# Programozási nyelvek I.
# (Az Ada)

**Kozsik Tamás**
**kto@elte.hu**
**http://kto.web.elte.hu/**

# A tárgy célja

- A programozási nyelvek alapfogalmainak bemutatása
  - Az előadásokon
  - Számonkérjük zh-kal és a Szoftverszigorlaton
- Az Ada nyelven keresztül
  - Az előadásokon és a gyakorlatokon
  - Számonkérjük zh-kal és a Szoftverszigorlaton
- És egy kis programozási gyakorlatszerzés…

# Miért az Ada?

- Gazdag, sokszínű
- Logikus felépítésű, világos fogalmakkal
- Jó szemléletet ad
- Sok mindenben különbözik a C++ nyelvtől
- Példát ad a Pascal jellegű nyelvekre

- Nehéz, kevesen használják
  - de pl. a PL/SQL nagyon hasonló

# Előfeltétel

- Bevezetés a programozáshoz II.
  (volt: Programozási módszertan I. )
- Programozási környezet

- Ajánlott előzmény (nem előfeltétel!):
  Programozási nyelvek II. (C++)

# Követelmények (1)

- Két géptermi zh, plusz egy pótzh
  - Új: számonkérünk elméletet és gyakorlatot is!
  - A zh-k 2 órásak, és 2 jegyet adunk rájuk
  - Az időpontok a tárgy honlapján lesznek meghírdetve
- Négy beadandó program
  - Mindkét zh előtt két-két beadandó

# Követelmények (2)

- Előadás: 0 kredit, kétfokozatú
  - feltétel: minden zh sikeres
- Gyakorlat: 4 kredit, gyakjegy
  - feltétel: minden zh sikeres
  - feltétel: a beadandók elkészültek
  - átlag: négy zh-jegyből

# Tananyag

- Nyékyné Gaizler Judit (szerk.) és mások:
  Az Ada95 programozási nyelv,
  1999. Eötvös Kiadó

- A tárgy honlapja:
  http://aszt.inf.elte.hu/~kto/teaching/pny1/

# 1. előadás

## Az Ada kialakulása és rövid jellemzése. Alaptípusok, deklarációk és utasítások.

# Egy jó programozási nyelv

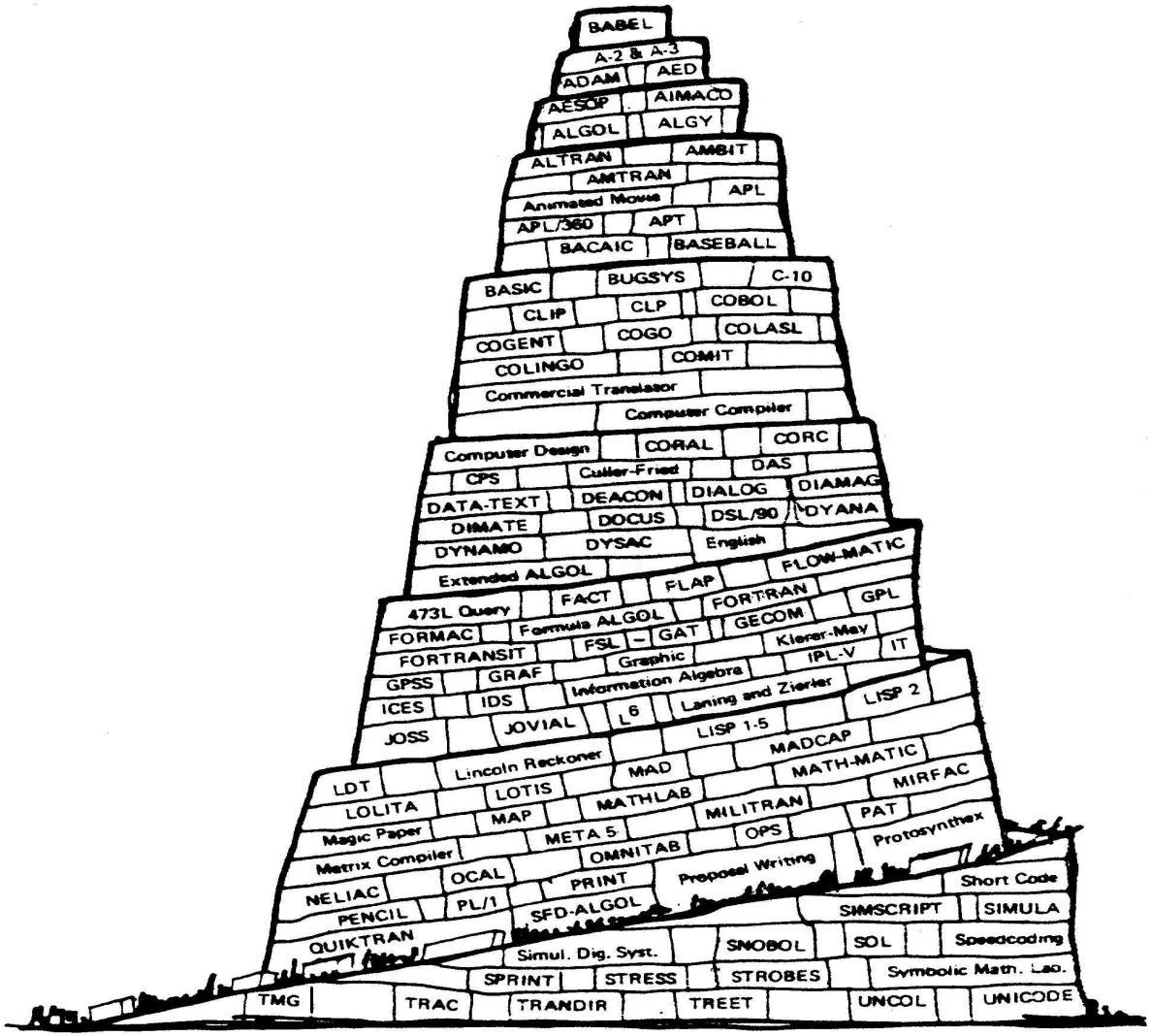
- Egyszerre vonatkoztat el a számítógéptől és az alkalmazásoktól
- Jó jelölésrendszert biztosít az algoritmusok megadásához
- Eszköz a programok bonyolultságának kezelésére

# Az Ada kialakulásának történeti háttere

- Hatalmas nemzetközi erőfeszítés
- Az 1970-es évek elején a US DoD (United States Department of Defense) megszervezte a HOL (High Order Language) elnevezésű munkát.

# A kiindulás

- a követelmények rögzítése és
- a létező nyelvek értékelése volt
  - A FORTRAN, COBOL, PL/I, HAL/S, TACPOL, CMS-2, CS-4, SPL/1, J3B, Algol 60, Algol 68, CORAL 66, Pascal, SIMULA 67, LIS, LTR, RTL/2, EUCLID, PDL2, PEARL, MORAL és EL-1




Tower of Babel

Sammet, J.E. : Programming Languages, 1969

(Bild nach Pieter Brueghel, 1563)

# Az eredmény

- Nincs olyan létező nyelv, ami egymagában megfelelne a célnak.
- Egyetlen nyelvet kell kialakítani.
- A követelmények definiálásának meg kell maradnia a „state-of-the-art” szintjén.
- Az új nyelv fejlesztését a meglevő értékek felhasználásával kell végezni.

# Az értékelés

- *Nem használható fel:* az elavult, vagy elhibázott nyelvek. (pl. FORTRAN, CORAL 66 )
- *Megfelelő:* közvetlenül nem alkalmas, de néhány elemük felhasználható. (pl. az RTL/2 és a LIS)
- *Alapnyelvként használható:*
  a Pascal, a PL/I és az ALGOL 68.

# A verseny

- A CII Honeywell Bull franciaországi laboratóriuma (a „Zöld”).
- Alapnyelv: a Pascal.

# Az új nyelv neve: **Ada**

- Ada Augusta Byron
  - Lord Byron lánya
  - Babbage asszisztense
  - A világon az első programozó

# Az Ada tervezésének szempontjai (1)

- Biztonságra törekvés: megbízható, könnyen karbantartható programok kellenek.
- Ezért a nyelv:
  - erősen típusos,
  - lehetőséget ad az adatabsztrakció kezelésére,
  - külön fordítható programegységekből áll,
  - eszközöket ad a kivételes és hibás helyzetek kezelésére.

# Az Ada tervezésének szempontjai (2)

- Ne legyen túl bonyolult, és legyen a rendszere következetes.
- Hatékony legyen.

# Ada83: az 1983-as szabvány

- Alapja a Pascal
- Néhány vonás: Euclid, Lis, Mesa, Modula, Sue, Algol 68, Simula 67, Alphard, CLU

# Utána további igények: **Ada 95**

- Az interfészek problémája: más nyelven írt rendszerekhez  illesztés
- Újrafelhasználhatóság: objektum-orientált programozás támogatása
- Rugalmasabb, hierarchikus könyvtárszerkezet
- A párhuzamos programozást támogató eszközök továbbfejlesztése

# Az Ada rövid jellemzése

- Egy Ada program egy vagy több programegységből áll
- A programegységek többnyire külön fordíthatóak (könyvtári egység)
- A programegységek egymásba ágyazhatók (blokkszerkezet)

# Az Ada programegységei (1)

- **az alprogramok:** végrehajtható algoritmusokat definiálnak, két fajtájuk van:
  - **az eljárás:** egy tevékenység-sorozat leírása, lehetnek paraméterei
  - **a függvény:** egy érték kiszámítása és visszaadása
- **a csomagok:** logikailag kapcsolatban álló entitások (alprogramok, típusok, konstansok és változók) gyűjteményeit definiálják

# Az Ada programegységei (2)

- **a sablon (generic) egységek:** típussal és alprogrammal is paraméterezhető, makrószerű csomagot vagy alprogramot jelentenek
- **a taszkok:** párhuzamosan végrehajtható számításokat definiálnak.
- **a védett egységek:** feladatuk a taszkok között megosztott adatok védett használatának koordinálása.

# Főprogram

- Egy paraméter nélküli eljárás
- Hivatkozás használt könyvtárakra

```
with Text_IO;
procedure Hello is
begin
   Text_IO.Put_Line("Hello");
end Hello;
```

# Könyvtárak

- Szabványos könyvtár (pl. Text_IO)
- A nyelv ad eszközöket a saját könyvtárak szervezésére.
- Minden könyvtár hierarchikusan van felépítve
  - individuális komponensekre dekomponálhatjuk a rendszert
- Egy önállóan fordított programegység meg kell nevezze a szükséges könyvtári komponenseket.

# A programegység részei

- **A specifikáció** tartalmazza azt az információt, ami más egységek felé látható kell legyen
- **A törzs** tartalmazza az implementációs részleteket -- ez **rejtett** más programegységek felé.

# A programegység részei: példa

```
with Text_IO;

procedure Hello is
begin
    Text_IO.Put_Line("Hello");
end Hello;
```


# A törzs részei

- **deklarációs rész:** definiálja a programegységben használt konstansokat, változókat, típusokat, hibaeseményeket (*exception*) és programegységeket
- **utasítássorozat:** definiálja a programegység végrehajtásának hatását.
- **kivételkezelő rész** (opcionális)

# A törzs részei: példa

```
with Ada.Integer_Text_IO;

procedure Négyzet is
      N: Integer;
begin
      Ada.Integer_Text_IO.Get( N );
      Ada.Integer_Text_IO.Put( N*N );
end Négyzet;
```

deklarációs rész

utasítás-sorozat

# Deklarációs rész

- Elválasztva az utasításoktól
- Változók, konstansok, típusok, kivételek, programegységek
- Változódeklaráció: állapottér megadásához

  azonosító: típus

  N: Natural;
  - Kezdőérték: érdemes, ha értelmes

    B: Boolean := True;
  - Több azonosítóhoz ugyanaz a típus (és esetleg kezdőérték)

    I, J: Integer;

    A, B: Positive := 3;

# Néhány használható típus

- Integer
- Natural
- Positive
- Boolean
- Character
- Float
- String

Beépített típusok

A Standard csomagban vannak deklarálva

# Mire való a típus?

- objektumok deklarációja:

  N: Integer;

  - kezdeti értéket is adhatunk:

    N: Integer := 42;

  - konstanst is deklarálhatunk:

    Max: constant Integer := 100;

    N_Vessző: constant Integer := N;

- fordítási hiba:

  I: Integer; B: Boolean; ... I := B;

# Logikai típus (Boolean)

- A *Standard* csomagban (automatikusan használható a programokban)
- Az *if* és a *while* ezt igényli
- Ilyet adnak: = /= < > <= >=
- predefinit operátorok:
  *not and or xor and then or else*

*if B = True then* *if B then*

*if B = False then* *if not B then*

# „Rövidzár” logikai operátorok

- Lusta kiértékelés: and then or else
  ***ha az első argumentumból meghatározható a kifejezés értéke, akkor megáll***

  ***if A>B and then F(A,B) then***
- Mohó kiértékelés: and or
  ***mindenféleképpen kiértékeli mindkét argumentumot***

  más eredmény: ha a második argumentum
  - nem mindig értelmes / kiszámítható (futási hiba)
  - mellékhatással rendelkezik

# Például

- Futási hiba - lineáris keresés tömbben

  while I <= N and then T(I) > 0 loop

- Mellékhatás - álvéletlenszám kérése

  if EOF or else Random(G) > 0.5 then

# Az Integer típus

- **Az egész számok halmaza:**
  **..., -3, -2, -1, 0, 1, 2, 3, ...**

- ***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*

# A mod és a rem különbsége

| A | B | A/B | A rem B | A mod B |
|---|---|---|---|---|
| 12 | 5 | 2 | 2 | 2 |
| -12 | 5 | -2 | -2 | 3 |
| 12 | -5 | -2 | 2 | -3 |
| -12 | -5 | 2 | -2 | -2 |

A=(A/B)*B +(A rem B) előjel A-é

A=B*N +(A mod B) előjel B-é (N egész)

# A Float típus

- ***Predefinit műveletek:***

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

  - ***hatványozás:*** **Y *egész***

# Kevert aritmetika

- Nem megengedett:

  ***I: Integer := 3;***

  ***F: Float := 3.14;***

  ***I := F + 1;*** ***F := I - 1.3;*** ***-- Hibásak!***

- Explicit konverzióra van szükség:

  ***I := Integer(F) + 1;*** ***F := Float(I) - 1.3;***

  - Integer(F) kerekít

| | | | |
|---|---|---|---|
| 1.4 | 1 | 1.6 | 2 |
| 1.5 | 2 | -1.5 | -2 |

# Precedenciák

- Pontosan definiálva van az operátorok precedenciája
  - később visszatérünk rá...
- Bal-asszociativitás
- Zárójelezés

# Utasítások

- Egyszerű utasítások
  - Értékadás
  - Üres utasítás
  - Alprogramhívás és return utasítás
- Összetett utasítások
  - Elágazások
  - Ciklusok
  - Blokk utasítás

# Az értékadás

- Egy kifejezés értékét egy változóhoz rendeljük

  I := 5;

- Az érték és a változó típusának meg kell egyeznie

  I := True;  -- fordítási hiba!

- Futási időben további ellenőrzés (altípus)
- A korlátozott típusú változóktól eltekintve minden változóra alkalmazható (sőt, „balértékre”)
- Az értékadás nem kifejezés, a := pedig nem operátor (nem definiálható felül, viszont van Controlled)
- Nincs szimultán értékadás

# Az üres utasítás

```
procedure Semmi is
begin
   null;
end;
```

- Üres begin-end nem írható
- Más szituációkban is használjuk
- Ezt a kulcsszót nem csak utasításként használjuk

# Alprogramok hívása és a return

- Alprogramokat nevük és aktuális paramétereik megadásával hívhatunk.

  Text_IO.Put_Line("Kiírandó szöveg");

  Text_IO.Get(Ch);

  -- ahol Ch egy karakter típusú változó

  Text_IO.New_Line;

- A return utasítás hatására az alprogram végrehajtása befejeződik.
- Függvénynél itt adjuk meg az eredményét is:

  **return** X+Y;

# Összetett utasítások

programkonstrukciók kódolására

- Elágazás: if, case
- Ciklus: léptetős, tesztelős, feltétel nélküli

- Blokk utasítás

# Elágazás: if

```
if  A>B  then
    Temp := A;
    A := B;
    B := Temp;
end if;
```

# Elágazás: if + else

```
if  A>B  then
    A := A - B;
else
    B := B - A;
end if;
```

# Elágazás: if + elsif (+ else)

```
if  Aktuális < Keresett  then
    U := I + 1;
elsif  Aktuális > Keresett  then
    V := I - 1;
else
    Megtaláltuk := True;
end if;
```

- *akárhány ág lehet*
- *elhagyható az else*

# Csellengő else

```
if (a>0)
        if (b>0)
                c = 1;
else
        c = 0;
```

- Veszély: C++, Java, Pascal stb.
- Az Adában ilyen nincs
- Az if utasítás le van zárva

# Elágazás: case (1)

```
Ch : Character;
```

***diszkrét típusú értéken***

```
case Ch is
      when ‘d’ => X := X+1;
      when ‘w’ => Y := Y-1;
      when ‘a’ => X := X-1;
      when ‘z’ => Y := Y+1;
      when others => null;
 end case;
```

# case - switch

- A case (Ada) a természetesebb, érthetőbb
- A switch utasítás (C++) általában break-kel jár
- Az „átcsorgásnak” vannak jó felhasználási területei
- Ezek gyakran megoldhatók case utasítással is

# Elágazás: case (2)

```
case  (X mod 20) + 1  is
        when 1..3 | 5 | 7 | 11 | 13 | 17 | 19 =>
                Felbonthatatlan := True;
        when others =>
                Felbonthatatlan := False;
end case;
```

# Ciklus: léptetős

```
with  Ada.Integer_Text_IO;
procedure Tízig is
begin
    for I in 1..10 loop
     Ada.Integer_Text_IO.Put( I );
    end loop;
end Tízig;
```

**A ciklusváltozó:**

- **csak a ciklusban létezik**
- **nem külön deklaráljuk**
- **nem változtatható az értéke a ciklusmagban**
- **„a ciklusmag lokális konstansa”**

- **csak egy intervallumot adhatunk meg neki**
- **csak egyesével léphet**

# Ciklus: léptetős

- Üres intervallum: ciklusmag nem hajtódik végre

- Visszafelé haladó ciklus:

```
for I in reverse 1..10 loop
      Ada.Integer_Text_IO.Put( I );
end loop;
```

# Ciklus: tesztelős (csak elöl)

```
with Ada.Integer_Text_IO;
procedure Tízig is
        I: Positive := 1;
begin
        while  I <=10  loop
                Ada.Integer_Text_IO.Put( I );
                I := I + 1;
        end loop;
end Tízig;
```

# Ciklus: feltétel nélküli

- Ritkábban használjuk, egyelőre nem is olyan fontos
- Végtelen ciklus írására (párhuzamos programok)
- Kilépés itt is lehetséges, sőt, feltételes kilépés is
  - exit utasítás

```
loop
    ...
end loop;
```

# A blokk utasítás

- Utasítások között egy blokk nyitható
- Tartalmazhat újabb deklarációkat

<utasítások>
**declare**
    <deklarációk>
**begin**
    <utasítások>
**end;**
<utasítások>

# A blokk utasítás: beágyazás

- Akármilyen mélységig mehet: *hierarchia*

```
…
declare
    …
begin
    …
    declare … begin … end;
    …
end;
…
```

# A blokk utasítás: Mikor van rá szükség?

- nagy adatstruktúrára van szükség a program egy rövid szakaszán
  - részben kiváltható mutatókkal
- egy deklarálandó objektum mérete futási időben dől el, deklarálás előtt ki kell számolni
  - részben kiváltható mutatókkal
- kivételkezelő kód helyezhető el a végén
  - később visszatérünk erre...

# Strukturálatlan utasítások: az exit utasítás (1)

- Ciklusból való kiugrásra használható
- Ciklusmagon belül bárhova írható
- Gyakran feltétel nélküli ciklushoz használjuk

```
loop
        …
        exit;
        …
end loop;
```

# Strukturálatlan utasítások: az exit utasítás (2)

- Feltételhez is köthető a kiugrás
- Így kódolhatunk például hátultesztelő ciklust is

```
loop
    Get(Ch);
    exit when Ch = ‘q’;
    …
end loop;
```

```
loop
    …
    exit when  <felt.>
end loop;
```

# Strukturálatlan utasítások: az exit utasítás (3)

- Egymásba ágyazott ciklusoknál is jól jön
- Névvel jelölt ciklusok segítségével…

```
A: for I in 1..10 loop
        for J in 1..10 loop
            if  Beta(I,J) then
                …
                exit A;
            end if;
        end loop;
end loop A;
```

exit A when <felt.>

# Strukturálatlan utasítások: a goto utasítás

- valójában nincs rá szükség!
- ha nagyon optimalizálni kell...

```
<<COMPARE>>
if A(I) < Elem then
   if Balszomszéd(I) /= 0 then
      I := Balszomszéd(I);
      goto COMPARE;
   end if;
   …  -- utasítások
end if;
```