



Programozási nyelvek és módszerek

3. ELŐADÁS – VEZÉRLÉSI SZERKEZETEK,
ALPROGRAMOK

Típusok folytatás

Pointer és referencia típusok

Egy pointer (és egy referencia) egy olyan objektum, amely megadja egy másik objektum címét a memóriában

Egy pointer értéke egy memóriacím

- gépi nyelvekben az indirekt címezés lehetősége motiválta a pointerek létrehozását.

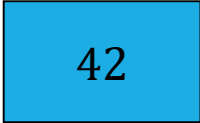
Vannak **típusos** és **típus nélküli** pointerek.

Referencia szint

Pointerek segítségével magasabb referencia-szinten hivatkozhatunk objektumainkra.

42 ← A 42 szám referencia szintje: 0

x  ← egy olyan változó referencia-szintje, amely tartalmazza a 42 értéket: 1

xp  →  xp ← a pointer referencia-szintje, amely erre a változóra mutat 2

Mire kellenek a mutatók?

Hatékonyság

- ahelyett, hogy nagy adatszerkezeteket mozgatnánk a memóriában, sokkal hatékonyabb, ha az erre mutató pointert másoljuk, mozgatjuk.

x

23	10	11	17	0	34	28	22	55	88
4	7	0	0	6	8	1	9	10	3

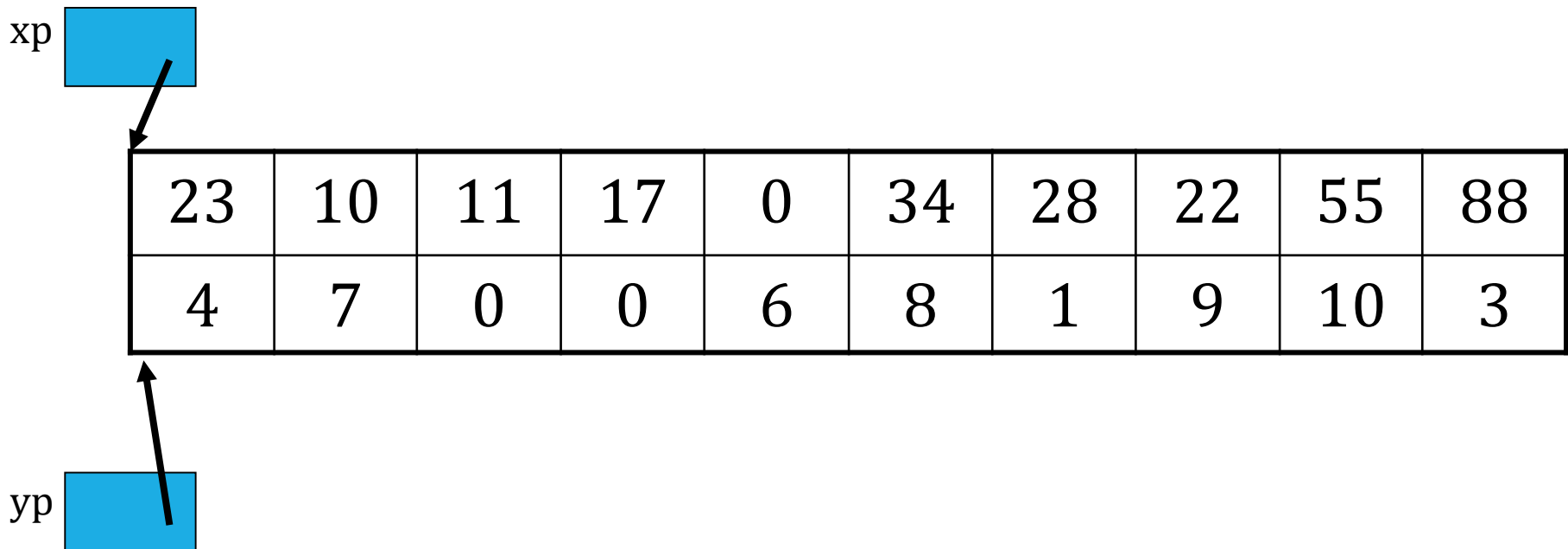
y

23	10	11	17	0	34	28	22	55	88
4	7	0	0	6	8	1	9	10	3

Mire kellenek a mutatók?

Hatékonyság

- ahelyett, hogy nagy adatszerkezeteket mozgatnánk a memóriában, sokkal hatékonyabb, ha az erre mutató pointert másoljuk, mozgatjuk.
- itt vigyázni kell az osztott használatra! – jó, ha van read-only elérési lehetőség is



Mire kellenek a mutatók?

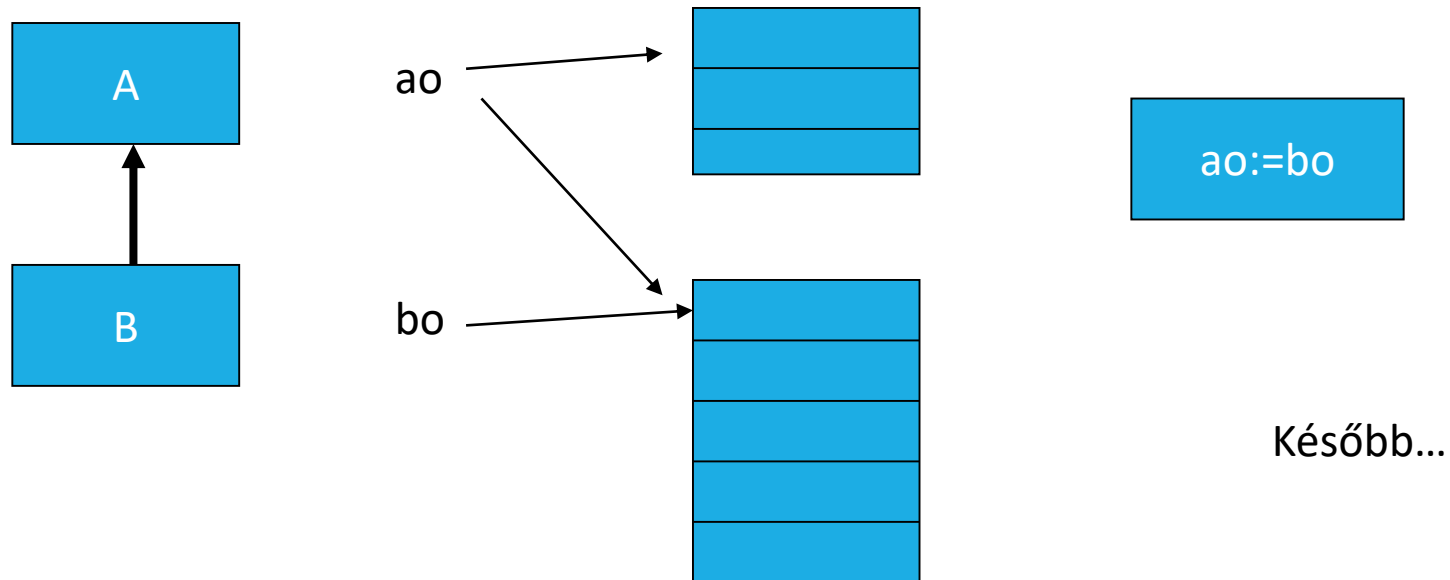
Dinamikus adatszerkezetek építéséhez



Mire kellenek a mutatók/referenciák?

Objektumorientált funkciókhoz

- a programozási nyelvekben a polimorfizmust akkor tudjuk támogatni, ha a változók objektumokra való referenciákat tartalmaznak.



A szokásos műveletek

értékadás - pointerek között, hivatkozott objektumok között (copy, clone, deep-copy, deep-clone!)

egyenlőség vizsgálat - ha két ugyanolyan típusú pointer ugyanarra az adatszerkezetre mutat (ez is több szinten lehet)

dereferencing - a *mutatott* objektum részére vagy egészére való hivatkozás

referencing - egy objektum címe

új objektum dinamikus **allokálása**

egy objektum **deallokálása** - explicit művelettel vagy implicit módon egy garbage collector-ral

néha (pl. C, C++) **összeadás**, **kivonás** is megengedett

„Csellengő” pointerek

„Csellengő” pointer: kísérlet olyan változó elérésére, ami már nem létezik.

```
#include <iostream>
int *r;
double *r2;
void f(){int v; r=&v;}
void g(){double v; v=2.1; r2=&v;}
```

Compiler, builder:

0 error(s), 0 warning(s)

Az eredmény megjósolhatatlan!

```
int main(){
    f(); g(); *r=3; *r2=1.2;
    cout << "dangling *r="<<*r;
    cout << "\n *r2 " <<*r2 << ".\n";
}
```

„Csellengő” pointerek 2.:

```
int main(){  
    int *j,*i;  
    double *d;  
    j=new int;  
    *j=3;  
    i=j;  
    delete j;  
    d=new double;  
    *d=4.2;  
    cout << *i;  
}
```

Compiler, builder:

0 error(s), 0 warning(s)

Az eredmény megjósolhatatlan!

Pointerek az egyes nyelvekben

A programozási nyelvek között a lehetséges különbségek:

- Csak konkrét típusra mutató pointerek megengedettek, vagy vannak típus nélküli pointerek is?
- Csak dinamikusan allokált objektumokra mutathat pointer, vagy "normál" változókra is?
- Lehetnek-e alprogramra mutató pointerek is?
- Milyen fajta konstans pointerek megengedettek?
 - Egy tömbnév C-ben egy konstans pointer a tömb objektum 0. elemére.
- Kötelező a pointer típusoknak önálló nevet adni, vagy csak a mutatott típust kell megadni?
 - `type PInteger is access to Integer;` (ADA-ban)
 - `int * x;` (C-ben)

Pointerek az egyes nyelvekben

Milyen biztonságosan kezelhető a “csellengő” pointerek problémája?

Mi a megengedett műveletek halmaza?

Kapnak a pointer változók kezdeti (üres) értéket a deklarációnál?

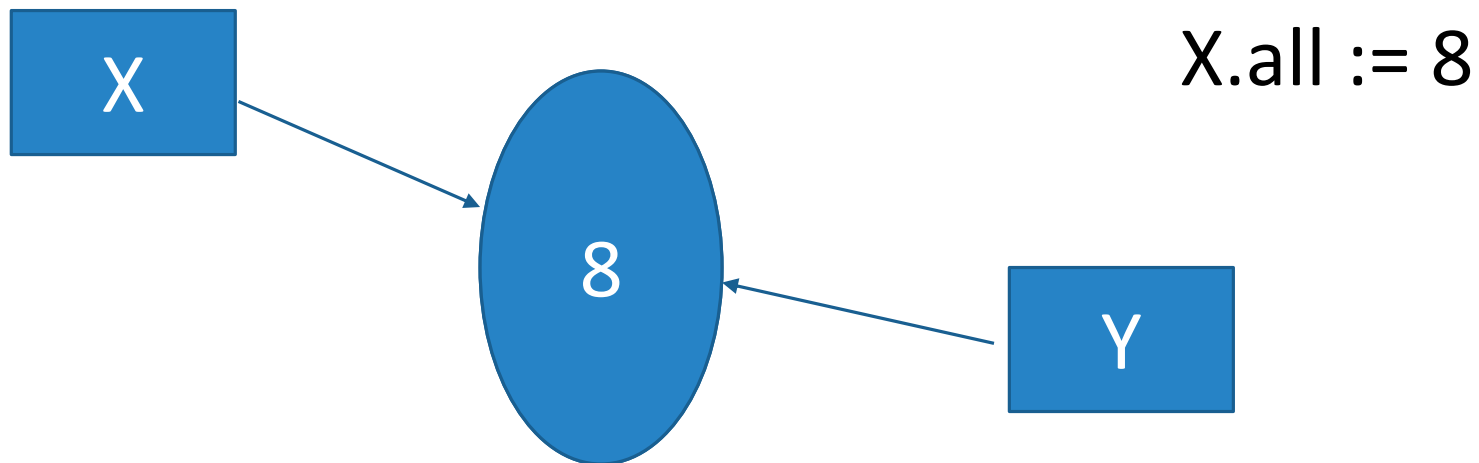
Lehetséges-e ugyanazt az adatot két (vagy több) pointeren keresztül is változtatni/elérni?

Pointerek

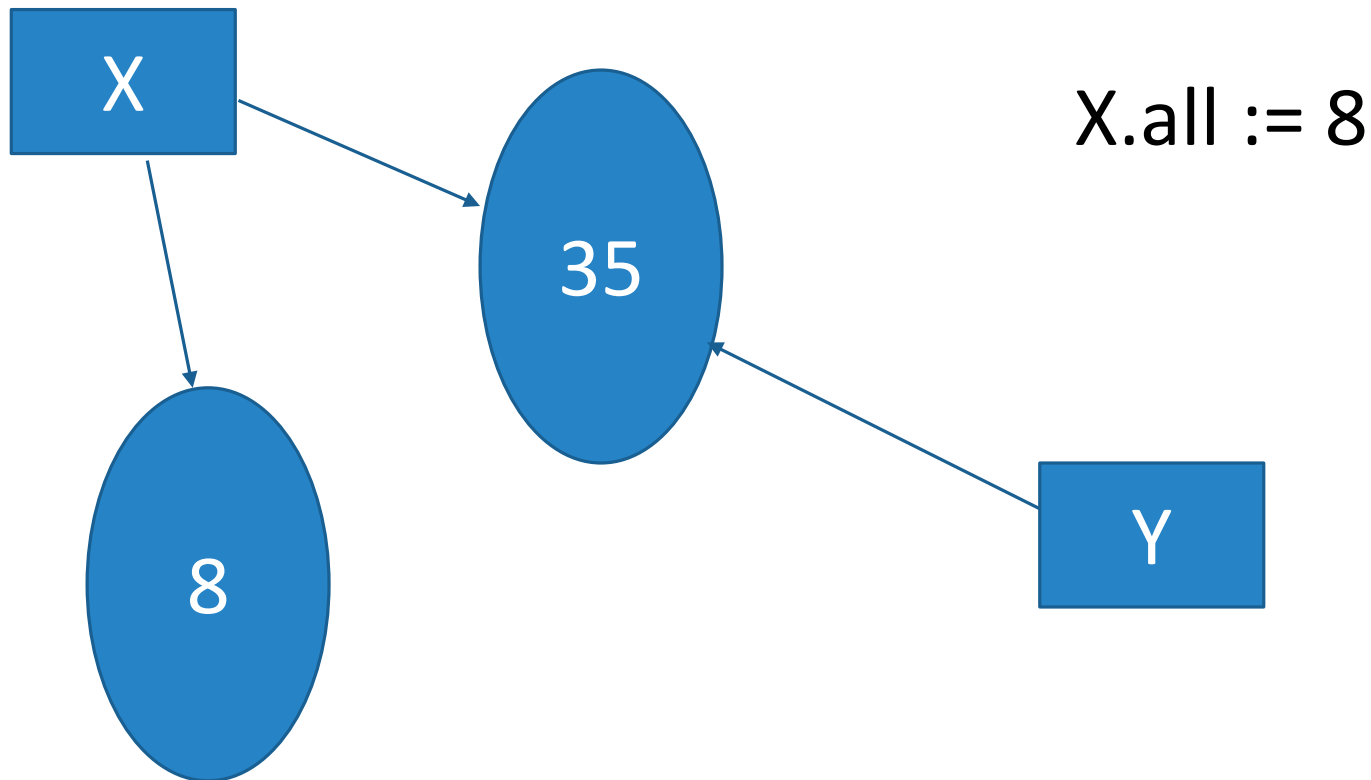
CLU

- A CLU-ban nincs hagyományos pointer típus.
- A program végrehajt műveleteket objektumokon
- Az objektumok mint egy univerzum részei léteznek, a program változói hivatkoznak ezekre az objektumokra
 - Garbage collection a felszabadításra
- A programban kétféle objektum lehet:
 - mindig ugyanaz az értéke (immutable)
 - változhat az értéke (mutable)

Mutable – változtatható



Immutable – nem megváltoztatható



Pointerek

C++

- A legtöbb T típusra, T^* a megfelelő pointer típus:
 - `int *p;`
- A tömbökre és függvényekre mutató pointereknek kicsit bonyolultabb jelölése van
 - `int (*vp)[10];` // pointer 10 int tömbjére
 - `int (*fp) (char, char*);`
// függvényre mutató pointer, amelynek
// (char, char*) argumentumai vannak és egy egész (int) ad vissza

Pointerek

C++

- A megengedett műveletek:
 - Dereferencing: prefix *
 - Dinamikus allokálás new
 - Deallokálás delete
 - értékadás: =
 - egyenlőség: ==
 - additív műveletek + -
 - increment, decrement ++ --
 - member ref. .* ->*

Pointerek

C++

- van egy speciális operátor, az "address_of" '&', ennek segítségével adhatjuk értékül változók címét pointereknek:
 - `int i = 10;`
 - `int *pi = &i; // a pi pointer az i változóra vonatkozik`
 - `int j = *pi; // j-t 10-re állítjuk`
- Az '&' operátorral létrehozhatjuk objektumok referenciáit is
 - egy referencia úgy tekinthető mint egy konstans pointer, ami mindig automatikusan dereferenciát hajt végre:
 - `int &r = i; // r és i ugyanarra vonatkozik`
 - `r = 2; // i=2`
- Az increment, decrement, ... veszélyesek lehetnek, vigyázzunk, ne keverjük össze a jelentését!
 - `pi++` a pointert inkrementálja, és a következő memóriacímre fog mutatni, ennek akkor van értelme, ha pi egy tömbre mutat, míg `r++` inkrementálja i értékét

Pointerek

Java

- Nincs hagyományos pointer típus
- A változóban kétféle érték tárolható
 - primitív értékek (egy numerikus típusból vagy egy logikai)
 - referencia értékek
- Az objektumokat (osztályok példányai vagy tömbök) referenciákkal kezeli.
- Ugyanarra az objektumra számos referencia hivatkozhat.
- Objektumok referenciáinak műveletei:
 - mező elérés, metódus hívás, casting, string concatenation, instanceof, '==' '!=' (referencia egyenlőségének vizsgálata) stb.

Pointerek

Eiffel

- Itt sincsenek hagyományos pointer típusok.
- A változóknak kétféle érték tárolható - kiterjesztett értékek és referencia értékek.
- Ugyanarra az objektumra számos referencia hivatkozhat.

Pointerek

C#

- A referencia típusok objektumait kezelhetjük referenciákkal.
- Egy „unsafe” környezetben egy típus lehet pointer is, erre számos művelet megengedett (pl. a ++, -- is).

Hogyan definiálhatunk új adattípusokat?

Példa

- Pascal

- `type <typn>=<value desc>;`
- `type myint=integer;`

- C++

- `typedef <value desc> <typn>;`
- `typedef int myint;`

- ADA

- `subtype <typn> is <typ1>;`
- `subtype Int is Integer;`

`type <typn> is new <typ1c>;`
`type My_Int is new Integer;`

- CLU

- `cluster ...`

- Java, Eiffel, C#

- `class ...`

Melyek a megengedett típus-konstrukciók?

Iterált

- egy kiinduló típusból

Direkt szorzat

- több kiinduló típusból

Unió

- több kiinduló típusból

Tömb típusok

„Egy tömb egy olyan adatszerkezet, amely azonos típusú elemek sorozatait tartalmazza.”

Általában egy tömb egy leképezés egy folytonos diszkrét intervallumról elemek egy halmazára, nem igazi sorozat típus.

- Tömbnév(indexértékek) $\rightarrow$ elem

A diszkrét intervallum elemeit hívjuk index értékeknek.

Az elemek száma ebben az intervallumban definiálja a tömb *méretét*.

A legfontosabb kérdések

- Milyen adattípusok lehetnek tömb típusok indextípusai?
- Mi lehet tömb típusok elemtípusa?
- Tartalmazzák-e a tömb típusok az indexhatárokat? És a tömb objektumok?
- Mikor dől el a mérete, a helyfoglalása?
- Van-e indextúlcsoordulás-ellenőrzés?
- Van-e többdimenziós tömb?
 - Van-e altömb (szelet) képzés?
 - Van-e teljes tömbre vonatkozó értékadás? (kezdő értékadás?)
 - Van-e tömbkonstans?
- Megváltoztatható-e egy tömb mérete?
- Rögzített méretű sorozat vagy nem?

Indexelés

Az alapművelet az *indexelés*

- $A[i]$
- Elvárás, hogy az A tömb i . elemét gyorsan el kell tudni érni!

Vannak programozási nyelvek, ahol az elemek különböző típusúak is lehetnek (SmallTalk) de általában az elemek ugyanahhoz a típushoz tartoznak, vagy egy adott típus lehetséges leszármazottai is lehetnek.

Tömbök

ADA

- Tömb típusok definiálhatók rögzített és megszorítás nélküli indexhatárokkal:
 - `type A is array(Integer range 2 .. 10) of Boolean;`
 - `type Vect is array(Integer range <>) of Integer;`
 - `type Matr is array(Integer range <>, Integer range <>) of Integer;`
- A konkrét indexhatárokat az adott objektum deklarációjánál kell meghatározni:
 - `V : Vect(1 .. 30);`
 - `B : Matr(1 .. 2, 1 .. 4);`
- Gyakran használják alprogramok paramétereiként, sablonoknál.

Tömbök

ADA

- Az index típusa tetszőleges diszkrét típus lehet, az elemek típusa tetszőleges típus
- A definíciókat futási időben értékeli ki, az aktuális indexhatárok nem kell, hogy statikusak legyenek.
- Tömbök szeletei is létrehozhatók:
 - $V(2..12)$
 - $de: V(1..1) \leftarrow V(1)!$
- Megengedett az értékadás azonos típusú tömbök között:
 - $V(1..5) := V(2..6);$
- Lehetséges értékadás tömb „aggregátokkal” is:
 - $V := (1..3 \Rightarrow 1, \text{others} \Rightarrow 0);$

Tömbök

ADA

- Három előredefiniált string típus:
 - `type String is array (Positive range <>) of Character;`
 - `type Wide_String is array (Positive range <>) of Wide_Character;`
 - `type Wide_Wide_String is array (Positive range <>) of Wide_Wide_Character;`
- Vannak speciális attribútumai:
 - `A'First/A'First(N)`
 - `A'Last/A'Last(N)`
 - `A'Range/A'Range(N)`
 - `A'Length/A'Length(N)`

Tömbök

generic

type Elem is private;

type Index is (<>);

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

procedure Glinker (V: Vekt; E: Elem;
 Found: out Boolean; Ind: out Index);

procedure Glinker (V: Vekt; E: Elem; Found: out Boolean;
 Ind: out Index) is

begin

Ind:=V'First;

Found :=V(V'First)=E;

while not Found and Ind<V'Last loop

 Ind:=Index'Succ(Ind);

 Found:=V(Ind)=E;

end loop;

end;

Tömbök

C++:

- Egy T típusra:
 - `T x[size]` a T típusú elemek `size` méretű tömbje. Az indexek 0 és `size-1` között.
 - `float v[3];` // 3 float tömbje
 - `int a[2][5];` // 5 int két tömbje
- Kezdeti érték adható:
 - `char v[2][5] = {{ 'a', 'b', 'c', 'd', 'e' },
 { '0', '1', '2', '3', '4' } };`
- A pointerek és tömbök szoros kapcsolatban vannak
 - egy tömbnév mindig a tömb nulladik elemére hivatkozik, így használható a pointeraritmetika tömbökre.
- Nem tudja a méretét!
- STL vector template osztály!

Tömbök

Java:

- "Java arrays are objects, are dynamically created and may be assigned to variables of type Object."
- `int [] ai; // egészek tömbje`
- `short [][] as1,as2; // as1 és as2 short-ok tömbjének tömbjei`
- ha a '[' -t a változó neve után írjuk, akkor csak ez a változó lesz tömb:
 - `long l, al[]; // l egy long típusú változó,
// al long típusú elemek tömbje`
- Tömb létrehozása:
 - `a = new int[20];`
- A tömb mérete lekérdezhető: `a.length`
- Kezdeti érték adható:
 - `String []colours ={"red","white","green"};`

Tömbök

Java

- Egy tömbök tömbjében (ami a Java esetén gyakran több dimenziós tömbbnek hívnak a dokumentációkban) az elemeknek lehet különböző mérete:

```
int[][] m = new int[3][];  
for (int i=0; i<m.length; i++ ) {  
    m[i] = new int[i+1];  
    for (int j=0; j<m[i].length; j++)  
        m[i][j] = 0;  
}
```

- A szövegek kezelését a `String` és `StringBuffer` (`StringBuilder`) osztályokkal oldották meg.
 - Karakterek egy tömbje nem `String`!

Tömbök

C#

- A tömb elemeinek számozása 0-val kezdődik
- Kétféleképpen lehet deklarálni
 - adott hosszúságú vagy dinamikus.
- A nyelvben a tömbök objektumok, a deklaráció után szükség van a tömb példányosítására (new)
- Inicializálásra: { }
- Adott méretű tömb deklarálása:
`int[] Tomb;     Tomb = new int[3];`
- Ugyanez a tömb inicializálva:
`Tomb = new int[3] { 1,2,3 }`
- Dinamikus tömb létrehozása inicializálással:
`Tomb = new int[] { 1,2,3 }`
- A deklarációval egybekötött inicializáció:
`int[] Tomb = new int[3] { 1,2,3 }`
- Ha egy tömböt nem inicializálunk, akkor a tömb elemei automatikusan inicializálódnak az elem típusának alapértelmezett inicializáló értékére.

Tömbök

C#

- A tömbök lehetőségei:
 - egydimenziós tömbök,
 - többdimenziósak vagy négyszögszerűek,
 - kesztyűszerűek (tömbök tömbjei)
 - kevert típusúak (az előzőekből)
- Példa egy kesztyűszerű dinamikus tömbre:
 - ```
int[][] numArray = new int[][] { new int[] {1,3,5},
 new int[] {2,4,6,8,}};
```
- minden tömb típus a `System.Array` bázistípusból „származik”.
  - Az `Array` osztály egy absztrakt bázisosztály, de a `CreateInstance` metódusa létre tud hozni tömböket.
  - Ez biztosítja a műveleteket a tömbök létrehozásához, módosításához, bennük való kereséshez, illetve rendezésükhöz.

# Tömbök

---

## C#

- Az Array osztály tulajdonságait megadó függvények:
  - IsFixedSize - rögzített hosszúságú-e
  - IsReadOnly - írásvédett-e
  - IsSynchronized - a tömb elérése kizárólagos-e (szálbiztos)
  - Length - a tömb elemeinek száma
  - Rank - a tömb dimenzióinak száma
  - SyncRoot - Visszatér egy objektummal, amit a tömb szinkronizált hozzáféréséhez használhatunk
- Az Array osztályban még számos szolgáltatás:
  - BinarySearch - bináris keresés a tömbön
  - Clear - minden elemet töröl a tömbből és az elemszámot 0-ra állítja
  - Clone - másolatot készít
  - Copy - egy tömb részét átmásolja egy másik tömbbe, végrehajtja az esedékes típuskényszerítést és csomagolást (boxing).

# Tömbök

---

## C#

- Az Array osztályban még számos szolgáltatás:
  - CopyTo - átmásolja az elemeket egy egy-dimenziós tömbből egy másik egy-dimenziós tömbbe egy megadott indextől kezdve.
  - CreateInstance - Létrehoz egy tömb példányt.
  - GetEnumerator - Visszatér egy IEnumerator-ra a tömbhöz.
  - GetLength - az elemek száma
  - GetLowerBound - Megadja a tömb alsó korlátját.
  - GetUpperBound - Megadja a tömb felső korlátját.
  - GetValue - megadott indexű elem értéke.
  - IndexOf - egy-dimenziós tömbben az első érték indexe.
  - Initialize - Egy értéktípusú tömbben minden elemre meghívja az elemek alapértelmezett konstruktorát.
  - LastIndexOf - Visszaadja az egy-dimenziós tömbben az utolsó érték indexét.
  - Reverse - Megfordítja a tömb vagy tömbrészet bejárési irányát.
  - SetValue - A megadott elemet beteszi a megadott helyre a tömbben.
  - Sort - A tömbön rendezést hajt végre.

# Tömbök

---

## Eiffel

- Az Eiffel tömbök az `ARRAY[G]` sablon osztály példányai.
- A stringeket a `STRING` osztály objektumai valósítják meg.

# Asszociatív tömbök

---

Egy asszociatív tömb elemek egy rendezetlen halmaza, amelyet megegyező számú, kulcsnak nevezett értékek indexelnek.

Ezeket a kulcsokat is tárolni kell

- az elemek így (kulcs, érték) párok



# Asszociatív tömbök

---

## Perl

- A hash skaláris adatok gyűjteménye, az indexek tetszőleges skalárok.
- Ezek a kulcsok, amiket használunk az elemek elérésére.
- A hash-eknek nincs sorrendjük.
- A hash változók % jellel kezdődnek. A hivatkozás {}-lel történik.

```
%szinek = ('piros' => 0x00f,
 'kék', => 0x0f0,
 'zöld', => 0xf00);
```

- A hash változókra:
  - a keys függvény a kulcsok listáját adja vissza,
  - a values pedig az értékeket.
  - a delete-tel lehet kulcs szerint törölni,
  - az each függvény végigmegy a hash-en visszaadva a kulcs-érték párokat,
  - az exists függvény megadja, hogy egy adott kulcs szerepel-e a hash táblában.

# Asszociatív tömbök

---

A Java, a C++, az Eiffel szabványos osztálykönyvtárában is megtalálhatók

A .NET keretrendszer osztálykönyvtárában is

Egyéb nyelvek:

- PHP,
- Python
- Ruby,
- Lua,
- Pike,
- stb.

# Rekord típusok

---

A direkt szorzat és az unió típuskonstrukciókat számos nyelvben az úgynevezett **rekord** típusok segítségével valósítják meg.

Ha adott két típus,  $S$  és  $T$ , direkt szorzatukat  $S \times T$  jelöli.

- $S \times T = \{(x,y) \mid x \in S; y \in T\}$
- Ez általánosítható több halmazra is:

$$S_1 \times S_2 \times \dots \times S_n$$

A COBOL, Pascal, Ada stb. rekordjait, az Algol68, C, C++, C# stb. struktúráit a direkt szorzat terminusaival érthetjük meg.

# Rekord típusok

---

```
record
<name1> : <type1>;
<name2> : <type2>;
...
<namek> : <typek>;
end
```

A komponenseket a rekord (struktúra) mezőinek hívják.  
Az alapművelet a komponens kiválasztás.

# Rekord típusok

---

Lehet-e paramétere a típusnak?

Van-e kezdő értékadás a mezőkre?

Van-e teljes rekordra vonatkozó értékadás?

Van-e rekord konstans?

Hogyan működik a kiválasztás művelet?

# Rekord típusok

---

## Pascal

- ```
type rektipnev = record
    mnev1 :  tipus1;
    ...
    mnevn : tipusn;
end;
```
- változó deklarálása
 - `rek: rektipnev;`
- hivatkozás ponttal
 - `rek.mnev1`
- csak mezőnkénti értékadás lehetséges

Rekord típusok

Pascal

- Speciális utasítás, aminek a segítségével a rekord mezőire közvetlenül tudunk hivatkozni:

```
type Date= record
  Year  : Integer;
  Month  : 1..12;
  Day    : 1..31;
end;
var R1, R2 : Date;
begin
  R1 := R2; {értékadás megengedett} ...
  with R1 do begin
    Year := 2011; Month:=9; Day:=29;
    Year := R2.Year;
  end;
```

Rekord típusok

C++

```
struct strnev {  
    típus1 mnev1;  
    ...  
    típusn mnevn;  
};
```

- változó deklarálása
 - `strnev x;`
- hivatkozás ponttal
 - `x.mnev1`

Rekord típusok

C++

- A tömbökre használt jelölés alkalmazható struktúrákra is. pl.:

```
struct address{  
    long number;  
    char* street;  
    char* town;  
    int zip;  
}
```

```
address a = {1, "Korkeakoulunkatu", "Tampere", 33720}
```

- Az inicializálásra a konstruktorok jobban használhatóak.
- Értékadás megengedett, de az egyenlőségvizsgálat nem előre definiált.
 - A felhasználó definiálhat operátorokat rá

Rekord típusok

ADA

- `type Complex is record`
 `Re: Float;`
 `Im: Float;`
`end record;`
- változó deklarálása
 - `C: Complex;`
- a komponenseire ponttal hivatkozhatunk
 - `C.Re`
 `C.Im`
- megengedett az értékadás is
 - `C1, C2: Complex;`
 `C1 := C2;`

Rekord típusok

ADA

- A rekord diszkriminánsa(i)
 - a típus paramétere
 - több diszkrimináns is lehet egy típusnak
 - a rekord diszkrimináns diszkrét típusú

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

Rekord típusok

C#:

- a rekord (struct) érték típus
- az osztály (class) referencia típus

Bizonyos programozási nyelvekben nincs rekord típus, a tervezők az osztályok használatát javasolják helyette

- SmallTalk
- Eiffel
- Java
- ...

Uniók és variáns rekordok

Az unió típusértékhalmaza az unió komponensei típusértékalmazának az uniója.

- Pl. bútor:
 - szék vagy asztal vagy szekrény
 - színe, anyaga – van mindegyiknek
 - egyéb speciális jellemzők – külön-külön

A variáns rekordok olyan direktszorzatok, ahol a direktszorzat egy – az utolsó – komponense unió.

„Választó” típusműveletek

A programozási nyelvek a megbízhatóság különböző szintjén támogatják ezt az adatszerkezetet.

Az unió típusnak van egy speciális, tag-nek nevezett komponense, és egy kiválasztási mechanizmusa, ami megadja a tag különböző értékeinek megfelelő alstruktúrákat

Ha a tag-et **tárolja** a rekord, és az alkomponensek elérhetősége ennek aktuális értékétől függ,

- akkor ez egy „megkülönböztetett” (discriminated) unió,
- különben ez egy „szabad” (free) unió.

Unió (Variáns rekord)

Meg lehet-e állapítani, hogy a rekord melyik változat szerint lett kitöltve?

Ki lehet-e olvasni a kitöltéstől eltérő változat szerint?

Szerkezete (gyakran):

```
case <tag-name> : <tag-type> of
  <const1> : ( <fields1>);
  <const2> : ( <fields2>);
  ...
  <constv> : ( <fieldsv>);
```

Unió (Variáns rekord)

A szabad uniók esetében a tag mezők használata opcionális, és a fordító nem ellenőrzi a kiválasztott mező és a tárolt érték konzisztenciáját.

- Ez a nyelv típusrendszerét megbízhatatlanná teszi.

A megkülönböztetett unió esetében fontos kérdés, hogyan lehet új értéket adni a tag mezőnek.

- A tag értéket beállítja a rekord létrehozásakor.
- Míg a program képes kell legyen új értéket adni a rekord "normális" komponenseinek, a tag megváltoztatása a rekord szerkezet megváltozását vonja maga után!

Példa

Pascal:

```
type Listptr = ^Listnode;  
type Listnode = record  
    Next: Listptr;  
    case Tag: Boolean of  
        False: (Data: char);  
        True: (Down: Listptr)  
    end;  
var p, q: Listptr; ...  
p^.Tag := true;  
p^.Down := q;  
p^.Tag := false;  
writeln(p^.Data);
```

HIBA

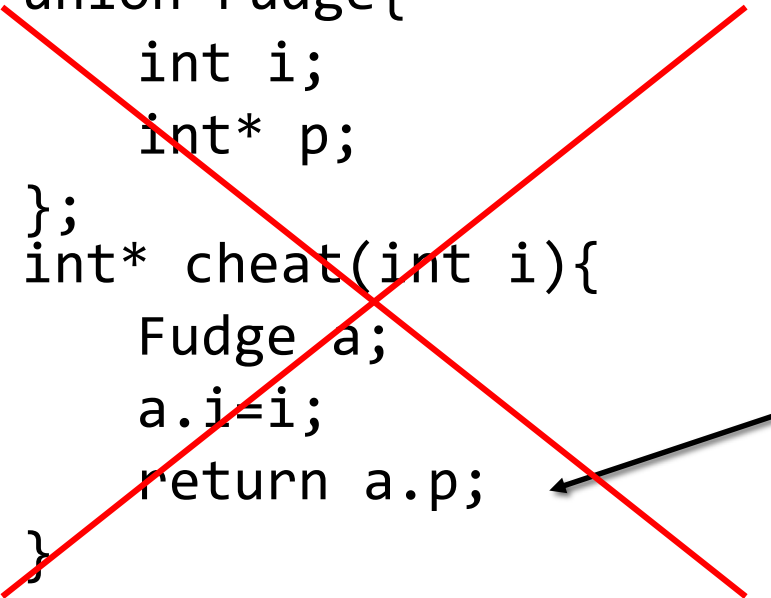
Példa

C++:

```
union typename{  
    typ1 field1;  
    ...  
    typm fieldm;  
};
```

```
union Fudge{  
    int i;  
    int* p;  
};  
int* cheat(int i){  
    Fudge a;  
    a.i=i;  
    return a.p;  
}
```

Jó ez??



Unió (Variáns rekord)

ADA

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

Unió (Variáns rekord)

ADA

- Hugó: Ember(Házas);
 - Családi_Áll, Neve, Neme, Születési_Ideje, Gyermekek_Száma, Házastárs_Neve
- Eleonóra: Ember(Egyedülálló);
 - Családi_Áll, Neve, Neme, Születési_Ideje, Gyermekek_Száma
- Ödön: Ember(Özvegy);
 - Családi_Áll, Neve, Neme, Születési_Ideje, Gyermekek_Száma, Házastárs_Halála
- Vendel: Ember(Elvált);
 - Családi_Áll, Neve, Neme, Születési_Ideje, Gyermekek_Száma, Válás_Dátuma, Gyerekek_Gondozója
- Aladár: Ember; -- Egyedülálló
Családi_Áll, Neve, Neme, Születési_Ideje, Gyermekek_Száma
- Helytelen (futási idejű hiba, altípus-megszorítás megsértése):
 - Hugó.Válás_Dátuma, Aladár.Házastárs_Neve

Unió (Variáns rekord)

ADA

- Megszorítatlan altípus használata
 - Aladár : Ember; -- alapért. Egyedülálló
 - Aladár := (Házass,);
 - Aladár := Elek;
 - Aladár := Hugó;
- A szerkezetét megváltoztathatjuk, diszkriminánsostul
- Csak úgy, ha az egész rekord értéket kap egy értékadásban

Unió (Variáns rekord)

Bizonyos programozási nyelvekben nincs unió típus, a tervezők az osztályok és az öröklődés használatát javasolják helyette:

- SmallTalk,
- Eiffel,
- Java,
- C#

Halmaz

Speciális iterált típus

- Mi lehet az eleme?
- Hány eleme lehet?
- Megvannak-e a 'hagyományos' halmazműveletek?

Halmaz

Pascal

- Alaphalmaz: diszkrét típus
- Elemek száma: max. 256
- Értékek sorszáma csak 0..255 között
 - `type Small_Letters = set of 'a'..'z';`
 - `type Digits = set of '0'..'9';`
 - `type Day = (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);`
 - `type Days = set of Day;`
`var A, B : Days;`
 - `A:=[Monday, Wednesday]` - halmazkonstruktor
- üres halmaz: `[]`

Halmaz

Pascal

- Műveletek:
 - értékadás,
 - halmazműveletek:
 - eleme-teszt (in),
 - $=$, $\subset$, $\supset$, részhalmaz reláció ('<=', '>=')
 - (a '<' és '>' nem megengedett!)
 - unió ('+'), differencia ('-').
 - metszet ('*')
- Ez a precedencia-sorrend is.

Mikor ekvivalens két típus?

```
x, y: array[0..9] of integer;  
z: array[0..9] of integer;
```

Strukturális ekvivalencia esetén:

- A rekordok mezőnevei is figyelembe vannak véve, vagy csak a struktúrájuk?
- Számít-e a rekordmezők sorrendje?
- Tömböknél elég-e az indexek számosságának egyenlőnek lenni, vagy az indexhatároknak is egyezniük kell?

Mikor ekvivalens két típus?

Értékül adhatóak egymásnak? Akarjuk?

```
Dimensions = RECORD
```

```
    Breadth: REAL;
```

```
    Length: REAL;
```

```
END;
```

```
Complex = RECORD
```

```
    RealPart: REAL;
```

```
    ImPart: REAL;
```

```
END;
```

```
Size: Dimensions;
```

```
Root: Complex;
```

Név szerinti ekvivalencia esetén

Deklarálhatók-e egy típushoz típusok, amelyekkel ekvivalens?

Névtelen tömb- illetve rekordtípusok ekvivalensek-e valamivel?

Típuskonverziók

Van-e, és hogyan működik?

- az automatikus konverzió?
- az identitáskonverzió?
- a bővítő konverzió?
- a szűkítő konverzió?
- a toString konverzió?

Típuskonverziók

Java

- identitáskonverzió
 - a boolean csak ez szabad
- bővítő konverzió
 - byte to short, int, long, float or double
 - short to int, long, float or double
 - int to long, float or double
 - long to float or double
 - float to double

Típuskonverziók

Java

- szűkítő konverzió
 - byte to char
 - short to byte or char
 - char to byte or short (miért is?)
 - int to byte, short or char
 - long to byte, short, char or int
 - float to byte, short, char, int or long
 - double to byte, short, char, int, long or float

Változók

Változó = (név, attribútumhalmaz, hely, érték)

Láthatóság, Elérhetőség

Hogyan definiálhatunk változókat és konstansokat?

Változók

	szintaxis	példa
Pascal	<code>var <identif>: <type>;</code>	<code>var i : integer;</code>
C++	<code><type> <identif>[= <value>;</code>	<code>int i = 0;</code>
Java	<code><type> <identif>[= <value>;</code>	<code>int i = 0;</code>
ADA	<code><identif>: <type>[:= <value>;</code>	<code>I : Integer := 0;</code>
CLU	<code><identif>: <type>[:= <value>;</code>	<code>i : int := 0;</code>
Eiffel	<code><identif>: [expanded]<type>;</code>	<code>I : INTEGER;</code>

Konstansok

	szintaxis	példa
Pasc.	<code>const <name> = <value>;</code>	<code>const val = 3;</code>
C++	<code>#define <name> <value> (?)</code> vagy <code>const <type> <name> = <value>;</code>	<code>#define val 3</code> <code>const int val = 3;</code>
Java	<code>final <type> <name> = <value>;</code>	<code>final int val = 3;</code>
ADA	<code><name> :</code> <code>constant <type>:= <value>;</code>	<code>Val :</code> <code>constant Integer := 3;</code>
C#	<code>const <type> <name> = <value>;</code>	<code>const double PI = 3.14159;</code>
Eiffel	<code><name> : <type> is <value>;</code>	<code>A: INTEGER is 3;</code>

Kifejezések

Prefix jelölés

- $+ a b$

Postfix jelölés

- $a b +$

Infix jelölés

- $a + b$

A műveletek precedencia szintjei fontosak

Precendica - Pascal

zárójel:	()
függvényhívás	fv(..)
unáris operátorok	not, @, ^, +,-
multiplikatív operátorok	*, /, and, div, mod, shl, shr
additív operátorok	+, - or, xor
relációk	in, <, >, <=, >=, <>
balról jobbra kiértékelés	

Precedencia – C++

zárójelek	()
scope	::
selection, call, size	. -> [] () sizeof
postf, pref, compl, not, un. add. address of, deref, cre., destroy, cast	++ --, ~ ! + -, & * new, delete ()
member	.* ->*
multipl. op.	* / %
binary add.	+ -
shift	<< >>
reláció	< <= > >=
egyenlőség	== !=
bitwise AND	&
bitwise excl. OR	^
bitwise OR	
logical AND	&&
logical OR	
cond. expr.	? :
értékdások	= *= /= %= += -= >>= <<= &= ^= !=
throw	throw
comma (sequence)	,

Precedencia – Java

postfix	. [] () ++ --
prefix	++ -- ~ ! + -
constr, cast	new ()
multipl. op.	* / %
binary add.	+ -
shift	<< >> >>>
relational	< <= > >= instanceof
equality	== !=
bitwise AND	&
bitwise excl. OR	^
bitwise OR	
logical AND	&&
logical OR	
cond. expr.	? :
értékadások	= *= /= %= += -= >>= <<= >>>= &= ^= !=

Érdekesség

Algol 68

- A kétoperandusú operátorok prioritása általában felüldefiniálható a `prio` kulcsszóval:
 - `prio + = 3, * = 2;`
 - `print(6+3*5);` # $(6+3)*5=45$ lesz az eredmény

Vezérlési szerkezetek

Egyszerű utasítások:

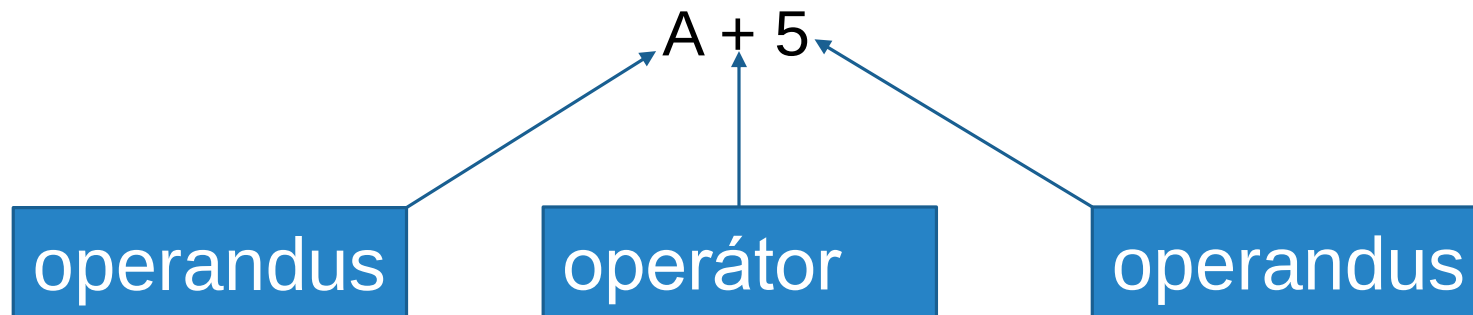
Értékadás – mit jelent?

- „változó” értékadásjel kifejezés
- helyette inkább:
- „balérték” értékadásjel „jobbérték”
 - (vagy fordítva?)
- Mit jelent ez?
 - Értékadásjel
 - Szokásosan := és =
 - De -> és <-
 - Szemantikája
 - „megfelelő” típusú kifejezés kell!

Kifejezés

operandusokból és operátorokból áll

- minden operandus lehet egy újabb kifejezés
- kiértékelem és megkapom az értékét:



Kifejezés az értékadás?

Wulf (BLISS): Of course, everything is

Richie (C): Yes, why not

Wirth (Pascal): No, only math-like things are expressions

Két vonulat:

- Pascal, CLU, ADA, Eiffel, ...
- C, C++, Java, ...

Van-e többszörös értékadás?

Szemlélet – COBOL

Eredetileg

- MOVE 23 TO A.
- MOVE B TO C.
- ADD A TO B GIVING C.
- SUBTRACT A FROM B GIVING C.
- MULTIPLY A BY B GIVING C.
- DIVIDE A BY B GIVING C.

Újabban

- COMPUTE A = (A + B - C / (A * B) - A * B)

Pascal

„balérték” := kifejezés;

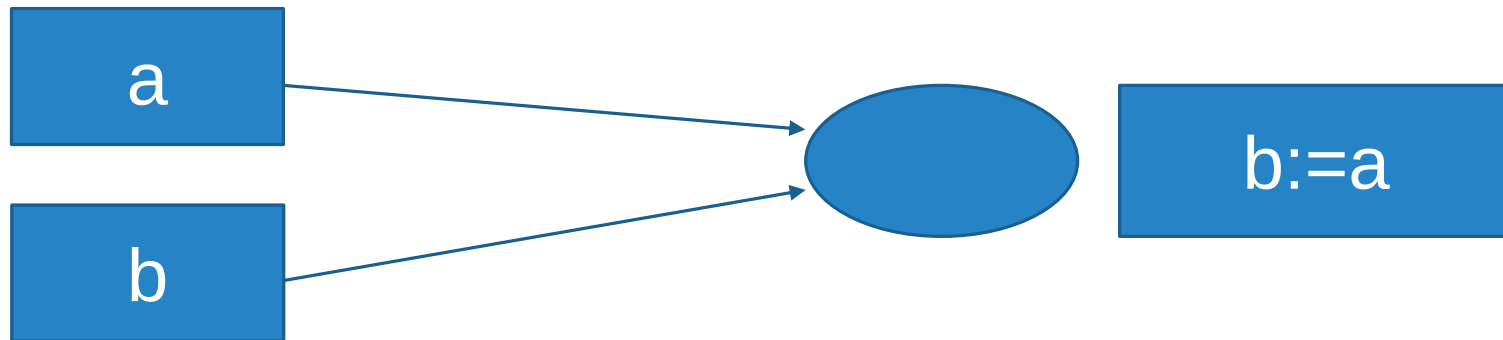
- A „ balérték” és a kifejezés típusa megegyező, de legalább kompatibilis kell legyen.
- m: integer;
 - m := m+1;
- vagy:
 - p := keres(gyoker, x);

Többszörös értékadás nem megengedett.

CLU

referenciákat használ

- értékadás hatására két referencia hivatkozhat ugyanarra az objektumra



- Többszörös értékadás megengedett:
 - $x, y := y, x.$

C++

Számos értékadó operátor jobbról balra feldolgozva

- `A=B=C`
 - jelentése: `A=(B=C);`

Értékadó operátorok

- `=` `*=` `/=` `%=` `+=` `-=` `>>=` `<<=` `&=` `^=`
- `y += g(x);`

Java

A C++ -hoz hasonló, számos értékadó operátor:

- `=` `*=` `/=` `%=` `+=` `-=` `>>=` `<<=` `>>>=` `&=` `^=`
- `E1 op= E2`
 - jelentése: `E1 = (T)((E1) op(E2))`,
 - ahol `T` az `E1` típusa.

Összetett értékadó operátoroknál mindkét operandus primitív típusú kell legyen

- kivéve: `+=`, ha a bal operandus `String` típusú),

Implicit cast előfordulhat!

- `short x=3; x+=4.6; eredménye: x ==7!`

`final`-nak deklarált változónak nem adható érték.

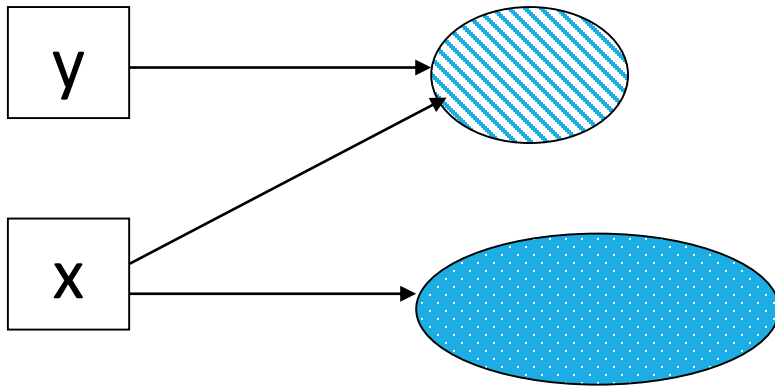
Alapvetően referenciákat használ

Eiffel

Az értékadás és a paraméterátadás szemantikája megegyezik.

- Ha $x:TX$, $y:TY$
 - akkor az $x:=y$ eredménye
- TX és TY -től függ: referencia vagy „kiterjesztett” típusok?

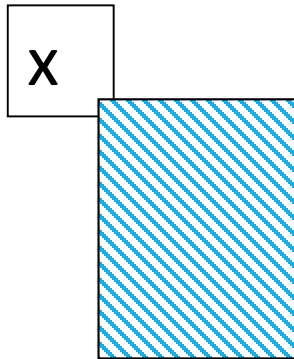
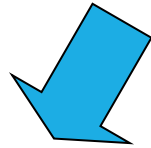
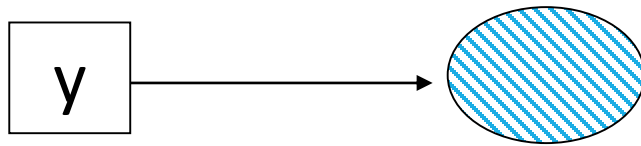
TX, TY referencia



$x := y$

referencia újrahozzárendelés

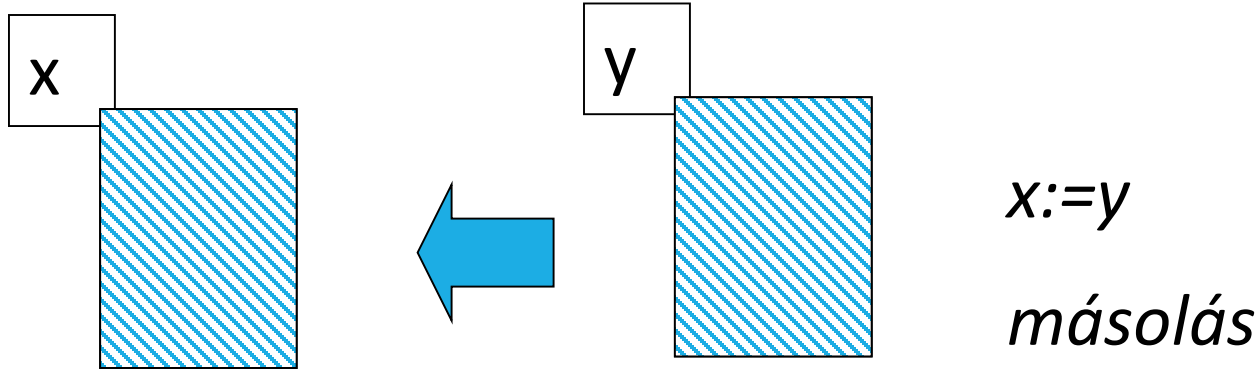
TX kiterjesztett, TY referencia



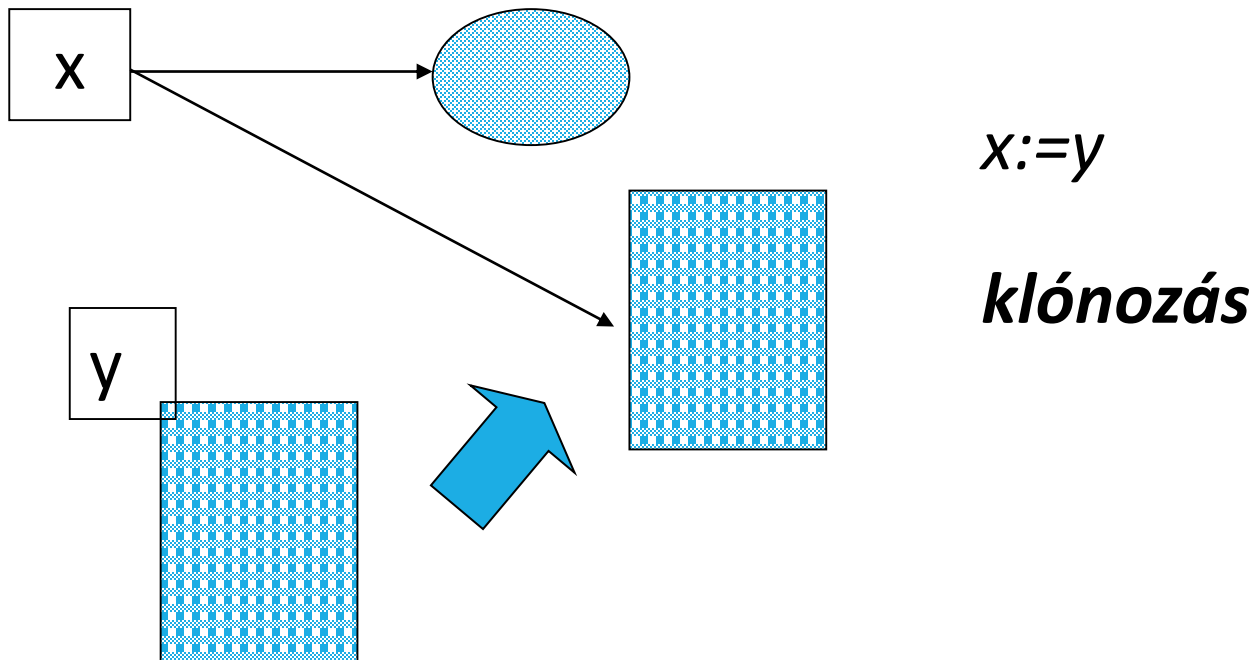
$x := y$

másolás

TX kiterjesztett, TY kiterjesztett



TX referencia, TY kiterjesztett



Üres utasítás

Pascal

- megengedett (case)

ADA

- null; (case)

C++, Java, stb.: “;” használható

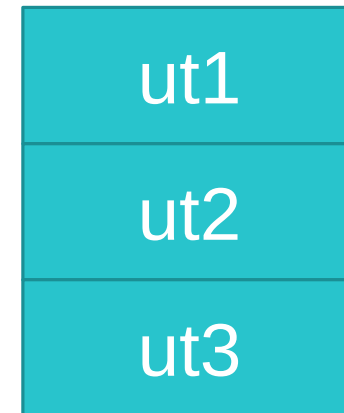
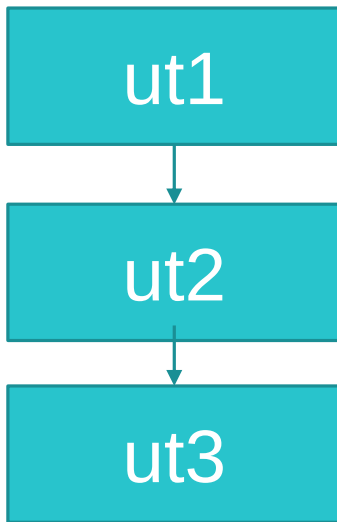
- Például ciklusnak lehet üres törzse

Eiffel: “;” használható

- nincs valódi szerepe

Összetett utasítások

Szekvencia:



ut1; ut2; ut3;

Szekvencia

Lehet-e üres?

Terminátor vagy utasításelválasztó-e a ';'?

Lehet-e blokkutasítást létrehozni?

Elhelyezhető-e a blokkutasításban deklaráció?

Szekvencia

Pascal: a “;” elválasztja az utasításokat:

- `begin ut1; ut2; ut3 end`
- üres utasítás is lehet:
 - `begin ut1; ut2; ut3; end`
- Az üres utasítás lehetősége miatt a “;” beírása megváltoztathatja a program jelentését!

ADA: a “;” lezárja az utasítást

C++, Java: a “;” lezárja az utasítást

- de nem mindet, például a blokk utasítást nem

Eiffel: nincs szükség elválasztójelre, de a “;” megengedett.

Blokk utasítások

Utasítások sorozatából alkothatunk egy összetett utasítást, blokkot.

Bizonyos nyelvekben lehet deklarációs része is.

Főleg azokban a nyelvekben fontos, ahol a feltételes és a ciklus utasítások csak egy utasítást tartalmazhatnak

- Pascal, C++, Java, ...

Pascal: begin utasítássorozat end

ADA:

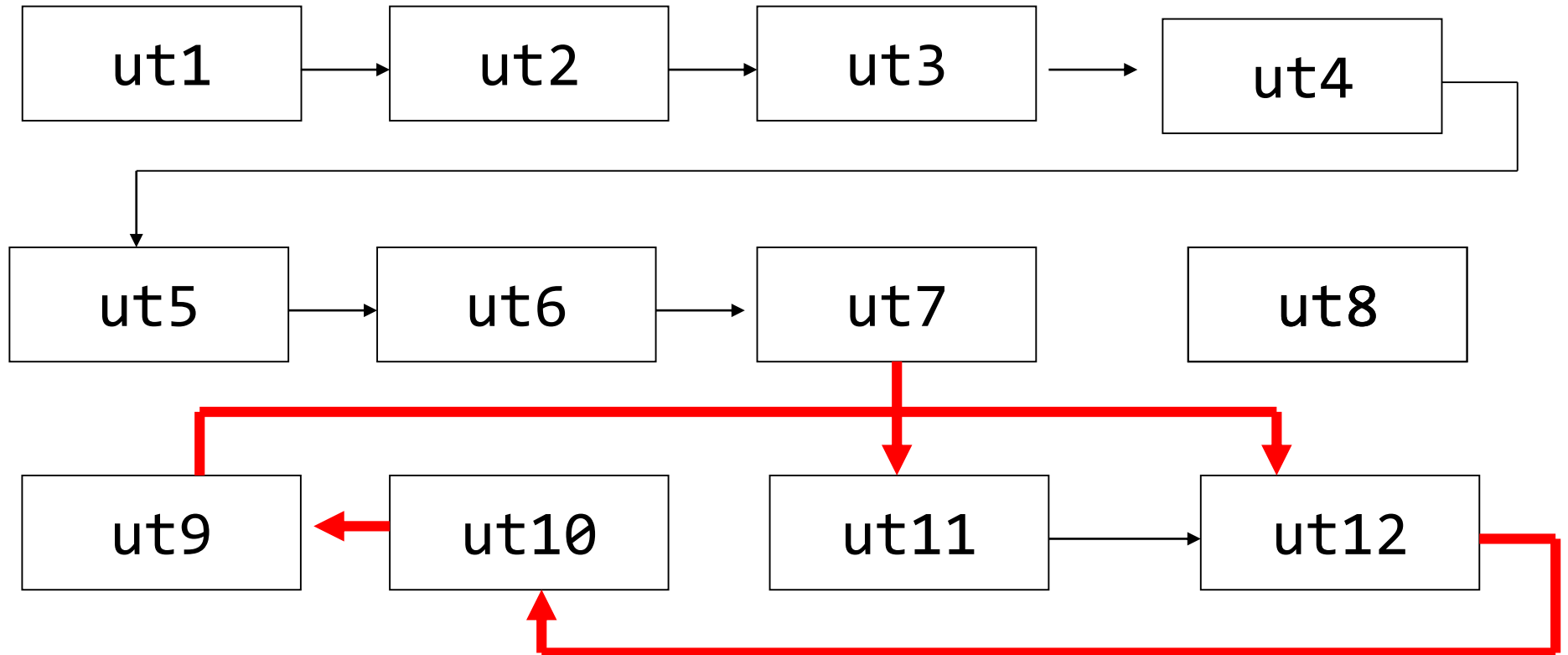
- [declare
deklarációk]
begin utasítássorozat [kivételkezelő rész] end;
- Például
 - Csere:
declare
Temp : Integer;
begin
Temp := I; I := J; J := Temp;
end Csere;

C++, Java, ...

- “{” és “}” között.

Feltétel nélküli vezérlésátadás

goto



Feltétel nélküli vezérlésátadás

javaslat: goto nélkül

ut1
ut2
ut3
ut4

Feltétel nélküli vezérlésátadás

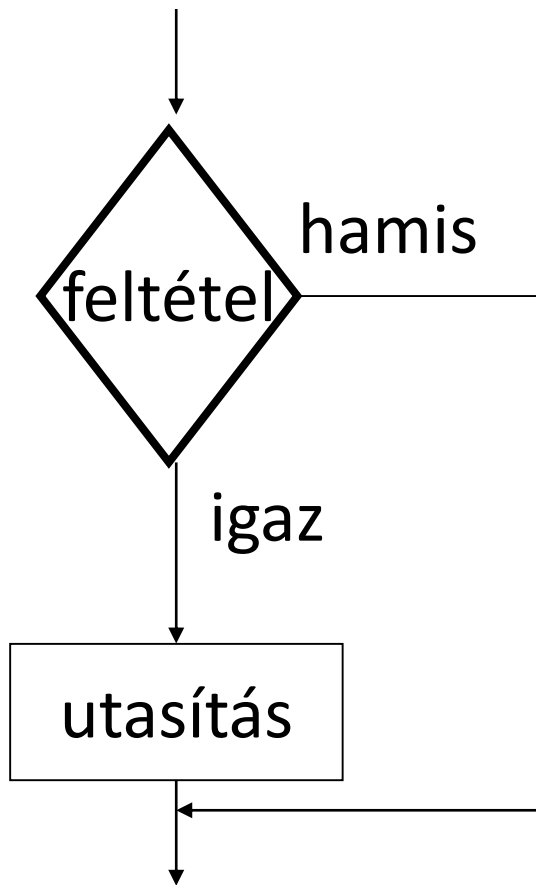
Ha mégis – Pascal, C stb.

- címke:
- utasítások ...
- goto címke;

Nincs: Modula-3, Java, ...

Elágazások

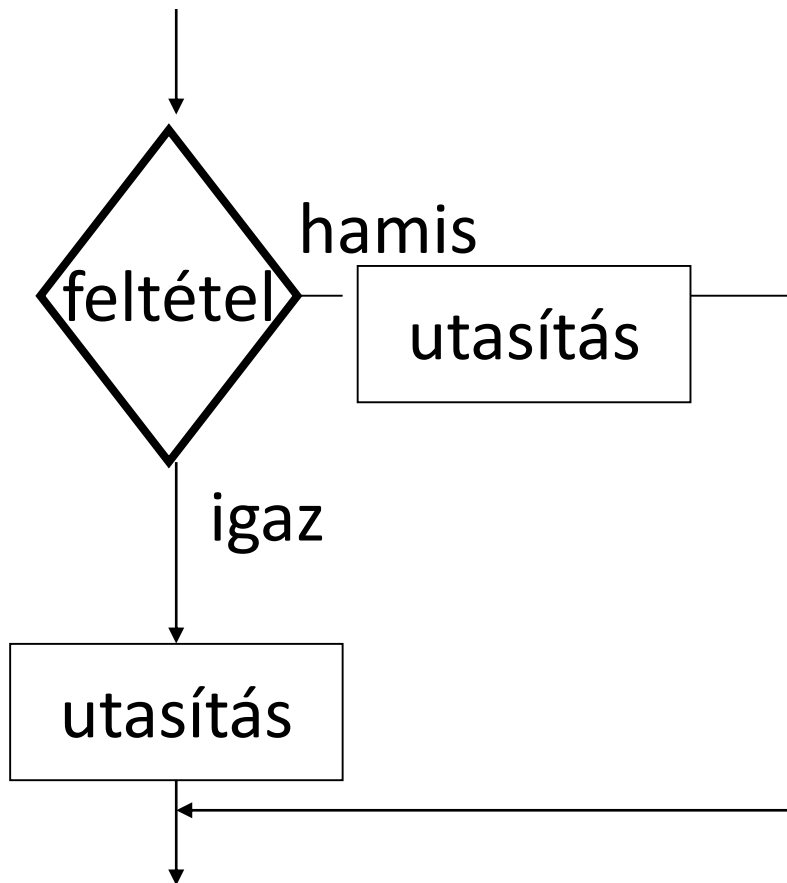
- Feltétel értékétől függően valamilyen utasítás(csoport) végrehajtása



feltétel	
igaz-ág	üres utasítás

Elágazások:

- Feltétel igaz vagy hamis értékétől függően valamilyen utasítás(csoport) végrehajtása



Elágazások

`if feltétel then ut`

`if feltétel then ut1 else ut2`

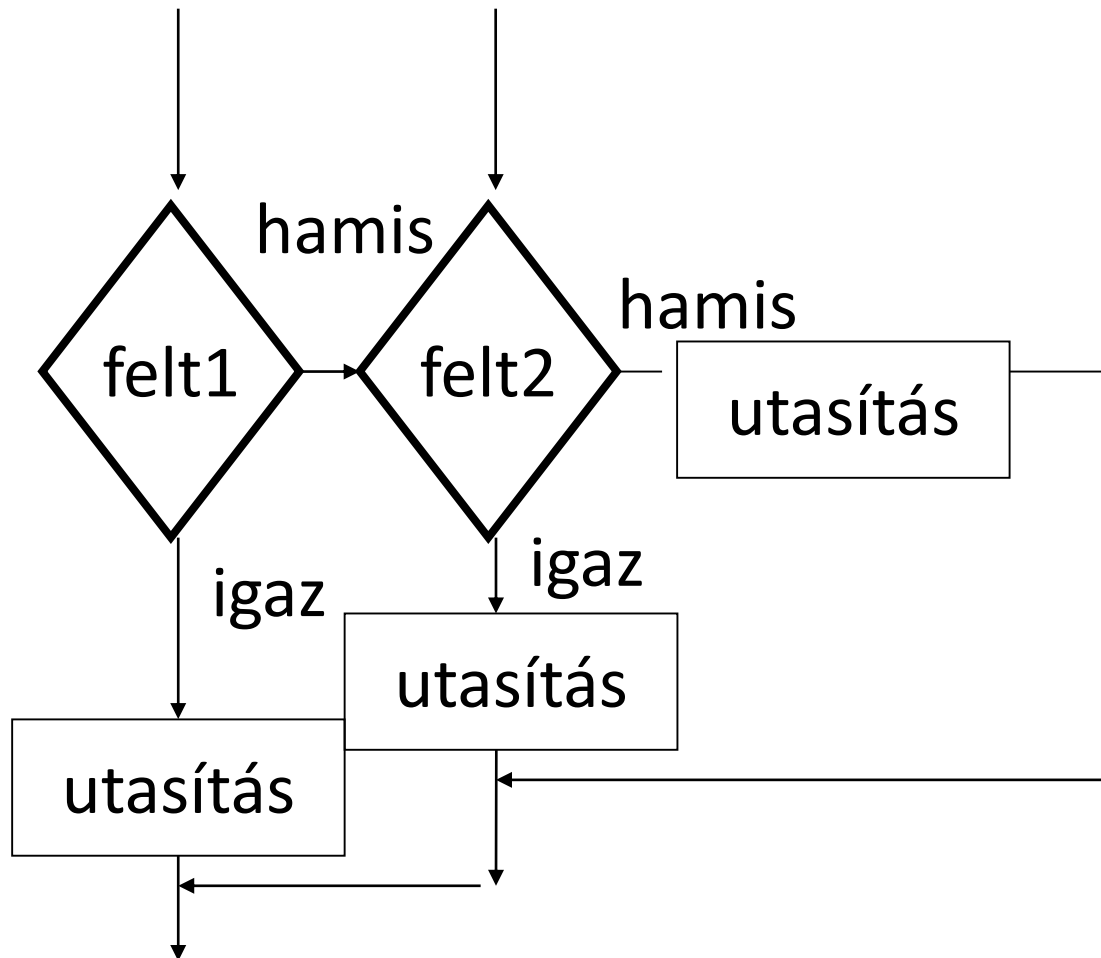
- “csellengő” else:
 - `if felt1 then if felt2 then ut1 else ut2`
- mit jelent:
 - `if felt1 then (if felt2 then ut1 else ut2)`
- vagy
 - `if felt1 then (if felt2 then ut1) else ut2 ?`

Elágazások – csellengő else

Különböző megoldások:

- blokk utasítás kell a beágyazásnál
- az `else` rész a legbelső `if`-hez tartozik
- `explicit end` konstrukció bevezetése

Többirányú elágazások



Többirányú elágazások

feltétel1		
igaz1-ág	feltétel2	
	igaz2-ág	hamis-ág

Többirányú elágazások

elsif lehetősége – óvatos módosítás kell!

```
if felt1 then s1;  
elsif felt2 then s2;  
elsif felt3 then s3;  
else s4;  
end if;
```

Pascal

`if felt then S1 [else S2]`

- több utasítás: blokk kell

“csellengő” else:

- legbelső if :
 - `if felt1 then`
 `if felt2 then S1`
 `else S2`

különben blokk:

- `if felt1 then`
 `begin`
 `if felt2 then S1`
 `end`
`else S2`

Pascal

```
if (a>b)
  then writeln("a>b")
else
  if (a<b) then writeln("a<b")
  else writeln("a=b");
```

Pascal

Vigyázat!

- `if expr then st1 else st2`
nem ugyanaz, mint:
- `if expr then st1; else st2`
ez ekvivalens:
- `if expr then st1; ??? else st2`
- ha
 - `if expr then S1 else S2` -ben
S1 után beszúrunk egy `;`-t, az hiba!

ADA

end if a végén, elsif
megengedett:

```
if <expr>1 then
    <statm>1;
{elsif (<expr>2)then
    <statm>2;}
[else
    <statm>3; ]
end if;
```

```
if A>B then
    Put_Line("a>b");
elsif A<B then
    Put_Line("a<b");
else
    Put_Line("a=b");
end if;
```

C++

aritmetikai kifejezés kell,
nem nulla: true.

“csellengő” else: legbelső if

- blokk kell különben

```
if (<expr>
    <statm1>
[else
    <statm2>]
```

```
if (a>b) cout << "a>b";
else
    if (a<b) cout << "a<b";
    else cout << "a=b";
```

aritmetikai if kifejezés pl.:

```
if (a<=b) max=b;
else max=a;
```

inkább:

```
max=(a<=b)?b:a;
```

Java

Hasonló a C++-hoz

- kivéve a kifejezés típusa boolean kell legyen

```
if (<expr>
    <statm1>
[else
    <statm2>]
```

```
if (a>b)
    System.out.println("a>b");
else
    if (a<b)
        System.out.println("a<b");
    else
        System.out.println("a=b");
```

Eiffel

end zárja le az utasítást.

Megengedett az elseif használata.

```
if <expr>1 then
    <statm>1
{elseif <expr>2 then
    <statm>2 }
[else
    <statm>3 ]
end
```

```
if (a>b) then
    io.putstring("a>b");
elseif (a < b) then
    io.putstring("a < b")
else
    io.putstring("a = b");
end
```

Python

Csellengő else: a bekezdés számít!!

```
a, b = 1, 2
if a == 1:
    if b == 1:
        print ("a és b is 1")
else:
    print ("a nem 1")
```

Python

Csellengő else: a bekezdés számít!!

Így mást jelent:

```
a, b = 1, 2
if a == 1:
    if b == 1:
        print ("a és b is 1")
    else:
        print ("a nem 1 és b nem 1")
```

if és értékadás

Ruby:

- `num += -1 if num < 0`

Scala:

- `val x = if (a > b) a else b`

Jobban olvasható, mint `a ? b : c` :