3. előadás

Típusrendszerek.
Az Ada típusai és típuskonstrukciós eszközei, I.

Típus

- Egy bitsorozatot mire szeretnénk használni
- Lehetséges értékek (típusértékhalmaz)
- Használható műveletek
- Típusspecifikáció/típus a módszertanban

Integer típus

- az egész számok halmaza:
  ..., -3, -2, -1, 0, 1, 2, 3, ...
- a szokásos műveletek:
  +, -, *. /, ** stb.

Mire használjuk a típusokat?

- Változók, objektumok, alprogramok deklarálásához, használati módjának megadásához (típusspecifikáció)
- Objektumok létrehozásához (típus)
  - Objektum-elvű nyelvekben: osztály, konstruktor
- Egyes nyelvekben szétválik

Mire jó a típusrendszer?

- Biztonság növelése
- Optimalizálás megkönnyítése
- Dokumentálás
- Absztrakció támogatása

Erősen típusos nyelv

- Minden (helyes) kifejezés típushelyes: nem lesz futás közbeni típushiba
- Ada, C++, Java: erre törekszenek
  - gyengén típusos nyelvek: például a szkriptnyelvek
- Nagyobb programok fejlesztésénél nagy segítség
- Védelem bitsorozatok (véletlenszerű) téves használata ellen
  - pl. int és pointer

Típushelyesség biztosítása

- Típusellenőrző rendszer
  - type checking
  - Ada, C++, Java, …
  - a programozó megadja az entitások típusát
- Típuskikövetkeztető rendszer
  - type inferencing
  - modern funkcionális nyelvek (ML, Haskell, …)
  - a fordító kikövetkezteti a típusokat

Típusmeghatározás

- Statikus típusellenőrzés és kikövetkeztetés
  - fordítási időben
  - hasznos: hibák korai felfedezése, hatékonyság növelése
  - Ada, C++, Java
- Dinamikus típusellenőrzés
  - futási időben
  - haszna: nagyobb rugalmasság, nagyobb kifejezőerő
  - Java, Smalltalk, LISP

Biztonságosság vs. rugalmasság

- Rugalmas, de nem statikus: Smalltalk
  - csak dinamikus típusellenőrzés
- Rugalmatlan, de meglehetősen biztonságos statikus típusrendszer: C++, Pascal
  - típuskényszerítés, variáns rekord
- Rugalmas, de nem biztonságos statikus t.r.: Eiffel
- Rugalmas és biztonságos statikus t.r.: Ada, Java
  - helyenként dinamikus típusellenőrzéssel kombinálva

Típusekvivalencia

- Mikor tekintünk két típust ekvivalensnek?
  - szerkezeti típusekvivalencia: ha ugyanolyanok
  - név szerinti típusekvivalencia: ha kijelentjük, hogy ekvivalensek

Típusekvivalencia a C++ nyelvben

- Osztályok: név szerinti ekvivalencia

class A { int x; }; class B { int x; };

  - nem ekvivalensek!

- typedef

typedef int LENGTH;

  - csak nevet deklarál, nem definiál új típust
  - szinonníma bevezetése

Típusekvivalencia az Ada nyelvben

- A C++ nyelvhez hasonlóan
  - Név szerinti ekvivalencia
  - Van lehetőség szinonníma definiálására
- Szigorúbb szabályok a használatra

Beépített típusok az Adában

- Integer
- (Positive)
- (Natural)
- Boolean
- Float
- Character
- String
- Wide_Character
- Wide_String
- Duration

Újabb típusokat lehet definiálni

```
type T is ...
```

- A típusértékek (és műveletek) megadásával, felsorolási típust:

  type Irány is (Fel, Jobbra, Le, Balra);
- Származtatással:

  type Terület is new Float;
- Építéssel (összetett típust, pl. tömb típust)

  type Tömb is array (1..10) of Integer;

Absztrakt adattípus

- Elvonatkoztatunk a reprezentációnak és a műveletek implementációjának részleteitől
- Átlátszatlan, „privát” típus

type Verem is private;

- Típusspecifikáció

Új típus származtatással

```
type Sebesség is new Float;
type Idő is new Float;
type Út is new Float;
T: Idő := 4.0;
V: Sebesség := 3.0;

T := V;     -- helytelen! fordítási hiba
```

- T és V különböző típusúak (név szerinti ekv.)
  - sok programozói hiba kiszűrhető így

Származtatás

- A származtatás során új típus jön létre
- Az eredeti típus típusértékhalmaza lemásolódik
- Az eredeti típus műveletei lemásolódnak
- Lexikálisan a két típus típusértékei és műveletei megegyeznek, de egymással nem lesznek kompatíbilisek (átlapolás)

V: Sebesség := 3.0 + 1.0;

F: Float := 3.0 + 1.0;

- Öröklődés: hasonlóságok és különbségek

Explicit típuskonverzió

- Oda-vissza konvertálhatunk típusnévvel
- Nincs automatikus (implicit) konverzió
  - A szokásosnál szigorúbbak a játékszabályok

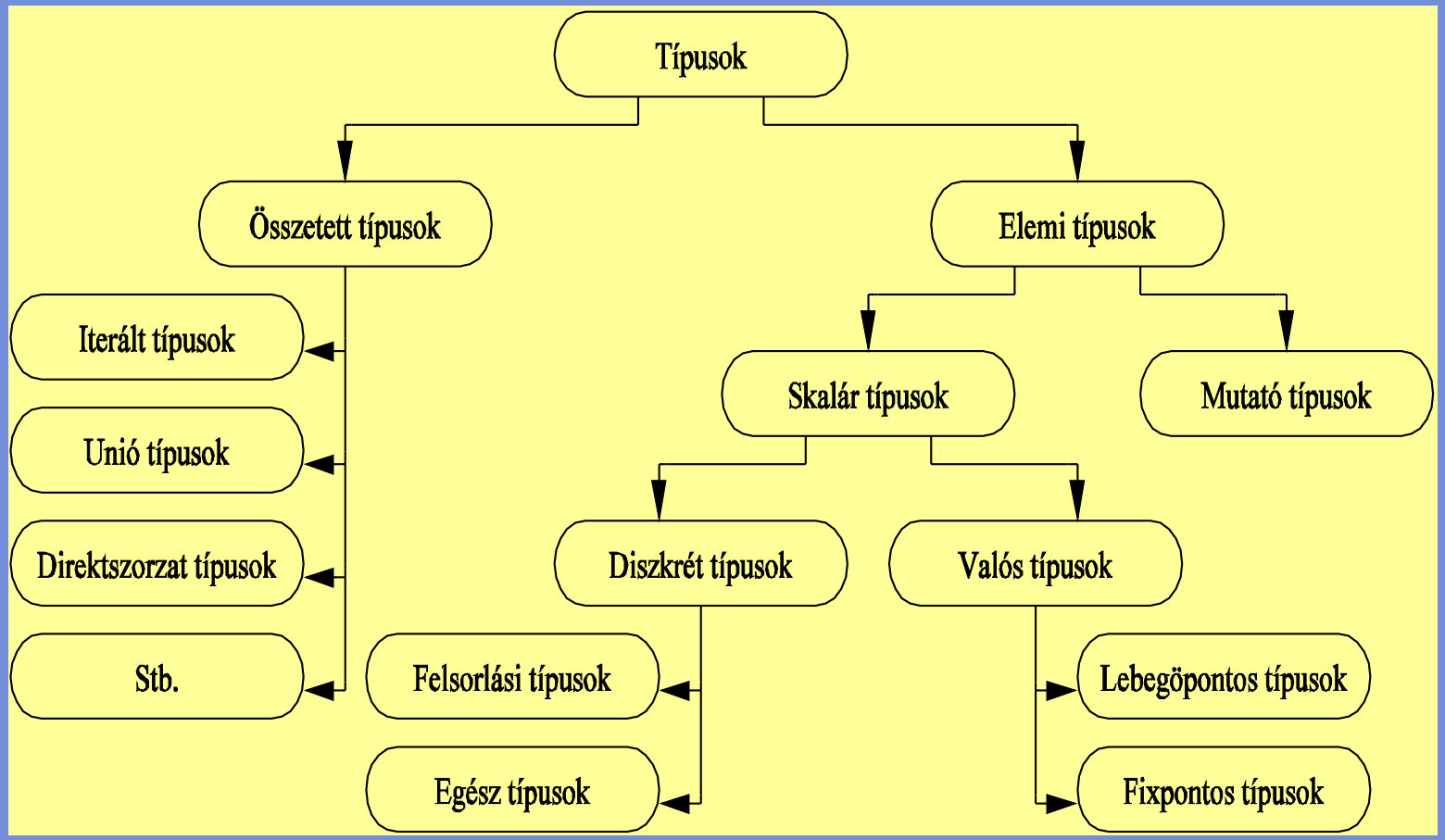
```
type Int is new Integer;
I: Integer := 3;
J: Int := Int(I);
K: Integer := Integer(J+1) + I + 1;
```

Típusosztályok

- ***Elemi típusok***
  - skalár
    - diszkrét (felsorolási és egész)
    - valós (fix- és lebegőpontos)
  - mutató
- ***Összetett típusok***

- tömb - rekord - *jelölt* - *taszk t.* - *védett t.*

Típusosztályok rajzban



Elemi típusok

- skalár
  - diszkrét
    - felsorolási (Character, Boolean)
    - egész
      - előjeles (Integer)
      - moduló típusok
  - valós
    - lebegőpontos (Float)
    - fixpontos
      - közönséges fixpontos
      - decimális fixpontos
- mutató

Felsorolási típusok

- Elemi, skalár, diszkrét
- Felsoroljuk a típusértékeket
  - mint a C++ enum
  - de nem kompatíbilisek az egész számok típusával

```
type Irány is (Fel, Jobbra, Le, Balra);
type Boolean is (False, True);
type Character is (…);

I: Irány := Fel;
```

Egész típusok (1) elemi, skalár, diszkrét

- Előjeles (signed) egész típusok, pl. *Integer*
  - Definiálhatunk újabbakat: *származtatással*

    type Másik is new Integer;
  - Ez a típus nem kompatíbilis az Integer-rel!

```
I: Integer := 5;          M2: Másik := I;
M1: Másik := 5;           J: Integer := M1;
M3: Másik := Másik(I);
K: Integer := Integer(M3);
```

- Moduló (unsigned) típusok

Egész típusok (2) elemi, skalár, diszkrét

- Előjeles egész típusok, pl. *Integer*
- Moduló típusok

type Mod10 is mod 10;

type Bájt is mod 256;

  - A típusértékek a 0..9, illetve a 0..255
  - A számokon értelmezett szokásos műveletek (például a “+”) a maradékosztályok szerint, azaz modulo számolnak.

Valós típusok (1) elemi, skalár

- Lebegőpontos számtípusok
  - egy rögzített hosszúságú *mantissza* és egy előjeles egész *exponens*

    type Real is digits 8;

    Pi: Real := 3.1415926
  - digits: a mantissza ábrázolásához szükséges decimális jegyek száma
  - predefinit típus: Float - implementációfüggő
  - a többi ebből, származtatással

    type Real is new Float digits 8;
- Fixpontos számtípusok

Valós típusok (2) elemi, skalár

- Lebegőpontos számtípusok
- Fixpontos számtípusok
  - rögzített számú számjegy és egy képzeletbeli tizedespont

    type Fix is delta 0.01 range -1.0 .. 1.0;
  - tizedes fixpontos típus:

    type Fix is delta 0.01 digits 15;

  az értékek a következő intervallumban:

  –(10**digits–1)*delta .. +(10**digits–1)*delta.

Összetett típusok: tömbök

- Típuskonstrukció
  - Elemek típusa + indexek típusa

  type Tömb is array (1..10) of Integer;
- Többdimenziós tömbök is vannak
- Kicsit később…

Altípusok

- Egy típusértékhalmazra tett megszorítás
- Nem vezetünk be új típust, csak egy altípust a meglévő típushoz

subtype Napok is Integer range 1..31;

Elseje: Napok := 1;

Egy: Integer := 1;

- Az Elseje és az Egy változó típusa megegyezik

Az altípusok tulajdonságai

- A típusértékhalmaz (általában) kisebb lesz
- A műveletek ugyanúgy használhatók lesznek
- Az altípus használható , mint az eredeti (változódeklaráció, formális paraméter dekl., újabb típusok stb.)
- Az altípusba tartozó entitás és a bázistípusba tartozó entitás kompatíbilisek egymással

Altípusok és kompatibilitás

Elseje := Egy;

Egy := Elseje;

- A fordító elfogadja az értékadásokat
  - azonos típusú értékek
- Azt is ellenőrizni kell, hogy az értékadás nem sérti meg a típusinvariánst

Elseje := 42;

Dinamikus típusellenőrzés

- Azt is ellenőrizni kell, hogy az értékadás nem sérti meg a típusinvariánst

Elseje := 42;

- A fordító ezt elfogadja, esetleg egy figyelmeztetést (warning) küld:

  warning: "Constraint_Error" will be raised at run time

- A futtató rendszer ellenőrzi, hogy a baloldal altípusának (a korlátozásnak) megfelel-e a jobboldali típusérték

Ada: típusok és altípusok

- A típus feladata:
  a ***fordítónak*** információ az entitásokról
  (melyik entitást mire lehet használni)
- Az altípus feladata:
  kiegészítő információk a ***futtató rendszer*** számára

Nevezetes (beépített) altípusok

```
subtype Natural is
                Integer range 0..Integer'Last;
subtype Positive is
                Integer range 1..Integer'Last;
```

lehetne…:

```
subtype Positive is
                Natural range 1..Natural'Last;
```

Ada RM terminológia

- Az Adában minden „típusdefiníció” igazából egy altípust definiál
  - az ún. „első altípust”
  - az Ada programokban egységesen mindig altípusok szerepelnek
- Az első altípusból újabb altípusok képezhetők „altípus deklarációkkal”

Altípusképzés altípus-deklaráció nélkül

```
subtype Napok is Integer range 1..31;
Nap: Napok := 7;
Február_Nap: Napok range 1..29 := 3;

Szám: Integer range 1..31 := 7;
```

Típusszinonímák

subtype Elem is Integer;

- nem vezet be megszorítást, csak egy új nevet
- a kód olvashatóságát növelheti
- C++ typedef

Dinamikus megszorítás

```
declare
        N: Integer;
begin
        Get(N);
        declare
                subtype Számok is Integer range 1..N;
                Szám: Számok := 3;
        begin
                …
        end;
        ...
end;
```

Példák altípusokra

```
subtype Napok is Integer range 1..31;
subtype Valószínűség is Float range -1.0 .. 1.0;
subtype Nagybetűk is Character range ‘A’ .. ‘Z’;

type Hét is (Hétfő, Kedd, Szerda, Csütörtök, Péntek,
                    Szombat, Vasárnap);

subtype Munkahét is Hét range Hétfő .. Péntek;


Ma: Hét := Szombat;

if  Ma in Munkahét  then  Elmegyek_Dolgozni;  end if;
```

Származtatás + altípusképzés

```
type Napok is new Integer range 1..31;
type Napok is range 1..31;

type Hónap_Napok is new Integer;
subtype Napok is Hónap_Napok range 1..31;
```

Típusműveletek

- Predefinit (előre definiált) operátorok
  - például egész számoknál: + - * / stb.
- A nyelv által definiált egyéb műveletek
  - értékadás, (nem)egyenlőség-vizsgálat
    := = /=
- A típusosztályhoz tartozó attribútumok
- Általunk írt bizonyos alprogramok
- Az ún. primitív műveletek öröklődnek a származtatás során

Attribútumok

- Speciális típusműveletek
- A típusosztályokhoz rendelt műveletek
  - Például skalár típusosztályhoz…
  - A típusosztály minden típusára használható
- Speciális szintaxis:

Integer 'Image (ez pl. skalár attribútum)

Példa attribútum használatára

```
F: Positive := Faktoriális(10);

…
Text_IO.Put_Line( Positive'Image(F) );
```

Skalár típusosztály

- diszkrét (felsorolási, egészek),
  valós (fixpontos és lebegőpontos)
- a skalár típusok rendezettek
  - relációs operátorok
    < <= > >= in not in
- részintervallum kijelölés

  range Alsó_Határ .. Felső_Határ
  subtype Számjegyek is Integer range 0..9;
- skalár attribútumok

Néhány skalár attribútum

S'First	az S legkisebb típusértéke pl. Natural'First = 0
S'Last	…
S'Range	ugyanaz, mint S'First .. S'Last
S'Image	*function S'Image(A: S'Base) return String* szöveges reprezentációra alakít
S'Value	*function S'Value(A: String) return S'Base* stringet elemez/értelmez/S-sé alakít

Tömbök

- Elemek egy sorozata
- Az elemekhez indexeléssel férünk hozzá: ()
- A nyelvek általában speciális szintaxist biztosítanak a tömbökhöz
- Hatékony az elemek elérése, de a tömb méretének módosítása nem

```
type T is array ( 1..10 ) of Integer;
X: T := (1,3,5,7,9,2,4,6,8,0);
I: Integer := X(5);
```

Tömb típusok az Adában

- Összetett (iterált) típusok megvalósításának egy eszköze
- Az indextípus tetszőleges diszkrét típus (felsorolási vagy egész)
- Az elemtípus tetszőleges (teljesen meghatározott) típus
- Szemben pl. a C++ nyelvvel

Típusekvivalencia tömb típusokra

- A tömb az Adában egy „valódi” típuskonstrukciós eszköz
  - szemben pl. a C++ tömbjeivel

  int t[10];
- Minden tömbtípus-definíció új típust hoz létre (név szerinti ekvivalencia)

type T1 is array ('a' .. 'z') of Natural;

type T2 is array ('a' .. 'z') of Natural;

  - T1 és T2 nem ekvivalensek!

Példák

```
type T1 is array ('a' .. 'z') of Natural;
type T2 is array (Character range 'a' .. 'z') of Natural;
type T3 is array (Character) of Natural;
type T4 is array (Character'First .. Character'Last) of Natural;
type T5 is array (1..10) of T2;
type T6 is array (Integer) of T3;    -- Storage_Error veszélye!
                                     --    X: T6;
```

Névtelen tömb típus

type Napok is (Hétfô, Kedd, Szerda, Csütörtök, Péntek, Szombat, Vasárnap);

Házimunka: array (Napok) of Natural;

- Az egyes komponensek:
  Házimunka(Hétfô) a tömb első eleme,
  Házimunka(Kedd) a második stb.

- Ebből a névtelen típusból csak egy objektum jön létre

Névtelen altípus használata

```
type Napok is (Hétfô, Kedd, Szerda, Csütörtök,
                                Péntek, Szombat, Vasárnap);
subtype Munkaórák is Natural range 0..24;
Munka: array (Napok range Hétfô .. Péntek)
                                of Munkaórák;
```

Megszorítások ellenőrzése

```
type Napok is (Hétfô, Kedd, Szerda, Csütörtök, Péntek,
                                        Szombat, Vasárnap);
subtype Munkaórák is Natural range 0..24;
Munka: array (Napok range Hétfô .. Péntek) of Munkaórák;
…
Munka(Szombat):=6;
Munka(Hétfô):= 44;
```

- Fordítás: warning, futtatás: Constraint_Error

Többdimenziós tömb

type T is array (Boolean, Boolean) of Boolean;
És: T := ((False, False), (False,True));
… És(False,True) …

- Nem olyan, mint a tömbök tömbje

type T1 is array (Boolean) of Boolean;
type T2 is array (Boolean) of T1;
Vagy: T2 := ((False,True), (True,True));
… Vagy(False)(True) …

- Sorfolytonos ábrázolás

Tömbműveletek

- Indexelés: X(3)
- Egydimenziós esetben szelet kiválasztása: X(1..3)
  - X(1..1) és X(1) más típusú!
- := = /=
- diszkrét értékű vektorokra a < , <= , > , >= (lexikografikusan értelmezett) relációk
- Az azonos hosszúságú logikai értékeket tartalmazó vektorokra az **and**, **or**, **xor**, és a **not** műveletek
- A tömb típusosztály attribútumai

Tömbök attribútumai

```
type T is array ( 1..10 ) of Integer;
X: T := (0,2,4,6,8,1,3,5,7,9);
```

X ' First	az első index, azaz 1
X ' Last	az utolsó index, azaz 10
X ' Range	X ' First .. X ' Last
X ' Length	az X hossza, azaz 10

- Nem a típus, hanem az objektum szerepel az attribútum neve előtt (v.ö. Integer ' First)

Iterálás tömbön

```
for I in X ' Range loop
Put_Line(   Integer ' Image( X(I) )   );
end loop;
```

Többdimenziós tömbök attribútumai

type T is array (Boolean, Boolean) of Boolean;

És: T := ((False, False), (False,True));

És ' First(1)	az első index az első dimenzióban, azaz False
És ' Last(2)	az utolsó index a második dimenzióban, azaz True

…

Megszorítás nélküli index

type Vektor is array (Integer range <>) of Integer;

type Mátrix is array (Integer range <>, Integer range <>)
of Float;

- Ekkor a változó deklarálásakor kell eldönteni a konkrét indexhatárokat:

V1: Vektor (1 .. 3);

V2: Vektor (2 .. 4);

V3: Vektor (1 .. 8);

A: Mátrix(1 .. 2, 1 .. 5);

M: Mátrix; -- hibás változódefiníció (fordítási hiba)

Rugalmas használat

```
type Elem is new Integer;
type Index is new Integer;
type Vektor is array (Index range <>) of Elem;

function Max( A: Vektor ) return Elem is
     M:Index := A'First;
begin
     for I in A'Range loop
          if   A(M) < A(I)   then   M:= I;   end if;
    end loop;
    return A(M);
end Max;
```

Megszorítás nélküli indexű tömb típus

- Alprogram formális paraméterének vagy visszatérési értékének típusa lehet ilyen
- Az aktuális paraméter, illetve a kiszámolt érték határozza meg az aktuális indexhatárokat
- Az attribútumokat használjuk az alprogramban
- Objektum létrehozásához meg kell adni az indexhatárokat (méretet), altípusképzéssel
- Különböző indexelésű tömbök egy típusba tartozhatnak

Altípus létrehozása

```
type Beosztás is array (Napok range <>) of Boolean;
subtype Munkanap is Beosztás(Hétfô .. Péntek);

Kati_Beosztása : Beosztás(Kedd .. Szombat);
Jancsi_Beosztása : Munkanap;

Peti_Beosztása : Beosztás := (True,True,False);
```

- Ilyenkor Napok'First az első index
- Megtévesztő lehet, ha Integer az indextípus: -2147483648

Megszorítás nélküli indexű tömbök jellemzői

- Üres tömb: üres indexintervallum

type Vektor is array (Index range <>) of Elem;

V: Vector(1..0);

- Értékadás szeletre: a szelet is balérték

W(1..3) := (3,5,1);

- Konkatenáció művelet: & operátor
  - tömböt tömbbel, elemet tömbbel, tömböt elemmel, elemet elemmel

Példák konkatenációra

```
type Vektor is array (Integer range <>) of Float;


K1: Vektor(0..2);
K2: Vektor(1..3);              --  ok:   K2 := K1;
K3: Vektor := K1&K2;           -- K3'First = 0
K4: Vektor :=K1&1.0;           --  ok:   K4(0..2) := K2;
K5: Vektor :=2.0&3.0;          -- K5'Length = 2
K6: Vektor :=1.0&K1;
```

Nem teljesen meghatározott típus

- indefinite type
- Például a meghatározatlan indexű tömb típus
- Nem lehet közvetlenül változót definiálni vele
  - nem ismerjük a méretét
  - de például helyes ez: X: T := (1,5,3);
- Lehet viszont pl. formális paraméter típusa
- Nem lehet például tömb elemtípusa
- Lesz még más ilyen típus is…

A String típus

- Beépített típus

type String is array (Positive range <>) of Character;

- Speciális szintaxis

S1: constant String := "GIN";

S2: constant String := ('G','I','N');

- Rugalmatlanabb, mint más nyelvekben
  - A tömbökre vonatkozó szabályok miatt
  - Mutatókkal, dinamikus memóriakezeléssel segíthetünk
  - Ada95: Bounded_String, Unbounded_String

Tipikus hibák String használatánál (1)

- Különböző méretű String-ek nem adhatók egymásnak értékül: Constraint_Error

```
S: String(1..256);
S := Integer'Image( Faktoriális(4) );
```

- Egy másik tanulság: lehetőleg ne égessünk bele a programba konstansokat.
  - Mi van, ha az Integer típus egy adott implementációban szélesebb?

Tipikus hibák String használatánál (2)

- Egy Get_Line-os beolvasás esetleg csak részben tölti fel a sztringet

```
S: String( 1..Sor_Hossz );
H: Natural;
Get_Line(S,H);
… Integer'Value( S ) …
… Integer'Value( S(1..H) ) …
```

Tömbaggregátum

type T is array (1..6) of Float;

- Pozícionális megadás

  X: T := (1.0, 3.0, 1.0, 2.0, 2.0, 2.0);
- Névvel jelölt megadás

  X: T := (2 => 3.0, 1|3 => 1.0, 4..6 => 2.0);
- Maradék

  X: T := (2 => 3.0, 1|3 =>1.0, others => 2.0);
- Keverés

  X: T := (1.0, 3.0, 1.0, others => 2.0);

Korlátozás nélküli index esetén

type T is array (Integer range <>) of Float;

- Pozícionális megadás

  X: T := (1.0, 3.0, 1.0, 2.0, 2.0, 2.0);
- Névvel jelölt megadás

  X: T := (2 => 3.0, 1|3 => 1.0, 4..6 => 2.0);
- Helytelenek:

  X: T := (2 => 3.0, 1|3 =>1.0, others => 2.0);

  X: T := (1.0, 3.0, 1.0, others => 2.0);
- Helyesek:

  X: T(1..10) := (1.0, 3.0, 1.0, others => 2.0);

  X: T(1..10) := (2 => 3.0, 1|3 =>1.0, others => 2.0);

Többdimenziós esetben

M: Mátrix(1..2, 1..3):=

(1 = > (1.1,1.2,1.3), 2 = > (2.1,2.2,2.3));

D: Mátrix := (1 .. 5 = > (1 .. 8 = > 0.0));

A skalár típusosztály attribútumai

S'First, S'Last, S'Range, S'Image, S'Value

S'Base	az S bázistípusa (a megszorítás nélküli altípus)
S'Min	function S'Min(A,B: S'Base) return S'Base
S'Max	a két érték maximuma
S'Succ	function S'Succ(A: S'Base) return S'Base rákövetkező elem (Constraint_Error lehet)
S'Pred	…
S'Width	maximuma az S'Image által visszaadott stringek hosszának

S'Wide_Image, S'Wide_Width, S'Wide_Value

A diszkrét típusosztály

- Felsorolási és egész (előjeles és moduló) típusok
- Ezen típusoknál a típusértékek a típuson belül pozíciószámmal azonosíthatók

S'Pos(A) function S'Pos(A: S'Base) return Integer
egy egész szám, a pozíció

S'Val(n) az adott pozíciónak megfelelő típusérték

- A pozíciószám követi a skalár rendezést

A felsorolási típusok osztálya

- A skalár rendezést itt a típusértékek felsorolási sorrendje adja
- A diszkrét pozíciószám szintén a felsorolást követi
  - nullától egyesével

Az egész típusok osztálya

- Predefinit operátorok

  +A -A A+B A-B A*B A/B

  A rem B A mod B abs A A**B

  - Az egész osztás csonkít (nulla felé...)
  - Hatványozásnál B nemnegatív
- Moduló típusoknál:

  S'Modulus a típus modulusa

A valós típusok osztálya

+X -X X+Y X-Y X*Y X/Y X**Y

- ***hatványozás:* Y *egész***

- Attribútumok
  - Lebegőpontos: *'Digits*
  - Fixpontos: *'Small, 'Delta, 'Fore, 'Aft, 'Scale, 'Round*
    - Decimális fixpontos: *'Digits*