:ONap);
      Procedure Elozo(Var x:ONap; elozoKi:ONap);
      Function Sorszam(Const x:ONap):Word;
      Procedure ONap(Var x:Word):ONap;
      Procedure Be(Var x:Word);
      Procedure Ki(Const x:Word);
      Function HibasE(Var x:ONap):Boolean;
    End;
```

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

```
    Constructor ONap.Inic;
      {Ef: -
       Uf: x=TNap(0,Hamis)}
    Begin
      felsKod:=0; hiba:=False
    End;
    Procedure ONap.Min(Var minKi:ONap);
      {Ef: -
       Uf: minKi=ONap(0,Hamis)}
    Begin
      minKi.felsKod:=0; minKi.hiba:=False
    End;
    …
    Function ONap.Szamossag:Word;
      {Ef: -
       Uf: Szamossag=7}
    Begin
      Szamossag:=7
    End;
    Procedure ONap.Kovetkezo(Var kovetkezoKi:ONap);
      {Ef: felsKod<>6
       Uf: kovetkezoKi.felsKod=felsKod+1}
    Begin
      {Ef-ellenőrzés:}
      If felsKod=6 then
      Begin
        hiba:=True; Exit
      End;
      {művelet-törzs:}
      kovetkezoKi.felsKod:=felsKod+1
    End;
  …
```

A felhasználás:

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

```
End.
```

# TARTALOM