Programozási tételek általánosítása 1.

Szlávi Péter
szlavip@elte.hu
2015

Tartalom

1. Cél+kiinduló pont
2. Az általánosítás lehetőségei
   a) Az algoritmizálás absztrakciójának mélyítése
   b) A kódolás „absztrakciójának” mélyítése
3. Algoritmizálás/kódolás absztrakciójának mélyítése
   a) Sorozatok – Tömbáltalánosítás
   b) To be continued…

Cél+kiinduló pont

- Általánosítás
  - **elvekben** – algoritmikusan (részben a specifikációt is érintve)
  - **gyakorlatban** – programozási nyelvben (most Pascalban)
- Várható előnyök
  - **oktatásban**: absztrakciós készség fejlődése
  - **szakmában**: újrafelhasználhatóság miatt a fejlesztési eredményesség és időhatékonyságának a növekedése
- „Kiinduló pont” – programozási tételek
  - **Állítás**: a specifikációt az algoritmus kielégíti
  - Mint **függvény**: *TételNév(bemenet):Kimenet*
  - Tipikus paraméter-kellékek: **tömbök**

Az általánosítás lehetőségei

- **Algoritmizálás** során

  a **nyelvi** eszköz – a modul-fogalom kettőse:

  - **Exportmodul** –

    a „kívülről” látszó fogalmak felsorolása

```
ExportModul ModulNév(paraméterek):
  típus-/konstans-definíciós „fejsorok”
  eljárás-/függvény-definíciós fejsorok
  operátor-definíciós fejsorok
Modul vége.
```

  - (Reprezentációs/Implementációs) **Modul** –

    a típus ábrázolása
    és a műveleteknek
    ábrázolás-függő
    megvalósítása

```
Modul ModulNév(paraméterek):
  Reprezentáció
    típus-/konstans-definíciók
  Implementáció
    eljárás-/függvény-definíciók
    operátor-definíciók
  Inicializálás
    inicializáló műveletek
Modul vége.
```

Kérdés:

Mi értelme a modul-duplázásnak?

Az általánosítás lehetőségei

- Kódolás során

  a nyelvi eszköz – (Free)Pascal unit fogalma

```
Unit UnitNév(paraméterek);
Interface
  típus-/konstans-definíciók (fejsor)
  eljárás-/függvény-definíciós fejsorok
  operátor-definíciós fejsorok
Implementation
  eljárás-/függvény-definíciók
  operátor-definíciók
Begin
  inicializáló műveletek
End.
```

↓

Következmény: „látszik”, azaz ki lehet használni a definíció részleteit, vissza lehet élni e többlet tudással.

Más működésű inicializálás.

És mi ezekkel a baj?!?

```
ExportModul ModulNév(paraméterek):
  típus-/konstans-definíciós „fejsorok”
  eljárás-/függvény-definíciós fejsorok
  operátor-definíciós fejsorok
Modul vége.
```

```
Modul ModulNév(paraméterek):
  Reprezentáció
    típus-/konstans-definíciók
  Implementáció
    eljárás-/függvény-definíciók
    operátor-definíciók
  Inicializálás
    inicializáló műveletek
Modul vége.
```

Az általánosítás lehetőségei

- Kódolás során **De!!!**

  a nyelvi eszköz – (Free)Pascal unit fogalma

```
Unit UnitNév(paraméterek);
{$mode objfpc}{$H+}
Interface
  osztály-sablon-definíció
  [eljárás-/függvény-definíciós fejsorok]
Implementation
  eljárás-/függvény-definíciók
Begin
  inicializáló műveletek
End.
```

```
ExportModul ModulNév(paraméterek):
  típus-/konstans-definíciós „fejsorok”
  eljárás-/függvény-definíciós fejsorok
  operátor-definíciós fejsorok
Modul vége.
```

```
Modul ModulNév(paraméterek):
  Reprezentáció
    típus-/konstans-definíciók
  Implementáció
    eljárás-/függvény-definíciók
    operátor-definíciók
  Inicializálás
    inicializáló műveletek
Modul vége.
```

absztrakt osztálynév | típusparaméter(ek)

```
Type
  Generic oNév<típPar>=Class(TObject)
    típus-, változó- (attribútum-) definíciók
    eljárás-/függvény-definíciós fejsorok
    operátor-definíciós fejsorok
  End;
```

Felhasználása a hívó programban (specializálás):

```
  Type
    konkONév=Specialize oNév<konkTíp>
```

konkrét osztálynév | absztrakt osztálynév | konkrét típus

Az általánosítás lehetőségei

- Kódolás során **De!!!**

  a nyelvi eszköz – (Free)Pascal unit fogalma

```
ExportModul ModulNév(paraméterek):
  típus-/konstans-definíciós „fejsorok”
  eljárás-/függvény-definíciós fejsorok
  operátor-definíciós fejsorok
Modul vége.
```

```
Modul ModulNév(paraméterek):
  Reprezentáció
    típus-/konstans-definíciók
  Implementáció
    eljárás-/függvény-definíciók
    operátor-definíciók
  Inicializálás
    inicializáló műveletek
Modul vége.
```

```
Unit UnitNév(paraméterek);
{$mode objfpc}{$H+}
Interface
  osztály-sablon-definíció
  [eljárás-/függvény-definíciós fejsorok]
Implementation
  eljárás-/függvény-definíciók
Begin
  inicializáló műveletek
End.
```

Objektum létrejövetele és kezdőértéke (inicializálás):

1. A deklarációkor még nem jön létre (csak egy címnyi helyet kap), a konstruktor meghívásával hozható létre.
2. Létrehozáskor a kezdőértéke „nulla” (a reprezentált bájtok 0 értékűek). Ha speciális kezdőértékű, akkor definiálni kell olyan saját konstruktort (is), amely a mezőket megfelelően beállítja.

```
Generic oNév<típPar>=Class(TObject)
  …
  Constructor Create(paraméterek);
  …
End;
```

Algoritmizálás/kódolás absztrakciójának mélyítése

Feladatok:

Letölt

- Készítsen **absztrakt** vermet (ti. verem-**sablon**t) `Verem_Abs_unit` unitba csomagolva, a saját vagy a mellékelt (String-elemű) verem unitból kiindulva!

Letölt

- Próbálja ki a unitját legalább kétféle konkrét veremre! Azaz írjon egy egyszerű programot, amely képes felépíteni (fájlos inputból) egy-egy adott típusú vermet, s elemeit kiolvasgatva kilistázni azokat. (Lehet az elemtípus: `Integer` és `String`.)
- Fogalmazza meg tapasztalatait a Free Pascal `Generic+Specialize` fogalompárról! (Pl. az `oNév.Create` metódus automatikusan létrejön, csak típus paramétere lehet a sablonnak, konstans, például méret, nem; egy osztály példányát használat előtt létre kell hozni az osztályához tartozó `Create` osztály-függvénnyel stb.)

Algoritmizálás/kódolás absztrakciójának mélyítése

- Tömbök helyett **sorozat**ok

  Sorozat: adattípus –

  értékhalmaz:
  - azonos típusú elemek együttese ($s \in H^*$)

  műveletek:
  - létrehozás/lerombolás
  - elemszám (hossz(s)),
  - elemérték-kiolvasás (pl. első(s), következő(s), dik(s,i)…, vége?(s), utolsó?(s), …),
  - elemérték-módosítás (pl. beilleszt(s,e), módosít(s,e), dik(s,i,e)…)

  leszármazottai (művelethalmaz leszűkítései):
  - tömb,
  - (szekvenciális) fájl(ok),
  - lista,
  - …

- **Cél**: a tételek olyan algoritmikus átírása, hogy **ne** legyenek érzékenyek a bennük előforduló **sorozatok milyenségére**.

Sorozatok – Tömbáltalánosítás

- Tömb-típuskonstrukció:
  indextípus + elemtípus
  - Exportmodulja:

```
ExportModul Tömb(Típus TIndex: Típus TElem):
  Eljárás
    Létrehoz(Változó t:Tömb)
    Lerombol(Változó t:Tömb)
  Függvény
    ElemSzám(Konstans t:Tömb):Egész
  Operátor
    ElemÉrték(Konstans t:Tömb, i:TIndex):TElem
      Másnéven t(i)
    ElemMódosít(Változó t:Tömb, Konstans i:Tindex, e:TElem)
      Másnéven t(i):=e
    AzonosE(Konstans t,tt:Tömb):Logikai
      Másnéven t=tt
    LegyenEgyenlő(Változó t:Tömb, Konstans tt:Tömb)
      Másnéven t:=tt
  …
Modul vége.
```

  - Kérdés:

    Miért kell az operátoroknál a **Másnéven** bejegyzés?

  - Megjegyzés:

    Később, ahol nyilvánvaló az operátor szintaxisa (pl. infix), ott eltekintünk ettől a bejegyzéstől.)

Sorozatok – Tömbáltalánosítás

- Elvárások az **indextípusok**tól (`TI`):
  - Célok és következmények:
    1. a tömb egyes **elemei értékének elérése**
    2. a tömb egyes **elemeinek módosítása (címének elérése)**
    3. mivel a tömb **sorozatféle** → a feldolgozása **ciklus**sal történik
    4. mivel a tömb **indexelt** sorozat → a szervezéshez kell az **értékadás**, **értékazonosság** operátor, **rendezési** reláció, **következő**, **előző**, **minimum**, **maximum** függvények, azaz az indextípus egy ún. **diszkrét típus** (pl. a `THét` felsorolás típus)
    5. mivel erre építhető a tömb reprezentálása (az ún. **címfüggvény**en keresztül) → **sorszám** konverziós függvény (index→természetes szám)
    6. praktikus okok miatt célszerű a **kiírás** műveletet is megengedni; gondoljon egy tömb ízléses („sablonos”) kiírásának szokásos kóddarabjára:

```
…
Writeln('Valami a tömb egészére utaló kiírás');
For i:=1 to N do
Begin
  Writeln(i:3,'.:',X[i]);
End;
…
```

Sorozatok – Tömbáltalánosítás

- Elvárások az **indextípusok**tól (`TI`):
  - Alapeset: eleve definiált diszktét típusból készítünk index-típust

```
ExportModul TI:
  Konstans
    Min,Max:TI
    Számosság:Egész
  Eljárás
    Ki(Konstans elő:Szöveg, e:TI, utó:Szöveg)
  Függvény
    Sorszám(Konstans x:TI):Egész
    Köv(Konstans x:TI):TI
    Elő(Konstans x:TI):TI
  Operátor
    AzonosE(Konstans x,y:TI):Logikai
      Másnéven x=y
    KisebbE(Konstans x,y:TI):Logikai
      Másnéven x<y
    LegyenEgyenlő(Változó x:TI, Konstans y:TI)
      Másnéven x:=y
Modul vége.
```

  - Egyszerűbben:

```
…
  Operátor
    =(Konstans x,y:TI):Logikai
    <(Konstans x,y:TI):Logikai
    :=(Változó x:TI, Konstans y:TI)
…
```

Sorozatok – Tömbáltalánosítás

- Elvárások az **indextípusoktól** (`TI`):
  - **Alapeset**: eleve definiált diszkrét típusból készítünk index-típust

```
ExportModul TI:
  Konstans [ezek eleve definiált típus-konstansok]
    Min,Max:TI
    Számosság:Egész
  Eljárás
    Ki(Konstans elő:Szöveg, e:TI, utó:Szöveg)
  Függvény
    Sorszám(Konstans x:TI):Egész
    Köv(Konstans x:TI):TI
    Elő(Konstans x:TI):TI
  Operátor
    =(Konstans x,y:TI):Logikai
    <(Konstans x,y:TI):Logikai
    :=(Változó x:TI, Konstans y:TI)
Modul vége.
```

  - **Használati példa**:

```
Konstans tól:Egész(?), ig:Egész(?)
Típus    TI=…tól..ig közötti egészek típusa [tól<ig]…
Változó  i,j:TI
j:=Max'TI
Ciklus i=Min'TI-tól Elő(Max'TI)-ig
  Ha j≥i akkor j:=Elő(j)
Ciklus vége
Ha j=(Számosság'TI+1) Div 2 akkor Ki:mit írjunk ki TI-ről?
```

  - **Feladat**: válaszoljon az algoritmusban feltett kérdésre („*mit írjunk ki TI-ről?*”)!

Sorozatok – Tömbáltalánosítás

- Elvárások az **indextípusok**tól (`TI`):
  - **Alapeset**: eleve definiált diszkrét típusból készítünk index-típust

```
ExportModul TI:
  Konstans
    Min,Max:TI
    Számosság:Egész
  Eljárás
    Ki(Konstans elő:Szöveg, e:TI, utó:Szöveg)
  Függvény
    Sorszám(Konstans x:TI):Egész
    Köv(Konstans x:TI):TI
    Elő(Konstans x:TI):TI
  Operátor
    =(Konstans x,y:TI):Logikai
    <(Konstans x,y:TI):Logikai
    :=(Változó x:TI, Konstans y:TI)
Modul vége.
```

  - **Megvalósítási ötlet**: alapesetben (pl. egészek intervalluma) magához, a típus értékhalmazához minden művelet rendelkezésre áll, vagy könnyen származtatható.
  - **Megjegyzés**: a **<** definiálja természetesen a ≥ operátort: **x≥y ↔ ¬(x<y)**.
  - **Feladat**: gondolja ezt meg az előbbi példa esetére!
  - **Kérdés**: `TI`-re hogyan építené egy `TE` elemű tömb-típus~~(konstrukció)~~ címfüggvényét?

```
Konstans tól:Egész(?), ig:Egész(?)
Típus    TI=…tól..ig közötti egészek típusa [tól<ig]…
Változó  i,j:TI
j:=Max'TI
Ciklus i=Min'TI-tól Elő(Max'TI)-ig
  Ha j≥i akkor j:=Elő(j)
Ciklus vége
Ha j=(Számosság'TI+1) Div 2 akkor Ki:mit írjunk ki TI-ről?
```

Sorozatok – Tömbáltalánosítás

- Elvárások az **indextípusok**tól (`TI`):
  - Amikor nem nyilvánvaló a **rákövetkezés** (pl. a **`Szöveg`** típus egy részhalmaza vagy pl. a prímek az indextípus alaphalmaza), akkor jól jön a `Sorszám` függvény, amely kielégíti:

    1° `Sorszám(Min)=0`
    2° `Sorszám(Köv(x))=Sorszám(x)+1`

```
ExportModul TI:
  Konstans
    …
  Eljárás
    …
  Függvény
    Sorszám(x:TI):Egész
    …
  Operátor
    …
Modul vége.
```

  - Használati példa$_1$:

```
Típus     TI=…Szöveg-sorozatként felsorolt, eltérő értékű,
             „rendezett” sorrendű szövegek…
Változó   i,j:TI
j:=Max’TI
[Max’TI:a legnagyobb (utolsó); Min’TI:a legkisebb (első)]
Ciklus i=Min’TI-tól Elő(Max’TI)-ig
  Ha j≥i akkor j:=Elő(j) [a TI-beli sorozat előző eleme]
Ciklus vége …
```

Probléma: a **`Szöveg`** típus rendezettsége ellenére, egy tetszőleges **részhalmazán** a **rákövetkezés** nem nyilvánvaló (‚A’→’B’ vagy ’AA’?).
Ugyanez a problémája a fent említett prímek indextípusként alkalmazásának, amely egy jól rendezett típusnak, az **`Egész`**-nek része.

Sorozatok – Tömbáltalánosítás

- Elvárások az **indextípusok**tól (`TI`):
  - Amikor nem nyilvánvaló a **rákövetkezés** (pl. a **`Szöveg`** típus egy részhalmaza vagy pl. a prímek az indextípus alaphalmaza), akkor jól jön a `Sorszám` függvény, amely kielégíti:

    1° `Sorszám(Min)=0`
    2° `Sorszám(Köv(x))=Sorszám(x)+1`

    Elvárás a megvalósítással szemben (l. a példa-program-részletet): `x<Köv(x), Elő(x)<x.`

```
ExportModul TI:
  Konstans
    …
  Eljárás
    …
  Függvény
    Sorszám(x:TI):Egész
    …
  Operátor
    …
Modul vége.
```

  - Használati példa$_2$:

Az előbbi működésének magyarázata.

```
Típus     TI=…Szöveg-sorozatként felsorolt, eltérő értékű,
             „rendezett” sorrendű szövegek…
Változó   i,j:TI
j:=Max’TI
[Max’TI:a legnagyobb (utolsó); Min’TI:a legkisebb (első)]
i:=Min’TI
Ciklus i≤Elő(Max’TI) [szöveg-hasonlítás alapján]
  Ha j≥i akkor j:=Elő(j) [a TI-beli sorozat előző eleme]
  i:=Köv(i) [a TI-beli sorozat következő eleme]
Ciklus vége …
```


```
Típus     TI=…Szöveg-sorozatként felsorolt, eltérő értékű,
             „rendezett” sorrendű szövegek…
Változó   i,j:TI
j:=Max’TI
[Max’TI:a legnagyobb (utolsó); Min’TI:a legkisebb (első)]
Ciklus i=Min’TI-tól Elő(Max’TI)-ig
  Ha j≥i akkor j:=Elő(j) [a TI-beli sorozat előző eleme]
Ciklus vége …
```


A fenti elvárás a rendezett sorrend esetén teljesül!

Sorozatok – Tömbáltalánosítás

➢ Elvárások az **indextípusok**tól (`TI`):

- Amikor nem nyilvánvaló a **rákövetkezés** (pl. a **`Szöveg`** típus egy részhalmaza vagy pl. a prímek az indextípus alaphalmaza), akkor jól jön a `Sorszám` függvény, amely kielégíti:

  1° `Sorszám(Min)=0`
  2° `Sorszám(Köv(x))=Sorszám(x)+1`

  Elvárás a megvalósítással szemben (l. a példa-program-részletet): `x<Köv(x), Elő(x)<x.`

```
ExportModul TI:
  Konstans
    …
  Eljárás
    …
  Függvény
    Sorszám(x:TI):Egész
    …
  Operátor
    …
Modul vége.
```

- **Megvalósítási ötlet**: ilyen esetben magát a típus értékhalmazát is tárolni kell egy konstans tömbben (felsoroljuk a szöveg-konstansokat, prímeket), minden műveletet erre a tárolt sorozatra kell építeni (kihasználandó a szövegok/egészek sorrendje≡rendezettsége). A `Sorszám`≡…konstans tömbbeli index…
- **Feladat**: gondolja meg: az előbbi példa esetén, mely művelet hogyan valósítható meg?
- **Feladat**: a példában szereplő `TI`-re hogyan építené egy `TE` elemű tömb-típuskonstrukció címfüggvényét? (Legyen a `Méret'T` a `T` típus ábrázolásához szükséges bájtok száma.)

```
Típus     TI=…Szöveg-sorozatként felsorolt, eltérő értékű,
            „rendezett” sorrendű szövegek…
Változó   i,j:TI
j:=Max'TI
[Max'TI:a legnagyobb (utolsó); Min'TI:a legkisebb (első)]
i:=Min'TI
Ciklus i≤Elő(Max'TI) [szöveg-hasonlítás alapján]
  Ha j≥i akkor j:=Elő(j) [a TI-beli sorozat előző eleme]
  i:=Köv(i) [a TI-beli sorozat következő eleme]
Ciklus vége …
```

Sorozatok – Tömbáltalánosítás

➢ Elvárások az **indextípusok**tól (`TI`):

- ❖ Amikor nem nyilvánvaló a **rákövetkezés** (pl. a **`Szöveg`** típus egy részhalmaza vagy pl. a prímek az indextípus alaphalmaza), akkor jól jön a `Sorszám` függvény, amely kielégíti:

  1° `Sorszám(Min)=0`
  2° `Sorszám(Köv(x))=Sorszám(x)+1`

  Elvárás a megvalósítással szemben (l. a példa-program-részletet): `x<Köv(x), Elő(x)<x.`

```
ExportModul TI:
  Konstans
    …
  Eljárás
    …
  Függvény
    Sorszám(x:TI):Egész
    …
  Operátor
    …
Modul vége.
```

- ❖ Használati példa$_3$:

Az előbbi magyarázata, **Sorszám** fv-nyel.

```
Típus     TI=…Szöveg-sorozatként felsorolt, tetszőleges sorrendű,
             eltérő szövegek…
Változó   i,j:TI
j:=Max'TI
[Max'TI:az utolsó (Sorszám(Max'TI)=Számosság'TI-1);
 Min'TI:az első (Sorszám(Min'TI)=0]
i:=Min'TI
Ciklus i≤Elő(Max'TI) [≈ Sorszám(i)≤Sorszám(Elő(Max'TI))]
  Ha j≥i akkor j:=Elő(j) [≈ j:=Sorszám(j)-1. TI-beli szöveg]
  i:=Köv(i) [≈ i:=Sorszám(i)+1. TI-beli szöveg]
Ciklus vége …
```

Sorozatok – Tömbáltalánosítás

- **Példa**: 1 és 10 közötti egészek mint **indextípus** megvalósítási modulja:

```
Modul TI_1_10:
 Reprezent
  Típus
    TI_1_1
  Konstans
    Min:TI
    Max:TI
    Számos
…
```

```
…
 Impleme
  Eljárá
    Ki:e
  Eljárá
  Függvé
    Sors
  Függvé
  Függvé
    Köv:
  Függvé
  Függvé
    Elő:=x-1 [az alulcsordulással most nem foglalkozunk]
  Függvény vége.
  Operátor AzonosE(Konstans x,y:TI_1_10):Logikai
    Másnéven x=y
    AzonosE:=x=Egészy [visszavezetés az „ős” típus azonosság műveletére]
  Operátor vége.
…
```

```
…
  Operátor KisebbE(Konstans x,y:TI_1_10):Logikai
    Másnéven x<y
    KisebbE:=x<Egészy [visszavezetés az „ős” típus kisebb műveletére]
  Operátor vége.
  Operátor :=(Változó x:TI_1_10, Konstans y:TI_1_10):
    x:=Egészy [visszavezetés az „ős” típus azonosság műveletére]
  Operátor vége.
 Inicializálás
  :=Min [deklarációkori értéke legyen: Min]
Modul vége.
```

```
ExportModul TI_1_10:
  Konstans
    Min,Max:TI_1_10
    Számosság:Egész
  Eljárás
    Ki(Konstans elő:Szöveg
  Függvény
    Sorszám(Konstans x:TI_
    Köv(Konstans x:TI_1_10
    Elő(Konstans x:TI_1_10
  Operátor
    =(Konstans x,y:TI_1_10
    <(Konstans x,y:TI_1_10
    :=(Változó x:TI_1_10,
Modul vége.
```

Sorozatok – Tömbáltalánosítás

- **Példa**: a kurzusra járók neveiből alkotott szövegekből álló érték-halmazú **indextípus** megvalósítási modulja:

```
Modul TI_Név:
 Reprezentáció
  Típus
    TI_Né
  Konstan
    KLéts
    KTago
  Konstan
    Min:T
    Max:T
    Számo
…
```

```
…
 Implementáció
  Függvér
    … x )
   Sorszá
  Függvér
  Eljárás
    Ki:e
  Eljárás
  Függvér
    Köv:=
  Függvér
…
```

```
…
  Függvény Elő(Konstans x:TI_Név):TI_Név
    Elő:=KTagok(Sorszám(x)-1) [alulcsordulás lehet!!!]
  Függvény vége.
  Operátor AzonosE(Konstans x,y:TI_Név):Logikai
    Másnéven x=y
    AzonosE:=x=Szöveg y [visszavezetés az „ős” típus azonosság műveletére]
  Operátor vége.
  Operátor KisebbE(Konstans x,y:TI_Név):Logikai
    Másnéven x<y
    KisebbE:=Sorszám(x)<Egész Sorszám(y) [visszavezetés]
  Operátor vége.
  Operátor :=(Változó x:TI_Név, Konstans y:TI_Név):
    x:=Szöveg y [visszavezetés az „ős” típus azonosság műveletére]
  Operátor vége.
 Inicializálás
  :=TI_Név(Min) [deklarációkori értéke legyen: Min]
Modul vége.
```

Sorozatok – Tömbáltalánosítás

- Elvárások az elemtípusoktól (TE):
  - Exportmodul:

```
ExportModul TE:
  Függvény
    Be(Konstans kérdés:Szöveg, Változó e:TE):Logikai
  Eljárás
    Ki(Konstans elő:Szöveg, e:TE, utó:Szöveg)
  Operátor
    =(Konstans x,y:TE):Logikai
    :=(Változó x:TE, Konstans y:TE)
    <(Konstans x,y:TE):Logikai
Modul vége.
```

- Feladatok:
  - Készítse el az 1-10 egészek elemtípusának megvalósító `TE_1_10` modulját!
  - Készítse el a kurzusra járók neveiből alkotott szöveg-típusnak a `TE_Név` megvalósító modulját, amit elemtípusnak szánunk!

Az „O” utal arra, hogy osztályként ábrázoljuk az adott típust.

Sorozatok – Tömbáltalánosítás

Házi feladat-1:

- Készítse el a gyakorlaton utóbb szóba került modulok unitjait, amelyben a típusokat osztályokként reprezentálja
  - `OI_1_10_Unit`,
  - `OE_1_10_Unit`,
  - `OI_Nev_Unit`,
  - `OE_Nev_Unit`.

  Kiindulhat az alábbi sablonokból: `OI_Sablon_Unit.pas`, `OE_Sablon_Unit.pas`.

  Fontos kiegészítés:

  Algoritmikusan mindenféle konverziós függvény létrejötte automatikus egy új típus definiálásakor (ezért nem jelent meg a korábbi exportmodulokban). Például valamely érték szöveggé, illetve belső ábrázolásából az adott típusú értékké konvertálás:

  - `Szöveg:Típus→Szöveg,` ill. az „inverze”: `Típus:Szöveg→Típus`
  - sokszor értelmes ez is: `Típus:Egész→Típus`

  Praktikus okok miatt (kihasználva a **polimorfia** lehetőségét) a megírandó indextípusokhoz tartozó *unitokban* az alábbi néven szerepeljen az alábbi konverziós függvény:

  - ```
    Function _2Str:String; {String-gé konvertálja az adott típusú értéket}
    Function Str2_:T; {T típusú értékké konvertálja a String-es megfelelőt}
    ```
  - ```
    Function Int2_:T; {T típusúvá konvertálja az belső ábrázolású értéket}
    ```

Sorozatok – Tömbáltalánosítás

Házi feladat-1 (folytatás):

- Kódolásnál további fontos bővítés a hibázás lehetőségének figyelembe vétele. Például egy indextípus esetén a `Köv/Elő` műveletek túl-/alulcsordulása, vagy az elemtípus esetén a beolvasás közben az értékhalmaz elvétése.
  Ennek szokásos kódolási trükkje egy `hiba:Boolean` mező bevezetése és kezelése:

```
Type
  Generic OX<…>=Class(TObject)
   {Private}
    … lényegi mezők deklarálása …
    hiba:Boolean;
   Public
    Constructor Create;
    … a lényegi műveletek fejsorai …
    Function HibasE:Boolean;
  End;

Implemetation
  Constructor OX.Create;
  Begin
    … alap állapotba hozás …
    hiba:=False; {még hibátlan}
  End;
  … a további lényeg műveletek megvalósítása …
  Function OX.HibasE:Boolean;
  Begin
    HibásE:=hiba; hiba:=False; {újból hibátlan}
  End;
```

Sorozatok – Tömbáltalánosítás

Házi feladat-1 (folytatás):

- A `Számosság'T / Min'T / Max'T` típus-konstansokat **osztálymetódusként** kell definiálni:

```
Generic OX<…>=Class(TObject)
    … a mezők deklarálása …
  Public
    Class Szamossag:Integer;
    Class Function Min:OX;
    Class Function Max:OX;
    … a lényegi műveletek fejsorai …
  End;

Implemetation
  Class Function OX.Min:OX;
  Begin
    Min:=OX.Create;
    … a mezők Min-hez illő beállítása …
  End;
  Class Function OX.Max:OX;
    …
  Class Function OX.Szamossag:Integer;
   …
  … a lényegi műveletek magvalósítása …
```

- Példaként nézze meg és értse meg az `OI 1 10 Unit.pas`-t és az `OE 1 10 Unit.pas`-t!

Sorozatok – Tömbáltalánosítás

- Házi feladat-2:
  - Építse össze az előbbi 4 unitot egyetlen próba programba. A program célja mindössze annyi, hogy kipróbálja az egyes típus-műveleteket. (Ciklus-szervezés mindkét indextípus felhasználásával, ellenőrzötten beolvasni mindkét elemtípusú változóba és kiírni annak tartalmát. Valami ilyesmire gondoltam: Tomb_I_1_10_E_1_10_Proba1.exe. Ez csak az 1-10 közöttiek index, ill. elemtípusát tartalmazza.)