4. előadás

Az Ada típusai és típuskonstrukciós eszközei, II. Rekordok és átlátszatlan típusok. A csomagok. Adatabsztrakció, egységbe zárás.

A rekord típus

- A direktszorzat és az unió típuskonstrukciók megvalósításához
- Összetett típus, amelynek komponens típusai különböző, már deklarált típusokhoz tartozhatnak
- Komponens: rekord mező
- A komponensekhez nevet rendelünk: szelektor
  - A komponensek nevének különbözőeknek kell lenniük

Egy példa

```
type Hónap is ( Január, … , December);

type Dátum is record
                        Év:    Integer;
                Hó:    Hónap;
                Nap:  Integer range 1 .. 31;
        end record ;
D: Dátum;

D.Év := 2003;   D.Hó := November;   D.Nap := 22;
```

Előre definiált műveletek

- A mezőelérés D.Év := D.Év + 1;
- Az értékadás és a (nem)egyenlőség vizsgálat (ha nem korlátozott a típus)

  if D1 /= D2 then D2 := D1; end if;
- A tartalmazás-ellenőrzés (altípusoknál)

  if D in Dátum then … end if;
- Az explicit konverzió (leszármaztatott típusnál)

  type Date is new Dátum;

  Today: Date := Date(D);

A mezők kezdőértéke

- A típus definíciójában megadhatjuk a típus objektumainak kezdőértékeit:

```
type Komplex is
 record
      Valós, Képzetes: Float := 0.0;
 end record;
C: Komplex;
I: Komplex := (0.0, 1.0)
```

Ha változhat a tárterület mérete...

```
Max_Méret: constant Integer := 10;
type Szöveg is
   record
      Hossz: Integer range 0 .. Max_Méret := 0;
      Érték: String( 1 .. Max_Méret  );
      Pozíció: Integer range 0 .. Max_Méret := 0;
   end record ;
```

Diszkriminánsos rekord

- Paraméteres típus:
  - A típusértékek paraméterezhetők
  - Néhány más típusnak is lehet diszkriminánsa Adában...
- Több diszkriminánsa is lehet egy típusnak
- A rekord-diszkrimináns diszkrét típusú

```
type Szöveg( Hossz: Natural ) is record
                Érték: String( 1 .. Hossz );
                Pozíció: Natural := 0;
                end record ;
```

Több diszkrimináns

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

type Táblázat ( Sorok, Oszlopok: Natural ) is
   record
      Sor_Címkék      : Tömb( 1..Sorok );
      Oszlop_Címkék   : Tömb( 1..Oszlopok );
      Adatok   :   Mátrix( 1..Sorok, 1..Oszlopok ) ;
   end record;
```

Megszorítatlan típus...

- A diszkriminánsos rekord egy megszorítatlan típus
- Olyan, mint a megszorítatlan indexhatárú tömb
  - Sok a hasonlóság a használatban is
- Nem használható objektum létrehozására

```
S: String;      -- hibás
Sz: Szöveg;     -- változódefiníciók
```

Nem teljesen meghatározott típus

- indefinite type
- Például
  - a megszorítás nélküli indexhatárú tömb típus
  - a diszkriminánsos rekord típus
- Nem lehet közvetlenül változót definiálni vele
  - nem ismerjük a méretét
  - de például helyes ez: X: T := (1,5,3);
- Nem lehet tömb elemtípusa sem
- Lehet viszont formális paraméter típusa

Rekord objektum létrehozása

- Meg kell szorítani a típust a diszkrimináns(ok) értékének megadásával
- Ez egy altípus definiálását jelenti
  - vagy altípus-deklarációval

    subtype Szöveg_10 is Szöveg(10);

    Sz: Szöveg_10;
  - vagy az objektum definíciójában (névtelen altípus…)

    Sz: Szöveg(10);
  - vagy a kezdőértékadás rögzítse az altípust

    Sz: Szöveg := (10, "abcdefghij", 0);

Diszkriminánsos rekord altípusai

- A diszkrimináns értékének megadásával altípusok hozhatók létre
  - Ha több diszkrimináns van, akkor mindnek értéket kell adni
- Ezek az altípusok már használhatók objektumok létrehozására
- Egy objektum altípusa nem változhat (≈)
- Beletartozás egy altípusba: *in* és *not in* operátorok

  if Sz in Szöveg_10 then ...

Mire jó a diszkriminánsos rekord

- Különböző méretű (memóriafoglalású) objektumokat tartalmazó típus

```
Sz_1: Szöveg(10);
Sz_2: Szöveg(100);

Sz_2 := Sz_1;     -- futási idejű hiba!
                  -- fordítás: warning
```

Mire jó a diszkriminánsos rekord

- Alprogramok általánosabb formában történő megírására
- function Szavak_Száma(Sz: Szöveg)
  return Natural
- Az aktuális paraméter határozza meg Sz méretét, azaz a Hossz diszkrimináns értékét a függvényben

Hivatkozás a diszkriminánsra

```
type Szöveg( Hossz: Natural ) is
      record
      Érték: String(1 .. Hossz);
                Pozíció: Natural := 0;
      end record ;
Sz: Szöveg(30);
```

- Az Sz rekord mezői:
  Sz.Hossz Sz.Érték Sz.Pozíció

```
Sz: Szöveg := (20, "  Vége az   órának! ", 4);
N: Natural := Szavak_Száma(Sz);

function Szavak_Száma( Sz: Szöveg ) return Natural is
      Szám: Natural := 0;
begin
      if Sz.Pozíció < 2  and then  Sz.Érték(1) /= ' ' then
           Szám := 1;
      end if;
      for I in Integer ' Max(2, Sz.Pozíció) .. Sz.Hossz loop
          if  Sz.Érték(I) /= ' ' and then  Sz.Érték(I-1) = ' ' then
              Szám := Szám + 1;
          end if;
      end loop;
      return Szám;
end;
```

Diszkrimináns alapértelmezett értéke

```
subtype Méret is Natural range 0..1000;
type Szöveg_D( Hossz: Méret := 10 ) is
 record
      Érték: String(1 .. Hossz);
      Pozíció: Natural := 0;
 end record ;

Sz_1: Szöveg_D(10);
Sz_2: Szöveg_D;
```

Mire jó, ha van?

- Nem kötelező megszorítást adni az objektumok létrehozásakor
- A diszkrimináns fel tudja venni az alapértelmezett értéket
- A diszkrimináns értéke - *az egész rekordra vonatkozó értékadással* - változtatható.
- Egyéb esetben a diszkrimináns értéke nem változtatható meg! Az altípus nem változhat!

```
Sz_1: Szöveg(10);     Sz_2, Sz_3: Szöveg(20);
Sz_1 := Sz_2;         -- futási idejű hiba
Sz_3 := Sz_2;
Sz_1 := (3, "abc", 0);  -- futási idejű hiba

Sz_D_1: Szöveg_D(10);      Sz_D_2: Szöveg_D(20);
Sz_D_1 := Sz_D_2;    -- futási idejű hiba
Sz_D_1 := (3, "abc", 0);  -- futási idejű hiba

Sz_D: Szöveg_D;              -- a Hossz 10, de változhat is!
Sz_D_1 := Sz_D;
Sz_D := Sz_D_1;
Sz_D := Sz_D_2;       Sz_D := (3, "abc", 0);
```

Veszély

Sz_D: Szöveg_D;

- A fordítóprogram a legnagyobb változathoz tartozó memóriát foglalja le
- Könnyen lehet futási idejű hiba a változó létrehozásakor
  - Storage_Error

Rugalmas szöveg típus?

```
type Rugalmas_Szöveg(Max: Integer := 0) is record
   Karakterek: String(1..Max ) := (others => '  ');
end record;
X: Rugalmas_Szöveg;

X := (10, (others => 'a'));
X := (20000, (others => 'b'));
```

- Az Integer túl nagy, nem tud annyit lefoglalni

A variáns rekord

- Biztonságos unió típus
  - szemben a Pascal vagy a C hasonló szerkezetével
- Diszkriminánsos rekord
  - a diszkrimináns értékén elágazó szerkezettel
- A reprezentációs reláció egy függvény
  - egy tárterület csak egyféleképpen értelmezhető
  - a típusértékek különböző szerkezetűek lehetnek
- Egy objektum szerkezetének megváltoztatására is van lehetőség

```
type Állapot is (Egyedülálló, Házas, Özvegy, Elvált);
subtype Név is String(1..25);
type Nem is (Nô, Férfi);
type Ember (Családi_Állapot: Állapot := Egyedülálló) is
    record
        Neve: Név;
        Neme: Nem;
        Születési_Ideje: Dátum;
        Gyermekek_Száma: Natural;
        case Családi_Állapot is
            when Házas          => Házastárs_Neve: Név;
            when Özvegy         => Házastárs_Halála: Dátum;
            when Elvált         => Válás_Dátuma: Dátum;
                                   Gyerekek_Gondozója: Boolean;
            when Egyedülálló => null ;
        end case;
    end record ;
```

Hugó: Ember(Házas);

Családi_Állapot, Neve, Neme, Születési_Ideje, Gyermekek_Száma, Házastárs_Neve

Eleonóra: Ember(Egyedülálló);

Családi_Állapot, Neve, Neme, Születési_Ideje, Gyermekek_Száma

Ödön: Ember(Özvegy);

Családi_Állapot, Neve, Neme, Születési_Ideje, Gyermekek_Száma, Házastárs_Halála

Vendel: Ember(Elvált);

Családi_Állapot, Neve, Neme, Születési_Ideje, Gyermekek_Száma, Válás_Dátuma, Gyerekek_Gondozója

Aladár: Ember; -- Egyedülálló

Családi_Állapot, Neve, Neme, Születési_Ideje, Gyermekek_Száma

- Helytelen (futási idejű hiba, altípus-megszorítás megsértése):

  Hugó.Válás_Dátuma, Aladár.Házastárs_Neve

Megszorított altípus létrehozása

subtype Egyedülállók is Ember(Egyedülálló);

Elek: Egyedülállók;

Eugénia: Ember(Egyedülálló);

- A szerkezet (azaz a diszkrimináns, és vele együtt az altípus) már nem változtatható meg
- Eleknek és Eugéniának sosem lesz házastársa.

Megszorítatlan altípus használata

Aladár: Ember; -- alapértelmezett Egyedülálló

Aladár := (Házas, ...);

Aladár := Elek;

Aladár := Hugó;

- A szerkezetét megváltoztathatjuk, diszkriminánsostul
- Csak úgy, ha az egész rekord értéket kap egy értékadásban

A különbségek…

- Aladár: Ember;
- Elek: Ember(Egyedülálló);

- Elek: Ember(Egyedülálló) := (Egyedülálló, …)
- Aladár: Ember := (Egyedülálló, …)
- Aladár: Ember := (Özvegy, …);

```
subtype Név_Hossz is Natural range 0..20;
type Foglalkozás is (Tanár, Diák);
subtype Átlag is Float range 0.0..5.0;

type Személy(  Betűszám: Név_Hossz := 0;
          Foglalkozása: Foglalkozás := Diák  ) is
       record
            Neve: String(1..Betűszám);
            case Foglalkozása is
           when Tanár => Oktatottak_Száma: Natural;
           when Diák  =>  Nappalis_E: Boolean := True;
                          Átlaga: Átlag;
            end case;
       end record;
```

```
type Alakzat is ( Téglalap, Négyzet, Kör );
type Geometriai_Alakzat ( S: Alakzat ) is   record
                case S is
                    when Téglalap   =>  A, B:  Float;
                    when Négyzet    =>  Oldal: Float;
                    when Kör        =>  Sugár: Float := 1.0;
                end case ;
            end record ;
function Terület( F: Geometriai_Alak ) return Float is
begin
      case F.S is
           when Téglalap  =>  return F.A * F.B;
           when Négyzet   =>  return F.Oldal ** 2;
           when Kör       =>  return 3.141592653589793 * F.Sugár ** 2;
      end case ;
end Terület;
```

Mire használható a diszkrimináns értéke

- Típusdefinícióban a diszkrimináns lehet
  - indexmegszorítás
  - a variáns rész kiválasztója
  - aktuális paraméter
- Minden esetben önmagában kell állnia, nem szerepelhet kifejezésben

Aktuális paraméter (1)

- Tömb megszorítása (Szöveg)

- A származtatott típusban a diszkrimináns lehet az ős típus paraméterének egy aktuális értéke:

```
type Táblázat ( Sorok, Oszlopok: Natural ) is
    record
        Sor_Címkék      : Float_Tömb( 1..Sorok );
        Oszlop_Címkék : Float_Tömb( 1..Oszlopok );
        Adatok: Integer_Mátrix( 1..Sorok, 1..Oszlopok ) ;
    end record ;
type Szorzótábla ( Méret: Positive ) is
            new Táblázat( Méret, Méret );
```

Aktuális paraméter (2)

- Ha a diszkriminánssal rendelkező típus újabb diszkriminánssal rendelkező típust tartalmaz, akkor ennek a diszkriminánsa függhet a külső típus diszkriminánsaitól (és csak attól függhet…)

```
type Könyv( Hossz: Positive ) is
    record
        ISBN: String(1..10);
        Tartalom: Szöveg(Hossz);
    end record ;
```

Rekordok 'Constrained attribútuma

- Legyen az A egy diszkriminánsos rekord típusú változó (vagy formális paraméter).
- Az A’Constrained értéke igaz, ha az A (illetve a neki megfeleltetett aktuális paraméter) egy konstans, egy érték, vagy egy megszorított változó.
- Azaz hamis akkor, ha az A egy megszorítást nem tartalmazó, a diszkrimináns alapértelmezett értékével létrehozott objektum, ami megváltoztathatja az altípusát.

Csomag

- Programok tagolásának egy eszköze
- Komponensek gyűjteménye
- Összetartozó „dolgok” egységbe zárása
  - Text_IO: Input-output műveletek gyűjteménye
  - (Adat)absztrakció támogatása
    - encapsulation, information hiding
    - típusértékhalmaz + műveletek

Egységbe zárás és absztrakció

- Egységbe zárás:
  ami egybe tartozik, az legyen egyben
- Absztrakció: vonatkoztassunk el a részletektől
  - rejtsük el a részleteket a többi komponens elől
- Moduláris programozást támogató nyelvek
  - Modula-2, CLU, Haskell
- Objektum-elvű nyelvek: osztály

Ismétlés

- Csomag: egy programegység
  - beágyazva vagy könyvtári egységként
- Specifikáció és törzs
  - szintaktikusan erősen szétválik
  - a specifikáció törzset nem tartalmazhat
  - külön fordítható: 2 fordítási egység
    - a GNAT specialitásai
    - a use utasítás: minősítés feloldása

Csomag - programegység

- **Specifikációs rész**
  külvilág számára nyújtott felület
  - **látható rész**
    kívülről használható komponensek specifikációja
  - **átlátszatlan rész** (opcionálisan megadható)
    reprezentációs információ a fordító számára
    (logikailag a törzshöz tartozik)
- **Törzs**
  megvalósítás, implementációs részletek

Specifikáció és törzs

```
package A is
    …               -- látható rész
private
    …               -- opcionális átlátszatlan rész
end A;

package body A is
    …               -- itt már törzsek is lehetnek
begin
    …               -- opcionális (inicializáló) utasítások
end A;
```

Mire használjuk a csomagokat?

- Alprogramgyűjtemény
- Egyke objektum megvalósítása (singleton)
  - pl. absztrakt állapotgép (Abstract State Machine)
- Típusmegvalósítás
  - pl. absztrakt adattípus (Abstract Data Type)

Alprogramok gyűjteménye

```
package Trigonometrikus is
    function Sin ( X: Float ) return Float;
    function Cos ( X: Float ) return Float;
    function Tan ( X: Float ) return Float;
    …
end Trigonometrikus;
```

- A törzsben az alprogramok megvalósítása…

Egyke objektum megvalósítása (1)

- Abstract State Machine - egységbe zárva
  - részletek elrejtve

```
package Stack is
    subtype Item is Integer;                -- később majd param.
    procedure Push( X: in Item );
    procedure Pop( X: out Item );
    function Top return Item;
    function Is_Empty return Boolean;
    …
end Stack;
```

Egyke objektum megvalósítása (2)

```
package body Stack is
    Capacity: constant Positive := 100;     -- ez is lehet paraméter
    Data: array( 1..Capacity ) of Item;
    Stack_Pointer: Natural := 0;
    procedure Push( X: in Item ) is
    begin
        Stack_Pointer := Stack_Pointer + 1;
        Data(Stack_Pointer) := X;
    end Push;
    …
end Stack;
```

Egyke objektum használata (1)

```
with Stack, Ada.Command_Line, Ada.Integer_Text_IO;
procedure Revert is
   N: Integer;
begin
   for I in 1..Ada.Command_Line.Argument_Count loop
      N := Integer'Value(Ada.Command_Line.Argument(I));
      Stack.Push( N );
   end loop;
   while not Stack.Is_Empty loop
       Stack.Pop( N );
       Ada.Integer_Text_IO.Put(N);
   end loop;
end Revert;
```

Egyke objektum használata (2)

```
with Stack, Ada.Command_Line, Ada.Integer_Text_IO; use Stack;
procedure Revert is
   N: Integer;
begin
   for I in 1..Ada.Command_Line.Argument_Count loop
      N := Integer'Value(Ada.Command_Line.Argument(I));
      Push( N );
   end loop;
   while not Is_Empty loop
       Pop( N );
       Ada.Integer_Text_IO.Put(N);
   end loop;
end Revert;
```

Egyke objektum

- Singleton
  - lesz még ilyen (taszk, védett egység)
- Mit tegyek, ha több vermet is szeretnék használni a programban?
  - sablonok…
- Verem típus készítése
  - típusműveletek összegyűjtése, összecsomagolása a típusértékhalmaz definíciójával

Típusmegvalósítás

```
package Matrices is
    type Matrix is array
     ( Integer range <>, Integer range <> ) of Float;
   function ”+” ( A, B: Matrix ) return Matrix;
   function ”-” ( A, B: Matrix ) return Matrix;
   function ”*” ( A, B: Matrix ) return Matrix;
   …
end Matrices;
```

- A törzsben a típusműveletek megvalósítása…

A Verem típus

```
package Stacks is
    subtype Item is Integer;              -- paraméter lesz
    Capacity: constant Positive := 100;          -- akár ez is
    type Item_Array is array( 1..Capacity ) of Item;
    type Stack is record
                        Data: Item_Array;
                        Stack_Pointer: Natural := 0;
                    end record;
    procedure Push( V: in out Stack; X: in Item );
    …
end Stacks;
```

Diszkriminánsos rekorddal

```
package Stacks is
    subtype Item is Integer;
    type Item_Array is array( Integer range <> ) of Item;
    type Stack(Capacity: Positive ) is record
                        Data: Item_Array(1..Capacity);
                        Stack_Pointer: Natural := 0;
                    end record;
    procedure Push( V: in out Stack; X: in Item );
    …
end Stacks;
```

Absztrakt adattípus

- Az előző példában az egységbe zárás elve (encapsulation) teljesül
- Mi van az implementációs részletek elrejtésével (information hiding)?
- Jó lenne elrejteni a Verem típus reprezentációját a külvilág elől!

Az absztrakció fontossága

- A szoftvertermék minőségi mutatóinak javítása
  - egységbe zárás: olvashatóság (átláthatóbb kód)
  - információ elrejtés: módosíthatóság
  - karbantarthatóság
  - modularizáció: könnyebb/gyorsabb elkészíteni

Átlátszatlan típus

- A reprezentációt elrejtem a csomag specifikációjának átlátszatlan részében
- A használó programegységek nem férnek hozzá a reprezentációhoz, ezért nem is függnek tőle
  - csak a műveleteken keresztül használják a típust
- A típus részleges nézetét látják, a teljes nézet az átlátszatlan (privát) részben van
- A fordítóprogram használhatja és használja is a reprezentációt (méret), ezért van a specifikációban

```
package Stacks is
    subtype Item is Integer;          -- ebből lesz majd paraméter
    type Stack is private;
    procedure Push( V: in out Stack; X: in Item );
    …
private
    Capacity: constant Positive := 100;
    type Item_Array is array( 1..Capacity ) of Item;
    type Stack is record
                      Data: Item_Array;
                      Stack_Pointer: Natural := 0;
                  end record;
end Stacks;
```

```
package Stacks is
    subtype Item is Integer;          -- ebből lesz majd paraméter
    type Stack (Capacity: Positive) is private;
    procedure Push( V: in out Stack; X: in Item );
    …
private
    type Item_Array is array( Integer range <> ) of Item;
    type Stack( Capacity: Positive ) is record
                        Data: Item_Array(1..Capacity);
                        Stack_Pointer: Natural := 0;
                    end record;
end Stacks;
```

Az átlátszatlan típus műveletei

- Ami a részleges nézetből kiderül a reprezentációról
  - az utóbbiban a Capacity:Positive mező
- A részleges nézetből származó műveletek
  := = /=
- A csomag által specifikált alprogramok,
  - melyek valamelyik paramétere vagy visszatérési értéke olyan típusú…

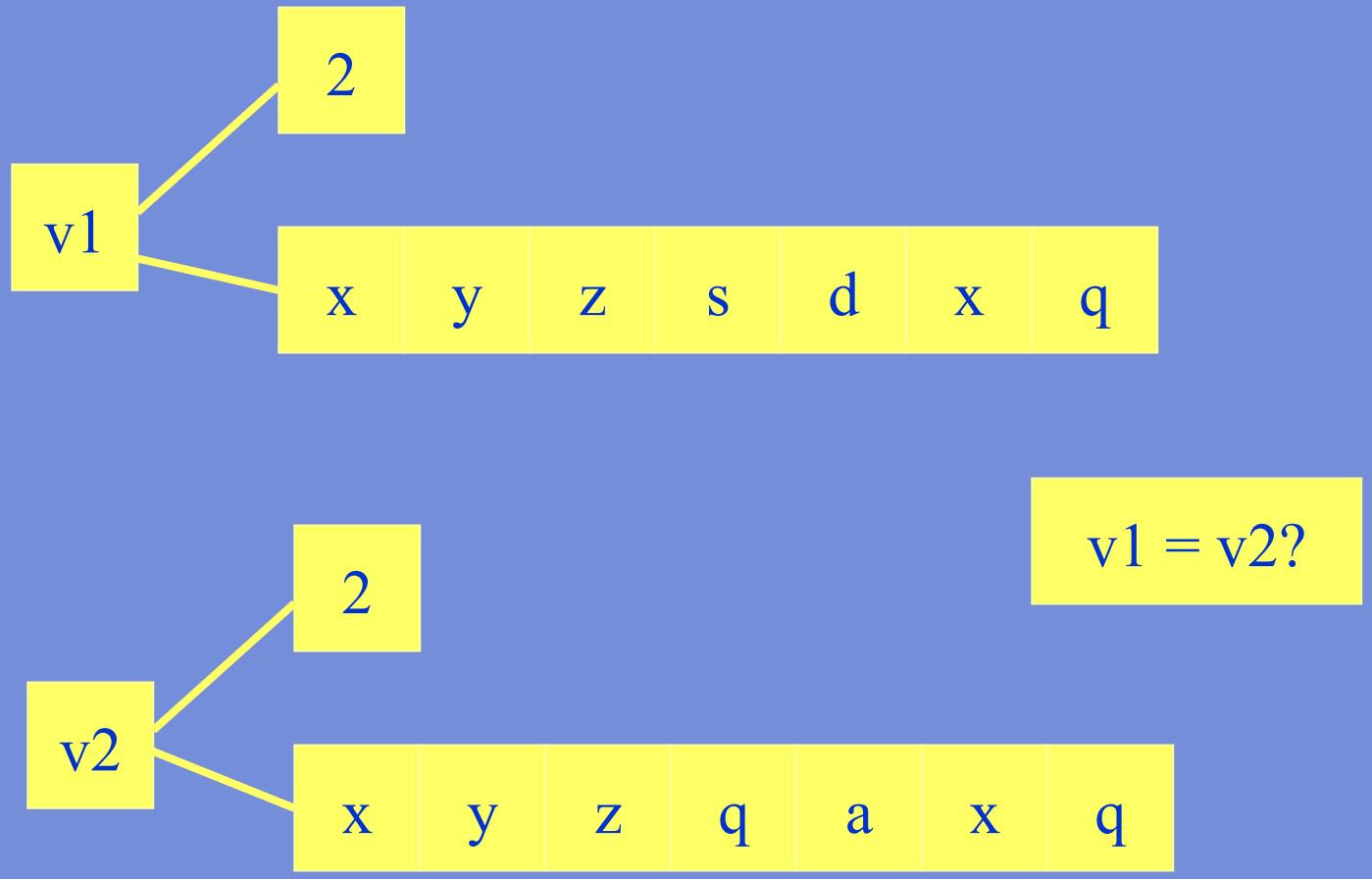
Korlátozott típusok

- Az értékadás és a (nem)egyenlőtlenség vizsgálata letiltható: nincs := = /=
- A típus definíciójában szerepel a limited

type R is limited record … end record;

- Egy nézet is lehet korlátozott, például a részleges…

Vermek összehasonlítása



Korlátozott átlátszatlan típus

- Mikor egyenlő két verem?
- Mit jelent az értékadás láncolt adatszerkezet esetén?
- A C++ nyelvben felüldefiniáljuk az értékadás és egyenlőségvizsgálat operátorokat
- Az Adában az értékadás nem operátor
  - A limited kulcsszót használhatjuk az átlátszatlan típus részleges nézetének megadásakor
  - Az Ada95-ben van még Controlled is

```
package Stacks is
    subtype Item is Integer;
    type Stack (Capacity: Positive) is limited private;
    procedure Push( V: in out Stack; X: in Item );
    …
private
    type Item_Array is array( Integer range <> ) of Item;
    type Stack(Capacity: Positive ) is record
                        Data: Item_Array(1..Capacity);
                        Stack_Pointer: Natural := 0;
                    end record;
end Stacks;
```

Hierarchikus csomagszerkezet

- Logikailag egységbe zárás
- Fizikailag szétbontás, modularizálás

- Gyermek csomagok
  - titkos és nyilvános gyermek

Késleltetett definíciójú konstans

```
package Komplexek is
    type Komplex is private;
    I : constant Komplex;
    function "+"(X,Y:Komplex) return Komplex;
    function "-"(X,Y:Komplex) return  Komplex;
    …
private
    type Komplex is …
    I: constant Komplex := (0.0, 1.0);
end Komplexek;
```

Példa csomag specifikációjára

```
package Text_IO is
    procedure Put_Line( S: in String );
    procedure New_Line( N: in Positive := 1 );
    procedure Get_Line( S: out String;
                        L: out Natural );
    procedure Put( S: in String );
    …
end Text_IO;
```

Specifikációk igen, törzsek nem!

Példa csomag törzsére

```
package body Text_IO is
    procedure Put_Line( S: in String ) is
        …
    begin
        …
    end Put_Line;
    …
end Text_IO;
```

Törzsek!

Programszerkezet és csomagok

- Egy csomagba beteszünk összetartozó dolgokat
  - pl. beágyazunk más programegységeket (alprogramokat, csomagokat)
- A csomagokat is beágyazhatjuk más programegységekbe (alprogramokba, csomagokba)
- Készíthetünk csomagokból könyvtári egységet is (azaz külön fordítási egységeket)

Programegységek beágyazása egy csomagba

- Specifikációt specifikációba vagy törzsbe
- Törzset törzsbe

- Alprogram törzsét vagy csomag törzsét nem írhatjuk egy csomag specifikációs részébe
- Írhatunk viszont specifikációt (pl. egy beágyazott csomag specifikációját) a csomag törzsébe...

```
package A is
    procedure Eljárás( Formális: in String );
    (sub)type T is …
    V: Integer;
    package B is … end B;
end A;

package body A is
    procedure Eljárás( Formális: in String ) is … end Eljárás;
    package body B is … end B;
    procedure Segéd( Formális: out Integer );
    (sub)type S is …
    W: Float;
    package C is … end C;
    package body C is … end C;
    procedure Segéd( Formális: out Integer ) is … end Segéd;
end A;
```

exportált (kívülről látható) dolgok specifikációja

rejtett dolgok, specifikációk és törzsek

Csomag definíciója

- Csomag specifikációjának és törzsének megadása
- Szerepelhet deklarációs részekben
  - alprogram deklarációs részében
  - declare utasítás deklarációs részében
  - másik csomag törzsében (annak deklarációs részében)
  - …
- Másik csomag specifikációs részével vigyázni kell!
- Szerepelhet külön könyvtári egységként is

Csomag beágyazása, pl. alprogramba

```
function F ( Formális: T ) return T is
    …
    package A is … end A;
    …
    package body A is … end A;
    …
begin
    …
end F;
```

Csomag, mint könyvtári egység

- Ha nem ágyazzuk be a csomagot
- Ekkor két fordítási egységből áll
  - külön fordítható (és fordítandó általában) a csomag specifikációja és a csomag törzse
- A csomag könyvtári egységet használó más fordítási egység csak a csomag specifikációs részétől függ
  - Ha lefordítottam a csomag specifikációját, akkor már használhatom is, a tőle függő fordítási egységeket lefordíthatom, mielőtt a csomag törzsét elkészíteném…
  - … a törzs csak az összeszerkesztéshez kell majd…

Miért fontos ez?

- Több szabadság a program komponenseinek elkészítési sorrendjében
  - A fordítás (ellenőrzés) lehetséges!
- Team-munka támogatása, párhuzamos fejlesztés
- Egy csomag törzse, azaz az implementációs részletek megváltoztathatók az egész program újrafordítása nélkül, csak összeszerkeszteni kell
  - Gyorsabb fordítás
  - Nagy rendszerek készítése…
  - Hibák javítása nem okoz inkonzisztenciát

Hogyan kell használni a csomag könyvtári egységet?

- a with és a use utasítás

```
with Text_IO;
procedure A is
begin
    Text_IO.Put_Line(”Szia!”);
end;
```

- **A használatot előre jelezni kell - with**
- **Mint amikor alprogramot**
- **A komponensekhez hozzáférek minősítéssel**

Hogyan kell használni csomag könyvtári egységet?

- a with és a use utasítás

```
with Text_IO; use Text_IO;
procedure A is
begin
    Put_Line(”Szia!”);
end;
```

- **A komponensekhez való hozzáférésnél a minősítés feloldható - use**
- **Csak csomag könyvtári egységre van értelme use-t írni**

A GNAT fordító és a csomagok

- A könyvtári egységként szereplő **A** csomag
  - specifikációja: **a.ads**
  - törzse: **a.adb**
- gnatmake: ez már az összeszerkesztő is
  - de azért használható ads fordítására is…

```
$ gnatmake a.ads
gcc -c a.ads
No code generated for file a.ads (package spec)
gnatmake: ”a.ads" compilation error
$
```