5. előadás

Parametrikus polimorfizmus.
Generikus programozás.
Az Ada sablonok.

Polimorfizmus

- Többalakúság
- Programozási nyelvek:
  több típussal való használhatóság
  - Alprogram
  - Adatszerkezet
  - Változó
- Kifinomultabb típusrendszerek

A polimorfizmus fajtái

- Parametrikus polimorfizmus
  - erről lesz most szó
- Altípusos polimorfizmus
  - objektumelvű nyelvek
  - általában öröklődés alapján
- Esetleges (ad-hoc)
  - túlterhelés, típuskényszerítés

Parametrikus polimorfizmus

- Típussal való paraméterezés
- Sok nyelvben
  - Modern funkcionális nyelvek (ML, …)
  - Ada, C++, Java
  - CLU, Eiffel stb.
- Generikus programozás

Generikus programozás

- Generic programming
- Algoritmusok és adatszerkezetek általános, típusfüggetlen leprogramozása
- Generikus rendezés, generikus verem
- Sablon: generic, template

Generikus felcserélés (C++)

```
template <typename T>
void swap( T& a, T& b )
{
    T c = a;
    a = b;
    b = c;
}
```

Generikus felcserélés (Ada)

```
generic
    type T is private;
procedure Swap( A, B: in out T );

procedure Swap( A, B: in out T ) is
    C: T := A;
begin
    A := B;
    B := C;
end Swap;
```

Generikus verem (C++)

```
template <typename Element>
class Stack
    public:
        Stack ( int capacity );
        void push ( const Element& e );
        Element pop();
        bool empty() const;
        …    // további műveletek
    private:
        …    // reprezentáció
};
…    // műveletek megvalósítása
```

Generikus verem (Ada)

```
generic
    type Element is private;
package Stacks is
    type Stack( Capacity: Positive ) is limited private;
    procedure Push ( S: in out Stack; E: in Element );
    procedure Pop ( S: in out Stack; E: out Element );
    function Is_Empty ( S: Stack ) return Boolean;
    ...   -- további műveletek
private
    ...   -- reprezentáció
end Stacks;
…   -- műveletek megvalósítása a csomag törzsében
```

Sablonok használata

- Bizonyos nyelvekben (Ada, C++) konkrét példányokat kell létrehozni belőlük
  - A sablon csak példányosításra használható
- Más nyelvekben (funkcionális ny., Java) erre nincs szükség
  - Mindig ugyanaz a kód hajtódik végre, nincs példányosítás

Példányosítás

- A sablon programegységekre alkalmazható művelet
  - Alprogram programegység: hívható
  - Csomag programegység: komponensei elérhetők
- Egy sablon alapján elkészít egy „igazi” programegységet
  - Alprogramot, csomagot, osztályt
- Emiatt tekintjük a generikus programozást generatívnak

Generatív programozás

- Generative programming
- Program generálása programmal
- Egy speciális fajtája:
  a generikus programozás
- Egy másik fajtája:
  a template metaprogramming
- A C++ template nem csak generikus, hanem generatív programozást támogató eszköz is

Verem példányosítása (C++)

```
…
Stack<int> stack(10);
stack.push(4);
stack.push(2);
if ( ! stack.empty() )
{
    stack.pop();
}
…
```

Verem példányosítása (Ada)

```
package Int_Stacks is new Stacks(Integer);
S: Int_Stacks.Stack(10);
E: Integer;
…
Int_Stacks.Push( S, 4 );
Int_Stacks.Push( S, 2 );
if  not Int_Stacks.Is_Empty( S )  then
     Int_Stacks.Pop( S, E );
end if;
```

Ha use-t használunk

```
package Int_Stacks is new Stacks(Integer);
use Int_Stacks;
S: Stack(10);
E: Integer;
…
Push( S, 4 );
Push( S, 2 );
if  not Is_Empty( S )  then
    Pop( S, E );
end if;
```

Felcserélés példányosítása (C++)

- Alprogramoknál implicit példányosítás
- Típuskikövetkeztetés alapján

```
int x = 4, y = 2;
double u = 1.2, v = 2.1;
swap(x,y);
swap(u,v);
```

Felcserélés példányosítása (Ada)

```
procedure Int_Swap is new Swap(Integer);
procedure Float_Swap is new Swap(Float);
X: Integer := 4;  Y: Integer := 2;
U: Float := 1.2;  V: Float := 2.1;
…
Int_Swap(X,Y);
Float_Swap(U,V);
```

Különbségek a példányosításban

- Alprogram
  - Ada: explicit, C++: implicit
- Csomag/osztály
  - Ada: önállóan áll, C++: „on-the-fly”
- Lusta példányosítás: C++
  - Egy sablonosztályból csak a használt alprogramok példányosulnak

Ada: programegység definiálása példányosítással

- Alprogramsablonból alprogram, csomagsablonból csomag hozható létre

```
procedure Int_Swap is new Swap(Integer);
package Int_Stacks is new Stacks(Integer);
```

- Ezekben a definíciókban nem adunk meg törzset (a törzs generálódik, makrószerűen)
- Beágyazva vagy könyvtári egységként

Példányosítás beágyazva

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Colours is
      type Colour is (Red, Blue, Yellow);
      package Colour_IO is new Enumeration_IO(Colour);
      C: Colour;
begin
       Colour_IO.Get( C );
       Colour_IO.Put( C );
      …
end Colours;
```

Példányosítás könyvtári egységként

```
with Ada.Text_IO;
package Ada.Integer_Text_IO is
    new Ada.Text_IO.Integer_IO(Integer);
```

- Egy fordítási egység
- Használat:

```
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;
```

Tipikus hiba a use kapcsán

- A sablon programegységek komponenseit nem lehet elérni
  - A sablon csak példányosításra való
- Ezért nincs értelme (és tilos) use utasítást adni egy sablonra (pl. a „**use Stacks;**” helytelen)
  - Sem csomagsablonra, sem alprogramsablonra
  - Hasonlóan az alprogramokhoz
  - Csomagsablonból létrehozott példányra már lehet use-t használni (pl. a „**use Int_Stacks;**” helyes)

Sablonok definiálása

- Sablon: egy programegység
  - Példányosításra való programegység
- Felépítése: specifikáció és törzs
  - Szintaktikusan szétváló
- Definiálható beágyazva és könyvtári egységként is
- Alprogramsablon és csomagsablon

Sablon specifikációja és törzse (1)

```
generic
     type T is private;
procedure Swap( A, B: in out T );


procedure Swap( A, B: in out T ) is
     C: T := A;
begin
     A := B;
     B := C;
end Swap;
```

Mindig különálló, alprogram-sablon esetén is!

Sablon specifikációja és törzse (2)

```
generic
    type Element is private;
package Stacks is
    …
end Stacks;


package body Stacks is
    …
end Stacks;
```

Sablon könyvtári egység

- Két fordítási egység a specifikáció és a törzs
  - Mind alprogram-, mind csomagsablon esetén
  - Mint a csomag könyvtári egység
  - GNAT: .ads és .adb forrásfájlok
- Gyakoribb, mint a beágyazás

Sablon definíciója beágyazva (1)

```
package Ada.Text_IO is
    …
    generic
        type Num is range <>;
    package Integer_IO is
        …
        procedure Get ( Item  : out Num;
                        Width : in  Field := 0 );
        …
    end Integer_IO;
    …
end Ada.Text_IO;
```

Sablon definíciója beágyazva (2)

```
package body Ada.Text_IO is
    …
    package body Integer_IO is
        …
        procedure Get ( Item  : out Num;
                        Width : in  Field := 0 ) is
            …
        end Get;
        …
    end Integer_IO;
    …
end Ada.Text_IO;
```

Sablon definíciója beágyazva (3)

```
with Ada.Text_IO;
package Ada.Integer_Text_IO is
     new Ada.Text_IO.Integer_IO(Integer);


with Ada.Text_IO; use Ada.Text_IO;
procedure Számos is
     package Pos_IO is new Integer_IO(Positive);
begin
     Pos_IO.Put( 1024 );
end Számos;
```

Sablon specifikációjának tördelése

```
generic
    type Element is private;
package Stacks is
    type Stack ( Capacity: Positive ) is limited private;
    …
end Stacks;

generic
    type T is private;
procedure Swap( A, B: in out T );
```

Sablonok formális paraméterei

- Típus
- C++ és Ada:
  objektum
  sablon osztály (C++), sablon csomag (Ada)
- Ada: alprogram

Sablon típusparamétere

```
generic
    type T is private;
…
```

- Paraméter típus: átlátszatlan
- A sablonban nem ismerem a T-nek megfeleltetett aktuálisnak a szerkezetét

Átlátszatlan

```
generic
    type Element is private;
package Stacks is
    type Stack(Capacity: Positive) is limited private;
    …
private
    …
    type Stack(Capacity: Positive) is …;
end Stacks;
```

A Stacks-en belül

A Stacks-en kívül

Sablon objektumparamétere (C++)

```
template <typename Element, int capacity>
class Stack
{
    public:
        Stack();
        void push ( const Element& e );
    …
};

Stack<int,10> s;
```

Sablon objektumparamétere (Ada)

```
generic
    type Element is private;
    Capacity: Positive;
package Stacks is
    type Stack is limited private;
    procedure Push ( S: in out Stack; E: in Element );
    …
end Stacks;

package Int_Stacks is new Stacks(Integer,10);
S: Int_Stacks.Stack;
```

Sablonparaméter versus típus-/konstruktorparaméter

```
generic
        type Element is private;
        Capacity: Positive;
package Stacks is
        type Stack is limited private;
        …
```

```
package IStacks is
        new Stacks(Integer,10);
use IStacks;
S1, S2: Stack;
```

```
generic
        type Element is private;
package Stacks is
        type Stack(Capacity: Positive) is limited private;
        …
```

```
package IStacks is
        new Stacks(Integer);
use IStacks;
S1: Stack(5);   S2: Stack(10);
```

Sablon kimenő objektumparamétere

- Az Adában a sablon objektumparamétere lehet in, out és in out módú is
- Olyan, mint az alprogramok paraméterei
- Alapértelmezett mód: in
- Legtöbbször in módút használunk

Sablon objektumparaméterének alapértelmezett értéke

- Csak bemenő paraméter esetén

```
generic
      type Element is private;
      Capacity: Positive := 10;
package Stacks is
      …
end Stacks;

package I30_Stacks is new Stacks(Integer,30);
package I10_Stacks is new Stacks(Integer);
```

Alprogrammal való paraméterezés

- Funkcionális nyelvekben magától értetődő
- Procedurális nyelvekben nehézkesebb
  - Szintaktikus, szemantikus, hatékonysági kérdések
- Alprogram átadása alprogramnak
  - Lokális alprogram? Csonk. (Modula-2)
  - Csak globális alprogram: C, Ada 95…
- Alprogramra mutató mutató átadása (C++, Ada 95)
- Funktor (C++ szépen, Java nehézkesen)
- Sablon alprogramparaméterei: Ada

Ada sablonok alprogramparaméterei

- Sablon alprogramnak és sablon csomagnak
- A sablon példányosításakor adjuk meg az aktuális műveletet
- Nem annyira rugalmas, mint az „alprogramnak alprogramot” lehetőség
  - Fordítás közben „derül ki” az aktuális, meglehetősen korai kötés
- Gyakran egy típus művelete a paraméter

Példa: maximumkeresés

```
generic
    type T is private;
    with function "<" (A, B: T) return Boolean;
function Maximum ( A, B: T ) return T;

function Maximum ( A, B: T ) return T is
begin
    if A < B then return B; else return A; end if;
end Maximum;
```

Tördelés és with

```
generic
    type T is private;
    with function "<" (A, B: T) return Boolean;
function Maximum ( A, B: T ) return T;

function Maximum ( A, B: T ) return T is
begin
    if A < B then return B; else return A; end if;
end Maximum;
```

Sablon alprogramparaméterének alapértelmezett értéke (triviális eset)

```
generic
    type T is private;
    with function "<" (A, B: T) return Boolean is <>;
function Maximum ( A, B: T ) return T;

function Maximum ( A, B: T ) return T is
begin
    if A < B then return B; else return A; end if;
end Maximum;
```

Sablon alprogramparaméterének használata

```
with Maximum;
procedure Max_Demo is
    function I_Max is new Maximum( Integer, "<" );
    function I_Min is new Maximum( Integer, ">" );
    function F_Max is new Maximum( Float );
  …
end Max_Demo;
```

Ugyanez a C++ nyelvben

```
template <typename T>
T maximum( T a, T b )
{
    if ( a < b ) return b; else return a;
}


int m = maximum( 46*32, 64*23 );        // az aktuális: int
```

Különbségek első pillantásra

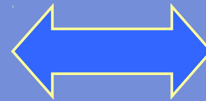
- A C++ nyelvben egyszerűbb:
  nincs a sablonnak alprogramparamétere
- Az Adában ugyanaz a sablon több feladat elvégzésére használható
  - ha más paramétert adok meg

Sablonszerződés

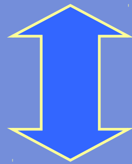
- A sablon specifikációja megadja, hogy hogyan lehet a sablont használni
  - Azaz hogyan lehet példányosítani…
- A sablon specifikációja egy „szerződés” a sablon törzse és a példányosítás között
- Ha a példányosítás betartja a szerződést, akkor a sablonból generált kód helyes lesz

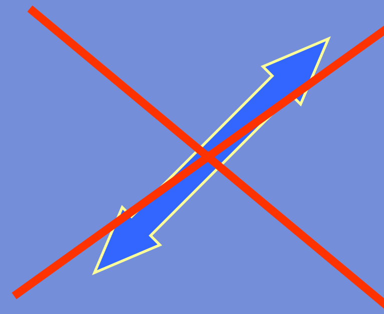
Függések

A sablon specifikációja



Példányosítás





?

Nem kell!!

A sablon törzse

A szerződés kimondja:

- A sablon törzse nem használhat mást, csak amit a sablon specifikációja megenged neki
- A példányosításnak biztosítania kell mindent, amit a sablon specifikációja megkövetel tőle

A sablon törzse betartja

```
generic
    type T is private;
    with function "<" (A, B: T) return Boolean is <>;
function Maximum ( A, B: T ) return T;

function Maximum ( A, B: T ) return T is
begin
    if A < B then return B; else return A; end if;
end Maximum;
```

legális

A példányosítás betartja

```
with Maximum;
procedure Max_Demo is
    function I_Max is new Maximum( Integer, "<" );
    function I_Min is new Maximum( Integer, ">" );
    function F_Max is new Maximum( Float );
   …
end Max_Demo;
```

Milyen a szerződés a C++ nyelvben?

- A sablon „specifikációja” az egész definíció
  - A belsejét is ismerni kell, hogy tudjuk, hogyan lehet példányosítani
- Ezért is írjuk az egészet fejállományba
- Az információ elrejtésének elve sérül
  - Az olvashatóság rosszabb
  - A bonyolultság nagyobb
  - A módosítások bajt okozhatnak (lusta példányosítás)
  - A fejlesztés nehezebb

A példánkban:

```
template <typename T>
T maximum( T a, T b )
{
    if ( a < b ) return b; else return a;
}



int m = maximum( 46*32, 64*23 );        // az aktuális: int
```

Taktika

- A sablon specifikációjában kérjünk…
  - olyan keveset a példányosítástól, amennyire csak lehet,
  - de amivel még meg tudjuk írni a törzset
- Így nő a sablon újrafelhasználhatósága
- Ellenérv: hatékonyság, könnyebb használat

Hogyan kérjünk kevesebbet a példányosítástól? (1)

```
with Maximum, Ada.Text_IO;
procedure Max_Demo is
    function Smaller ( A, B: String ) return Boolean is
        …
    end Smaller;
    -- fordítási hibát okoz:
    function S_Max is new Maximum( String, Smaller );
begin
    Put_Line( S_Max( "Hi" , "Salut" ) );
end Max_Demo;
```

Nem teljesen definiált típust is megengedünk

Így már lehet az aktuális a String

```
generic
     type T(<>) is private;
     with function "<" (A, B: T) return Boolean is <>;
function Maximum ( A, B: T ) return T;

function Maximum ( A, B: T ) return T is
begin
     if A < B then return B; else return A; end if;
end Maximum;
```

Az új szerződés értelmében:

- A példányosításkor választhatjuk az aktuális paramétert
  - teljesen meghatározott, valamint
  - nem teljesen meghatározott típusúnak is
- A sablon törzsében nem hozhatok létre objektumot ebből a típusból

Ha az aktuális lehet indefinit

```
generic
    type T(<>) is private;
procedure Swap( A, B: in out T );

procedure Swap( A, B: in out T ) is
    C: T := A;
begin
    A := B;
    B := C;
end Swap;
```

Fordítási hiba

Hogyan kérjünk kevesebbet a példányosítástól? (2)

```
with Maximum, Ada.Text_IO;
procedure Max_Demo is
     type R is limited record … end record;
     function Smaller ( A, B: R ) return Boolean is
          …
     end Smaller;
     -- fordítási hibát okoz:
     function R_Max is new Maximum( R, Smaller );
    …
end Max_Demo;
```

Korlátozott típust is megengedünk

Így már lehet az aktuális a korlátozott rekord (R)

```
generic
    type T is limited private;
    with function "<" (A, B: T) return Boolean is <>;
function Maximum ( A, B: T ) return T;

function Maximum ( A, B: T ) return T is
begin
    if A < B then return B; else return A; end if;
end Maximum;
```

Ha az aktuális lehet korlátozott

```
generic
     type T is limited private;
procedure Swap( A, B: in out T );

procedure Swap( A, B: in out T ) is
     C: T := A;
begin
     A := B;
     B := C;
end Swap;
```

Összefoglalva

A sablon alprogram-paraméterei mellett

Formális	*Aktuális*	*Törzsben*
type T is private;	definit nem korlátozott	obj. létrehozás := = /=
type T(<>) is private;	(in)definit nem korlátozott	:= = /=
type T is limited private;	definit (nem) korlátozott	obj. létrehozás
type T(<>) is limited private	(in)definit (nem) korlátozott	

Korlátozott indefinit típust is megengedünk

```
generic
    type T(<>) is limited private;
    with function "<" (A, B: T) return Boolean is <>;
function Maximum ( A, B: T ) return T;

function Maximum ( A, B: T ) return T is
begin
    if A < B then return B; else return A; end if;
end Maximum;
```

Példányosíthatom akár így is

```
with Maximum, Stacks;
procedure Max_Demo is
    package IStacks is new Stacks(Integer); use IStacks;
    function Smaller ( A, B: Stack ) return Boolean is
    begin
        if Is_Empty(B) then return False;
        else return Is_Empty(A) or else Top(A) < Top(B);
        end if;
    end Smaller;
    function S_Max is new Maximum( Stack, Smaller );
  …
```

Sablon típusparaméterei

- Átlátszatlan típus (private + műveletek)
- Könnyítés a példányosításon, szigorítás a sablontörzs lehetőségein:

  (<>), limited, tagged, abstract
- A típus szerkezetére tett megszorítás: **típusosztály**

Típusosztályok használata (1)

```
generic
    type Element is private;
    type Index is (<>);
    type Vector is array (Index range <>) of Element;
    with function Op( A, B: Element ) return Element;
    Start: in Element;
function Fold ( V: Vector ) return Element;
```

Típusosztályok használata (2)

```
function Fold ( V: Vector ) return Element is
    Result: Element := Start;
begin
    for I in V'Range loop
        Result := Op( Result, V(I) );
    end loop;
    return Result;
end Fold;
```

Típusosztályok használata (3)

```
with Fold;
procedure Fold_Demo is
    type T is array ( Integer range <> ) of Float;
    function Sum is new Fold( Float, Integer, T, "+", 0.0 );
    function Prod is new Fold( Float, Integer, T, “*", 1.0 );


…
end Fold_Demo;
```

Néhány fontos típusosztály-megjelölés

- Egész típus

  type T is range <>;

- Diszkrét típus

  type T is (<>);

- Moduló, lebegőpontos, fixpontos, decimális fixpontos
- Tömb, mutató, diszkriminánsos, származtatott, kiterjesztett

Mire jó a típusosztály

- A sablon törzsében használhatók a típusosztályra jellemző műveletek
  - Diszkrét: lehet tömb indextípusa
  - Tömb: indexelhető, van ‘Range attribútuma
- A példányosításkor csak olyan aktuális paraméter választható
  - Integer
  - type T is array (Integer range <>) of Float

Ha a paraméter egy tömb típus

- Gyakran az elem- és az indextípus is paraméter
  - Persze ilyenkor az indextípus „(<>)” vagy „range <>”
- Nem mindegy, hogy megszorított, vagy megszorítás nélküli indexelésű

```
type T is (Index range <>) of Element;
type T is (Index) of Element;
```

  - nem feleltethetők meg egymásnak sem így, sem úgy
- Lehet többdimenziót is

Kétdimenziós, lebegőpontos

```
generic
     type Element is digits <>;
     type Index is (<>);
     type Matrix is array (Index range <>, Index range <>)
                                 of Element;
package Matrix_Arithmetics is
     function "*" (A, B: Matrix ) return Matrix;
     …
end Matrix_Arithmetics;
```

Sablonok csomagparamétere

```
generic
    with package Arithmetics is
        new Matrix_Arithmetics(<>);
    use Arithmetics;
function Inverse( M: Matrix ) return Matrix;
```

- Példányosításnál az aktuális paraméter a Matrix_Arithmetics csomagsablonból létrehozott példány kell, hogy legyen

Sablon a sablonban

```
generic
    …
package …
    generic
        …
    function …
end;
```

- Sablonok egymásba ágyazhatók

```
generic
   type Element is private;
package Stacks is

   type Stack( Capacity: Positive ) is limited private;

   procedure Push( S: in out Stack; Item: in Element );
   ...

   generic
      with procedure Process_Element( Item: in Element );
   procedure For_Each( S: in Stack );

private
   type Vector is array( Natural range <> ) of Element;
   type Stack( Capacity: Positive ) is record
               Last: Natural := 0;
               Data: Vector(1..Capacity);
            end record;
end Stacks;
```

```
package body Stacks is

   procedure Push( S: in out Stack; Item: in Element ) is
   begin
      S.Last := S.Last + 1;
      S.Data(S.Last) := Item;
   end Push;

   ...

   procedure For_Each( S: in Stack ) is
   begin
      for I in S.Data'First .. S.Last loop
         Process_Element( S.Data(I) );
      end loop;
   end For_Each;

end Stacks;
```

```ada
with Stacks, Ada.Integer_Text_IO;
procedure Stack_Demo is

   package IStacks is new Stacks(Integer);

   procedure Put_Int( Item: Integer ) is
   begin
      Ada.Integer_Text_IO.Put(Item,0);
   end Put_Int;

   procedure Put_Stack is new IStacks.For_Each(Put_Int);

   S: IStacks.Stack(100);

begin
   IStacks.Push(S,3);
   IStacks.Push(S,66);
   IStacks.Push(S,91);
   Put_Stack(S);
end Stack_Demo;
```

Iterátor

- Adatszerkezet bejárása (ciklussal)
- Minden elemmel csinálni kell valamit
- Külső iterátor
  - A ciklus a klienskódban van
  - Az adatszerkezet-objektumtól független iterátorobjektum
- Belső iterátor
  - A ciklus az adatszerkezet implementációján belül van
  - Az elemeken végzendő művelettel paraméterezett művelet
  - Stacks.For_Each

```
with Stacks;
procedure Stack_Demo is
   package IStacks is new Stacks(Integer); use IStacks;
   function Multiplicity( S: Stack; Pattern: Integer )
     return Natural
   is
      Counter: Natural := 0;
      procedure Compare( Item: in Integer ) is
      begin
         if Item = Pattern then
            Counter := Counter + 1;
         end if;
      end Compare;
      procedure Compare_All is new For_Each(Compare);
   begin
      Compare_All(S);
      return Counter;
   end Multiplicity;
begin ... end Stack_Demo;
```

Rekurzív alprogramsablon

```
function Fold_R ( V: Vector ) return Element is
begin
    if V'Length = 0 then return Start;
    elsif V'Length = 1 then return V(V'First);
    else return Op( V(V'First),
                    Fold_R( V(Index'Succ(V'First) .. V'Last) )
                  );
    end if;
end Fold_R;
```

- Folding from the right

„Egyke”

```
generic
     type Element is private;
     Max: in Positive := 1024;
package Stack is
     procedure Push( X: in Element );
     procedure Pop( X: out Element );
     function Top return Element;
     function Is_Empty return Boolean;
    …
end Stack;
```

Lehet több is belőle

```
with Stack;
procedure Egyke_Demo is
    package V1 is new Stack(Integer, 10);
    package V2 is new Stack(Float);
    package V3 is new Stack(Float);
begin
    V1.Push(3);   V2.Push(3.0);   V3.Push(V2.Top);
end Egyke_Demo;
```

- Fordítási időben meghatározott számú verem létrehozására

Függelék: típusosztályok jelölése

- Diszkrét típus — type T is (<>);
- Előjeles egész — type T is range <>;
- Moduló típus — type T is mod <>;
- Lebegőpontos — type T is digits <>;
- Fixpontos — type T is delta <>;
- Decimális fixpontos — type T is delta <> digits <>;
- Korlátozott, indefinit, *jelölt, osztályszintű, absztrakt, nem dinamikusra mutató*
- Definit tömb, indefinit tömb, *álneves elemű tömb, diszkriminánsos, mutató, konstansra mutató, származtatott, kiterjesztett*