Programozási nyelvek és módszerek

3. ELŐADÁS – VEZÉRLÉSI SZERKEZETEK, ALPROGRAMOK

Vezérlési szerkezetek

Egyszerű utasítások:

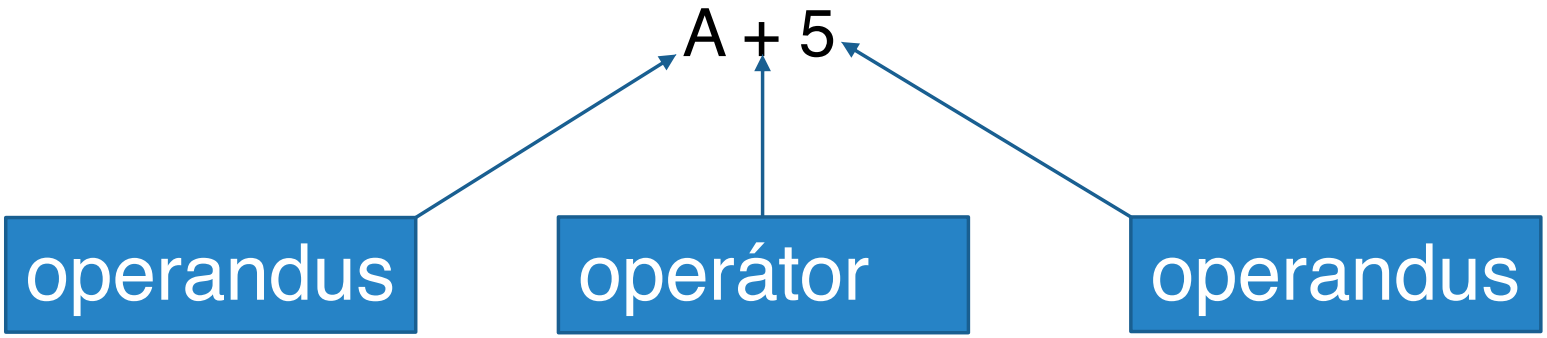
Értékadás – mit jelent?

- „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;

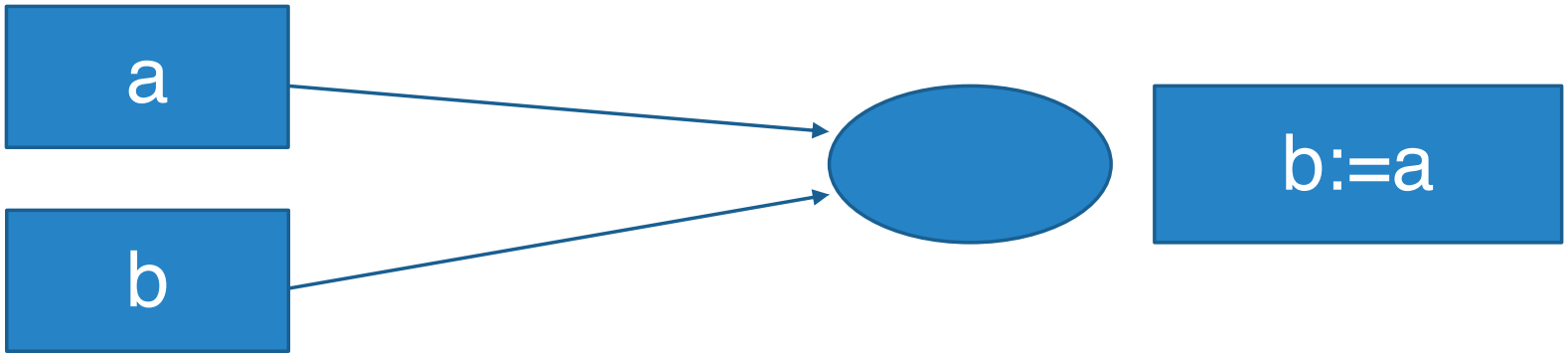
- A „ balérték” és a kifejezés típusa megegyező, de legalább kompatibilis kell legyen.
- m: integer;
  - `m:= m+1;`
- vagy:
  - `p := keres(gyoker, x);`

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.

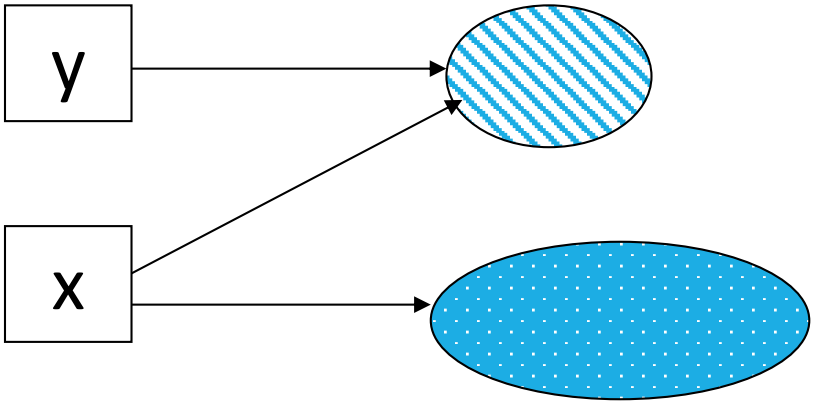
Alapvetően referenciákat használ

Eiffel

Az értékadás és a paraméterátadás szemantikája megegyezik.

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

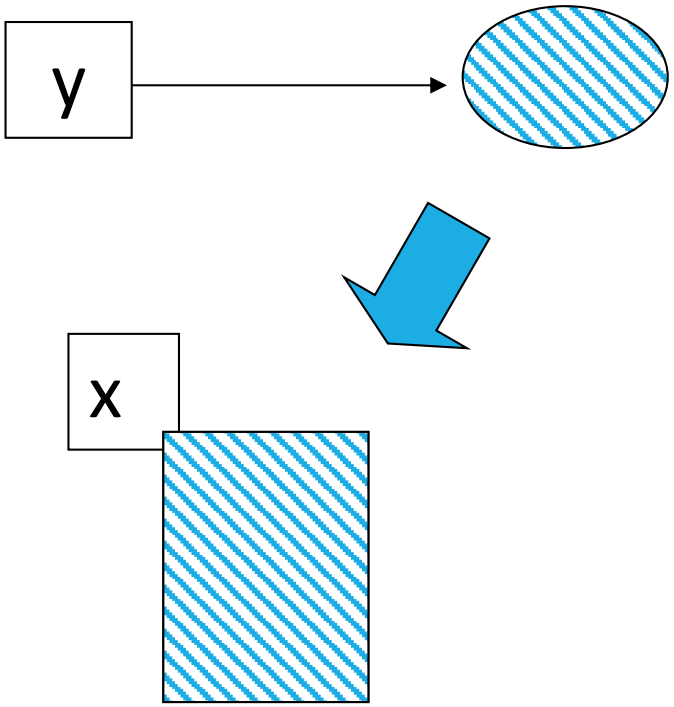
TX, TY referencia



x:=y

referencia újrahozzárendelés

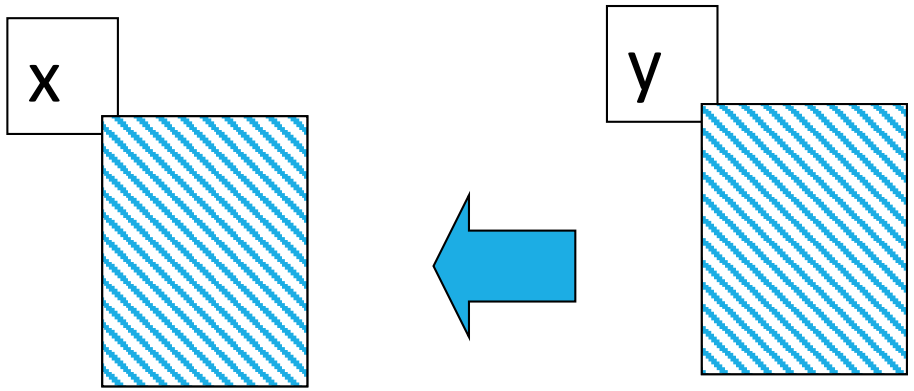
TX kiterjesztett, TY referencia



x:=y

másolás

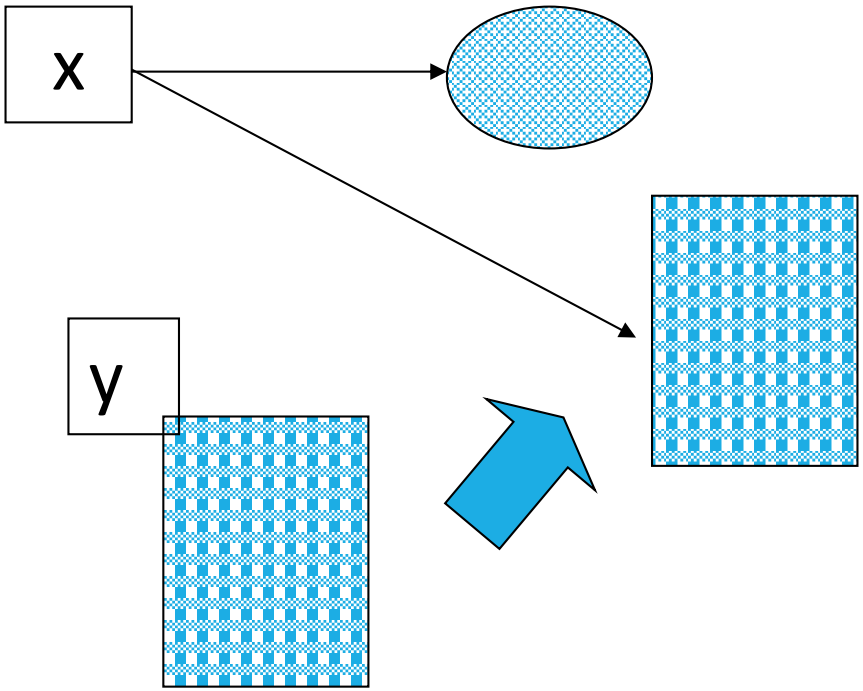
TX kiterjesztett, TY kiterjesztett



x:=y

másolás

TX referencia, TY kiterjesztett



x:=y

klónozás

Üres utasítás

Pascal
- megengedett (`case`)

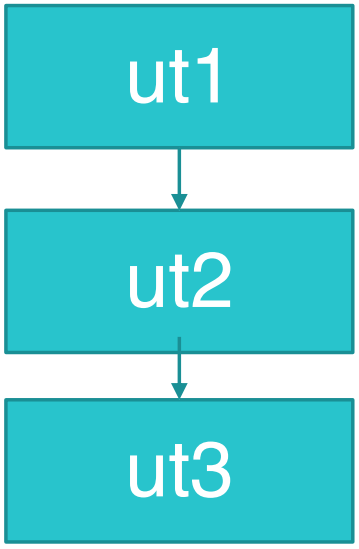
ADA
- null; (`case`)

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:

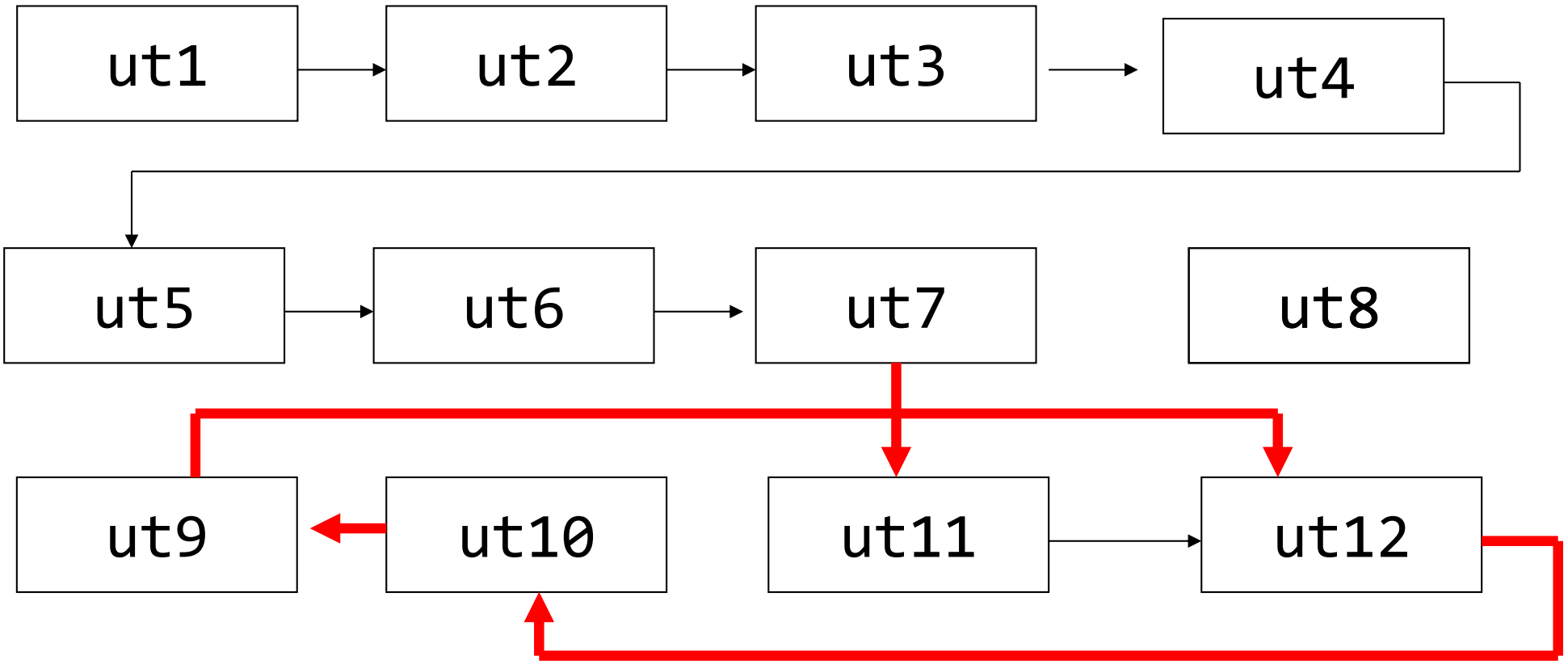
- `[declare deklarációk] begin utasítássorozat [kivételkezelő rész] end;`
- Például
  - ```
    Csere:
      declare
      Temp : Integer;
        begin
          Temp := I; I := J; J := Temp;
        end Csere;
    ```

C++, Java, …

- “{“ és “}” között.
    ```

Feltétel nélküli vezérlésátadás

goto



Feltétel nélküli vezérlésátadás

javaslat: goto nélkül

`ut1`
`ut2`
`ut3`
`ut4`

Feltétel nélküli vezérlésátadás

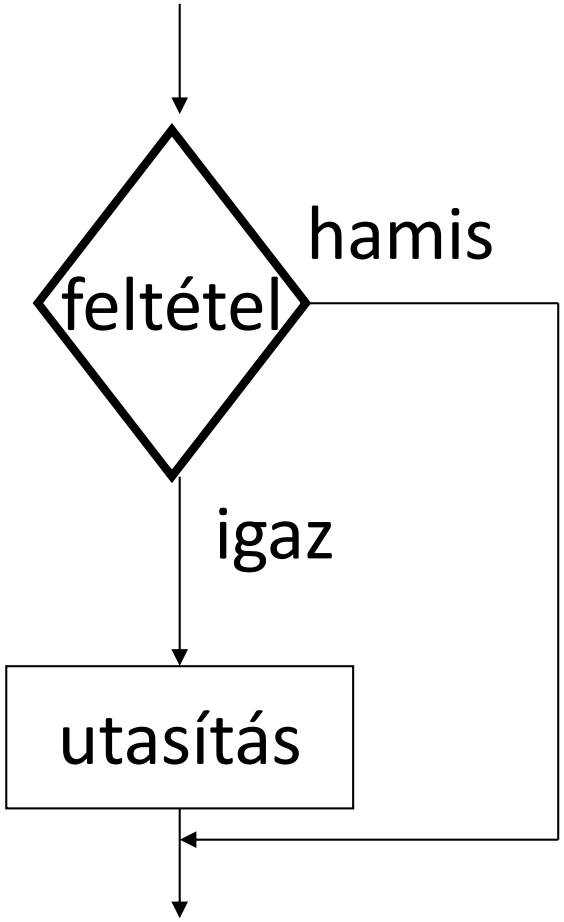
Ha mégis – Pascal, C stb.

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

Nincs: Modula-3, Java, …

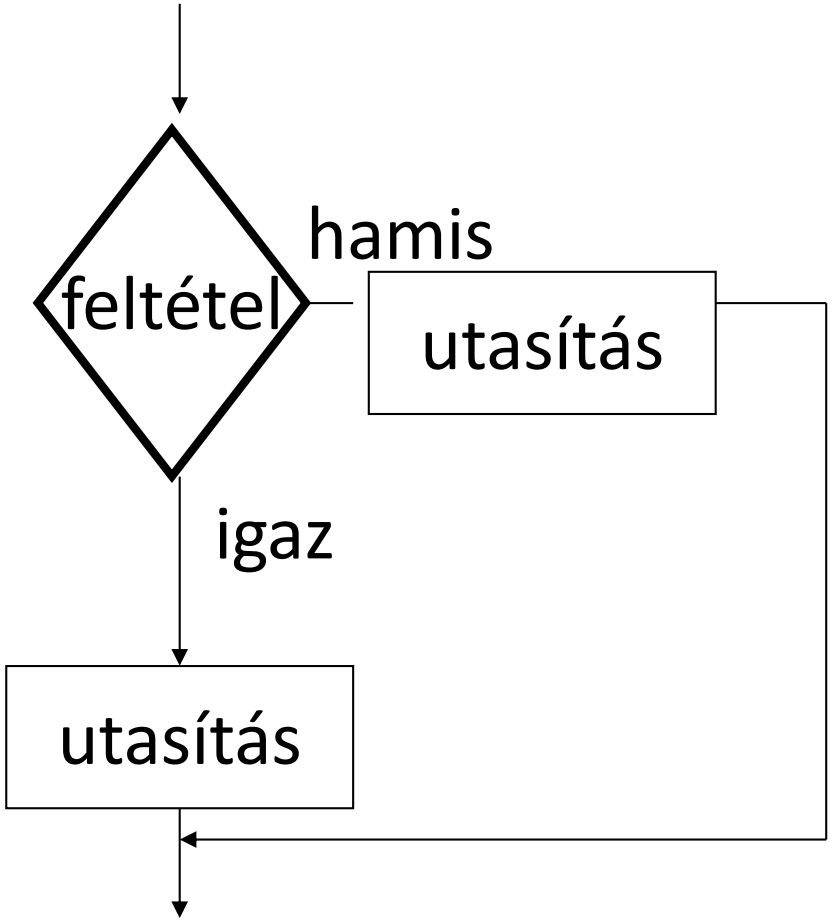
Elágazások

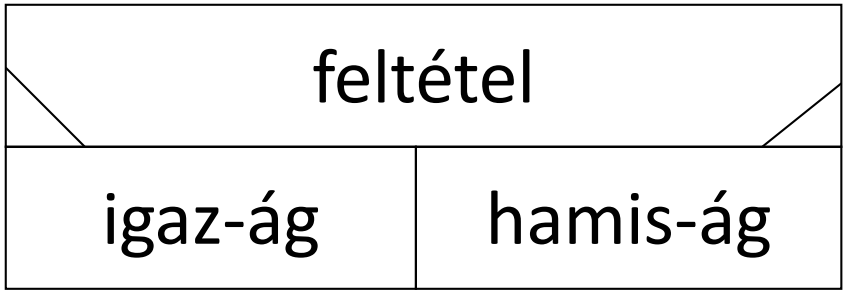
- Feltétel értékétől függően valamilyen utasítás(csoport) végrehajtása



Elágazások:

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





Elágazások

```
if feltétel then ut

if feltétel then ut1 else ut2
```

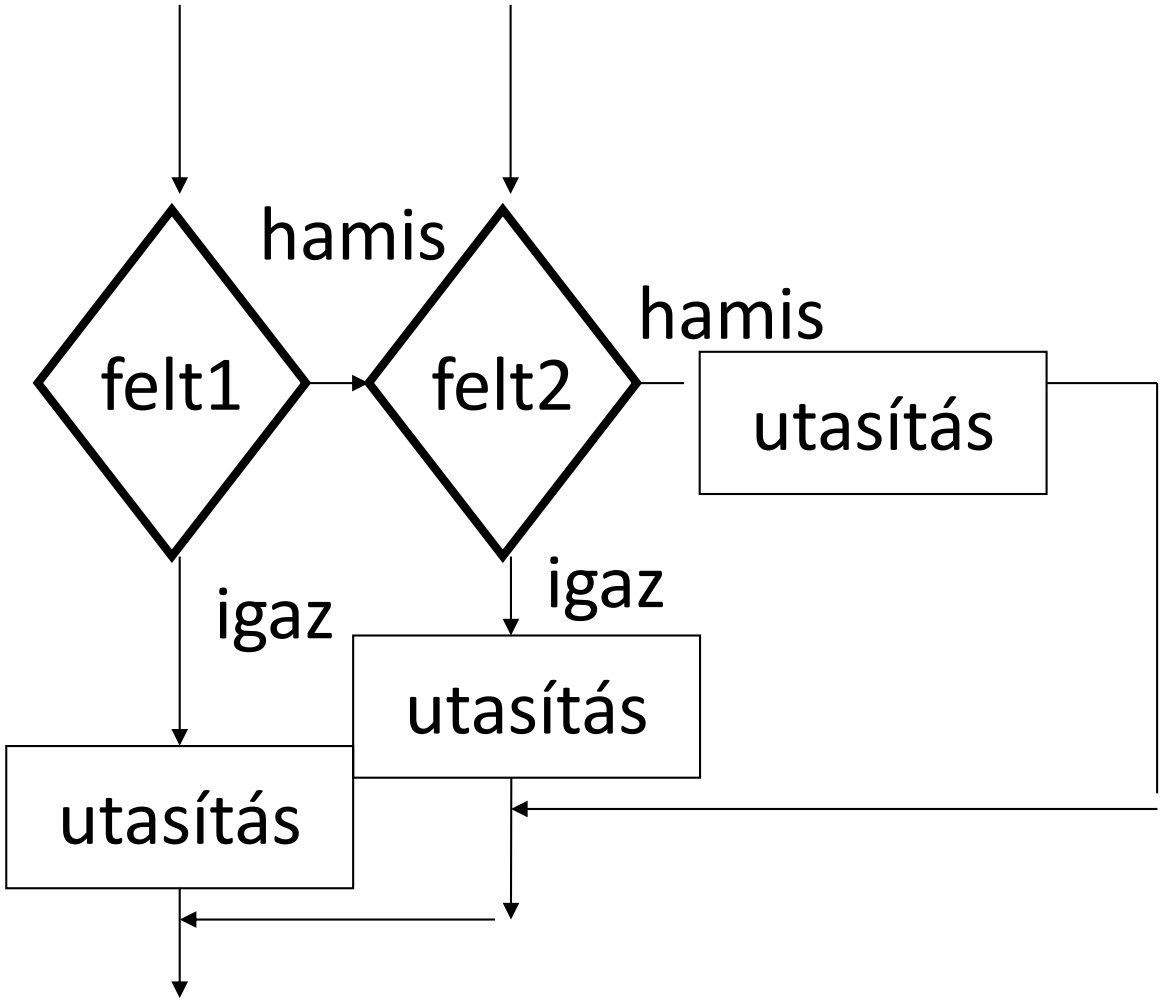
- “csellengő” else:
  - `if felt1 then if felt2 then ut1 else ut2`
- mit jelent:
  - `if felt1 then (if felt2 then ut1 else ut2)`
- vagy
  - `if felt1 then (if felt2 then ut1) else ut2 ?`

Elágazások – csellengő else

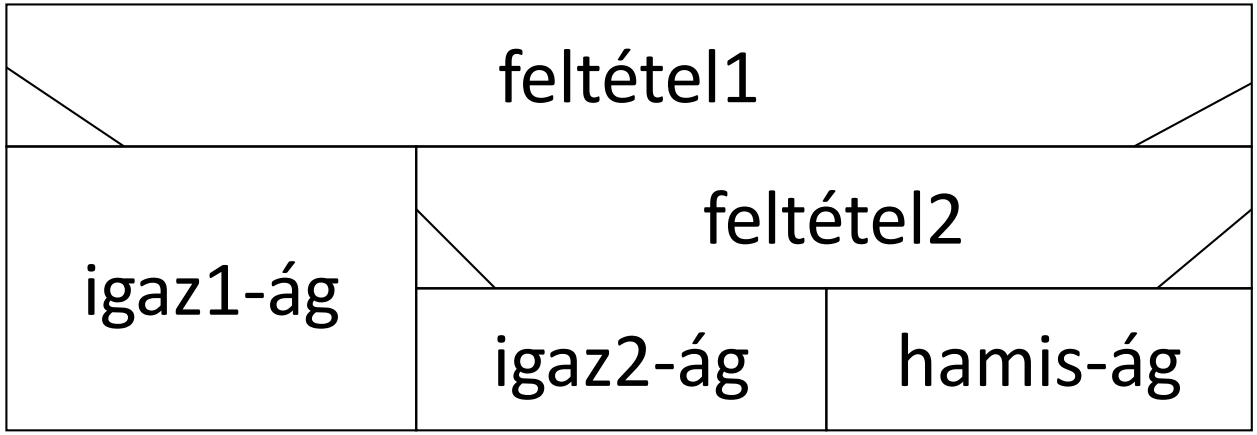
Különböző megoldások:

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

Többirányú elágazások



Többirányú elágazások



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

- kivévea 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 `? :`

Esetkiválasztásos elágazások

valamilyen kifejezés (szelektor) értékétől függően:

kifejezés					
é1	é2	é3	é4	…	…
i	i	i	i	i	i
g	g	g	g	g	g
a	a	a	a	a	a
z	z	z	z	z	z
1	2	3	4	…	…
-	-	-	-	-	-
á	á	á	á	á	á
g	g	g	g	g	g

Esetkiválasztásos elágazás

Kérdések

- Mi lehet a szelektor típusa?
- Fel kell-e sorolni a szelektortípus minden lehetséges értékét?
- Mi történik, ha fel nem sorolt értéket vesz fel a szelektor?
- „Rácsorog”-e a vezérlés a következő kiválasztási ágakra is?
- Diszjunktnak kell-e lennie a kiválasztási értékeknek?
- Mi állhat a kiválasztási feltételben?
  - egy érték
  - értékek felsorolása
  - intervallum

“Case” utasítások

```
case expr of
  const1: st1;
  const2: st2;
 …
  constn: stn
end
```

“Case” utasítások

Sokszor igaz a case konstansokra:

- Tetszőleges sorrend,
- Nem feltétlenül egymás után,
- Több is vonatkozhat ugyanarra az alutasításra,
- Mind különböző kell legyen - különben, ha $const_i$ és $const_j$ egyenlőek, akkor melyiket választanánk, st_i -t vagy st_j-t?
- A kifejezés típusa diszkrét kell legyen (egész vagy felsorolási típ)
- Kell adni egy “others” ágat, hogy minden lehetőséget lefedjünk.

Pascal

A szelektor típusa lehet: integer, character, boolean vagy tetszőleges felsorolási vagy intervallum típus.

"else" megengedett, de nem kötelező (skip).

```
case var of
  val1: statm1;
  val2.: statm2;
...
  vali…valj : statmi;
[else statm]
end;
```

```
var Age: Byte;
case Age of
  0..13: Write(‘Child’);
 14..23: Write('Young');
 24..65: Write(‘Adult’);
 else    Write(‘Old’);
end
```

ADA

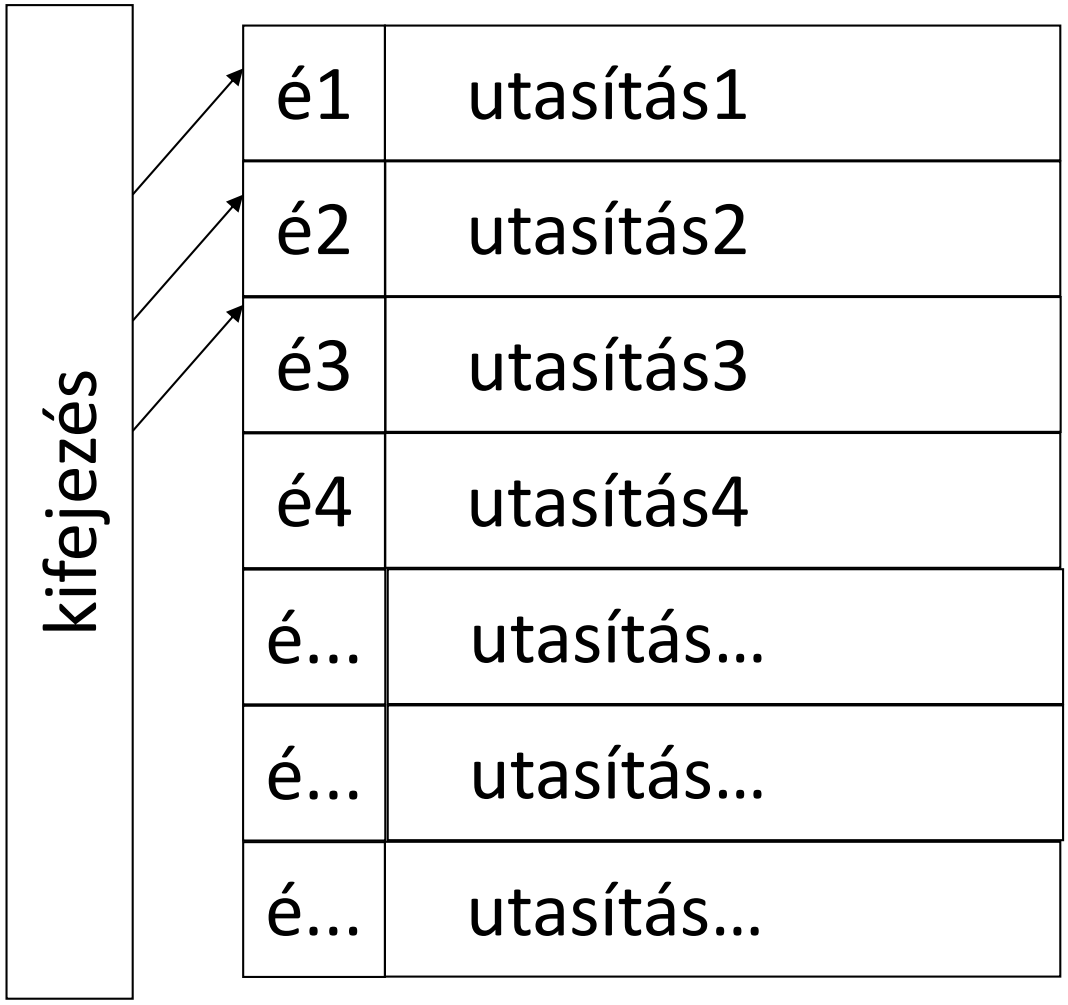
```
case expr is
 when choice1 => statms1;
 when choice2 => statms2; ...
 when others =>  statms;
end case;
```

others kötelező, ha nincs minden lefedve!

.. - intervallum,
| - vagy

```
 case Today is
  when Monday => Initial_Balance;
  when Friday => Closing_Balance;
  when Tuesday..Thursday =>
Report(Today);
  when others => null;
end case;
```

C++


kifejezés
é1
utasítás1
é2
utasítás2
é3
utasítás3
é4
utasítás4
é...
utasítás...
é...
utasítás...
é...
utasítás...

break

C++

```
switch (intexpr) {
  case label1: statm1; break;
  case label2: statm2; break;
...
  default: statm;
}
```

kifejezés “integral” típusú

- “default:” címke lehetősége (nem kötelező).
- itt break kell, ha nem akarom folytatni!

C++

```
switch (val){
  case 1 : cout<”case 1\n”;
  case 2 : cout<”case 2\n”;
  default: cout<”default case ”;
}
```

ha val=1, az eredmény:

```
  case 1
  case 2
  default case
```

Java

Mint a C++:

```
 switch (val){
 case 1 : System.out.println(”case 1\n”);
 case 2 : System.out.println(”case 2\n”);
                          break;
 default: System.out.println(”default case ”);
}
```

ha val=1, az eredmény:

```
case 1

case 2
```

Eiffel

Pascal-szerű szemantika

A változó típusa INTEGER vagy CHARACTER

```
inspect var
  when expr1 then statmblock1
  when expr2 then statmblock2
...
  else statmblock
end
```

Exception lép fel, ha nem gondoskodtunk a megfelelő választék-elemről (null utasítás szükséges lehet).

Eiffel

```
inspect c
  when '0'..'9'              then n:= n + 1;
  when 'a'..'z','A'..'Z' then s:= s + 1;
  else  o:= o + 1;
end
```

C#

```
switch (kif)
{
      case ertek1 : utasítások… break;
      case ertek2 : utasítások… break;
      case ertek3 : utasítások… break; vagy:
                    goto case KONST/default;
…
}
```

szemantika

Ciklusként is működhet!!

SWIFT

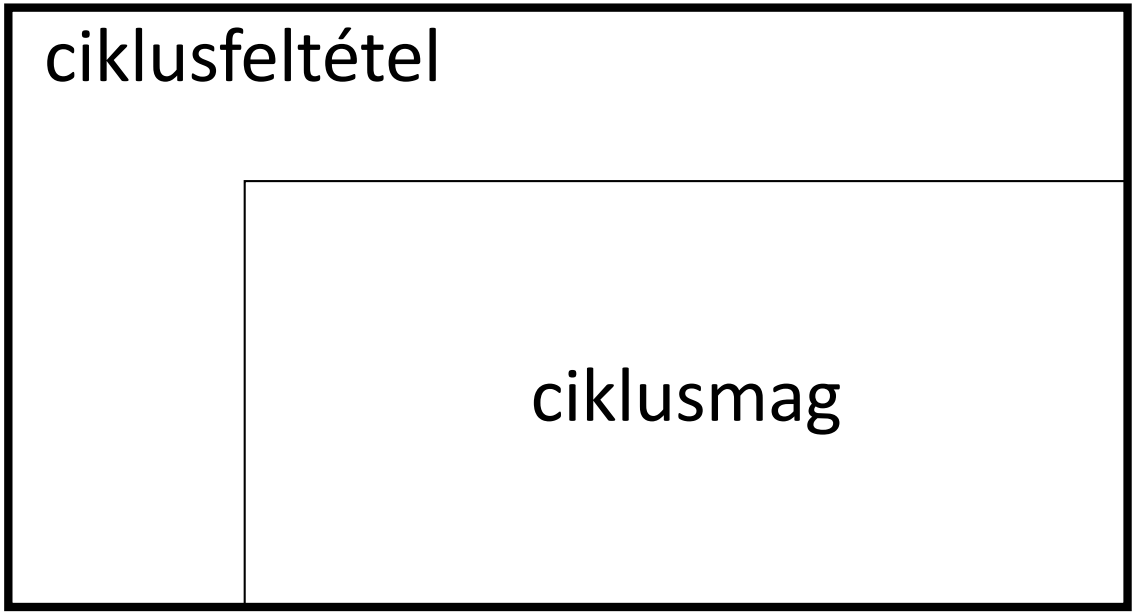
Lehetséges az értékre rész-megszorítást adni

```
switch vegetable {
  case "celery":
    let vegetableComment = …
  case "cucumber", "watercress":
    let vegetableComment = …"
  case let x where x.hasSuffix("pepper"):
    let vegetableComment = …
  default:
    let vegetableComment = …
}
```

Ciklusok

utasítások ismételt végrehajtása

- valamilyen feltételtől függően



Ciklusok általános kérdései

Vannak-e nem ismert lépésszámú ciklusok?

- Van-e elöl/hátul tesztelős ciklus?
- A ciklusfeltétel logikai érték kell legyen, vagy más típusú is lehet?

Kell-e blokkot kijelölni a ciklusmagnak?

Ciklusok általános kérdései

Van-e előre ismert lépésszámú ciklus?

- A ciklusváltozó mely jellemzője állítható be a következők közül?
  - alsó érték - felső érték - lépésszám
- Mi lehet a ciklusváltozó típusa?
- Biztosított-e a ciklusmagon belül a ciklusváltozó változtathatatlansága?
- Mi a ciklusváltozó hatásköre, definiált-e az értéke kilépéskor?

Ciklusok általános kérdései

Van-e általános (végtelen :-)) ciklus?

Léteznek-e a következő vezérlésátadó utasítások?

- break - continue

Van-e ciklusváltozó-iterátor?

Nem ismert lépésszámú ciklusok

Elöltesztelő “while” ciklusok:

- `while <kif> do <utasítás>`

Pascal: a ciklusmag egyetlen utasítás lehet (de blokk is)

- `while a < b do  b := b - a;`

ADA

end loop zárja, így a ciklusmag utasításssorozat is lehet:

```
while kifejezés loop
  ciklusmag
end loop
```

kifejezés Boolean típusú

C++

A kifejezés aritmetikai értéket ad

- nem 0: true, 0: false

A ciklusmag egyetlen utasítás lehet (de blokk is)

```
while (<expr> ) <statm>
```

```
while (a < b)
   b = b – a;
```

Java

Hasonló a C++-hoz,
kivéve: a kifejezés típusa boolean kell legyen.

```
i=0;
while (i<10) {
 System.out.println(i);
 i++;
}
```

Eiffel

Amíg a ciklusfeltétel hamis!

- end zárja, így a ciklusmag utasításssorozat is lehet.

```
from
   init
[invariant
   loop-inv
   variant
   variant-func]
until
   loop-cond
loop
   loop-body
end
```

Hátultesztelő “repeat-until” ciklusok

Amíg egy feltétel igaz nem lesz.

Ciklusmag egyszer biztosan lefut

```
repeat
  utasítássorozat
until ciklusfelt
```

nincs szükség blokk utasításra – a ciklusmag vége meghatározott

`while E do S`

- Jelentése

```
if E then
    repeat S until not E
```

Hátultesztelő ciklusok

Pascal:

- ```
  repeat
     b := b - a;
  until (b <= a);
  ```

C++: a kifejezés aritmetikai érték, nem 0: igaz, amíg értéke 0 (hamis) nem lesz

- `do <statm> while (<expr>);`

Java: hasonló a C++-hoz, kivéve: a kifejezés típusa boolean kell legyen

Eiffel:

- “until_do” az Iteration könyvtárból.
  ```

Előre ismert lépésszámú ciklusok “For” ciklusok

index változó kezelése – lépésköz - határ

Egyszer, a ciklusba való belépés előtt értékeli ki a lépésközt és a határt, vagy minden végrehajtás után újra?

A határt a ciklusmag végrehajtása előtt vagy után ellenőrzi?

Mi lehet a ciklusváltozó típusa?

Biztosított-e a ciklusmagon belül a ciklusváltozó változtathatatlansága?

Mi a ciklusváltozó hatásköre, definiált-e az értéke kilépéskor?

Pascal

lépésköz és határ: egyszer

határ ellenőrzése ciklusmag előtt

ciklusváltozó nem változtatható (ma már)

értéke kilépéskor nem definiált

ciklusváltozó diszkrét típusú lehet

```
for i := 1 to n do sum := sum + i;
for c := ‘z’ downto ‘a’ do write(c);
```

Ada

Az index a ciklus lokális konstansa. Diszkrét típusú kell legyen. A ciklus értékintervallumát csak egyszer, a ciklusba való belépés előtt számítja ki.

```
for <var> in loop-range loop
 <utasítások>;
end loop;
```

```
Sum:=0;
for I in 1..N loop
  Sum := Sum + I;
end loop;
```

A “reverse” hatására : fordított sorrend.

C++

a `for (for-init-stm [expr-1]; [expr-2]) stm` jelentése

```
{ for-init-stm
    while (expr-1) {
      stm
      expr-2;
    }
}
```

Kivéve, hogy egy `continue` a `stm`-ben `expr-2`-t hajt végre az `expr-1` kiértékelése előtt.

Hiányzó `expr-1` ekvivalens: `while(1)`

Ha a `for-init-stm` egy deklaráció, akkor a deklarált nevek hatásköre a `for` utasítást tartalmazó blokk

```
sum=0;
for (int i = 1; i < n; i++)
  sum = sum + i;
```

Java

Eredeti for ciklus: hasonló a C++-hoz:

```
public class ForDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 32, 87, 199, 622, 127, 485 };
        for (int i = 0; i < arrayOfInts.length; i++) {
            System.out.print(arrayOfInts[i] + " "); i++;
        }
        System.out.println();
    }
}
```

Mit csinál?

Java

for-each ciklus:

```
public class ForEachDemo {
  public static void main(String[] args) {
    int[] arrayOfInts = {32, 87, ...., 622, 127};
    for (int i : arrayOfInts) {
          System.out.print(i + " ");
  }
    System.out.println();
  }
}
```

Java

Gyűjteményekkel (Collection) és tömbökkel használható

Az iterációk leggyakoribb formájára, amikor az index, illetve az iterátor értéket semmilyen más műveletre nem használják, csak az elemek elérésére.

Még egy példa:

- …. ha van egy `Number` típusunk:

```
List<Number> szamok = new ArrayList<Number>();
    szamok.add(new Integer(42));
    szamok.add(new Integer(-30));
    szamok.add(new BigDecimal("654.2"));
    for ( Number number : szamok ){
      …
    }
```

C#

„hagyományos” for ciklus, hasonlóan a C++-hoz:

```
using System;
public class ForLoopTest
{
   public static void Main()
   {
      for (int i = 1; i <= 5; i++)
         Console.WriteLine(i);
   }
}
```

C#

foreach utasítás:

```
foreach (type identifier in expression) statement
```

- a ciklusváltozó értékét ne változtassuk, ha ez érték típusú, akkor nem is lehet.
- A kifejezés gyűjtemény vagy tömb lehet, `IEnumerable`-t implementál, vagy egy olyan típust, ami deklarál egy `GetEnumerator` metódust.

C#

Foreach tömbökre:

```
public static void Main() {
        int odd = 0, even = 0;
        int[] arr = new int [] {0,1,2,5,7,8,11};
        foreach (int i in arr) {
            if (i%2 == 0)
                even++;
            else
                odd++;
        }
        …
    }
}
```

C#

foreach gyűjteményekre

- foreach (ItemType item in myCollection)
- a myCollection gyűjteményre a következőknek kell teljesülnie:
  - interface, class, vagy struct típus kell legyen
  - kell legyen egy GetEnumerator nevű példánymetódusa, aminek a visszaadott típusára (pl. Enumerator) fennáll:
    - Van egy Current nevű property-je, ami ItemType-ot, vagy erre konvertálhatót ad vissza – a gyűjtemény aktuális elemét
    - Van egy MoveNext, bool-t visszaadó metódusa, ami növeli az elemszámlálót, és true-t ad, ha van még elem a gyűjteményben

C#

foreach gyűjteményekre – lehetőségek

- Létrehozunk egy gyűjteményt a fenti szabályokkal – ez csak C# programokban használható persze
- Létrehozunk egy gyűjteményt a fenti szabályokkal, és implementáljuk az IEnumerable interfészt
  - ez más nyelvekben is használható lesz, mint pl. a Visual Basic
- Használjuk az előredefiniált gyűjtemény osztályokat

C#

foreach gyűjteményekre – példák

```
using System;
public class MyCollection {
    int[] items;
    public MyCollection()  {
        items = new int[5] {12, 44, 33, 2, 50};
    }
    public MyEnumerator GetEnumerator()   {
        return new MyEnumerator(this);
    }
```

C#

foreach gyűjteményekre – példák

```
public class MyEnumerator   {
      int nIndex;
      MyCollection collection;
      public MyEnumerator(MyCollection coll)  {
         collection = coll;
         nIndex = -1;
      }
      public bool MoveNext()  {
         nIndex++;
         return(nIndex < collection.items.GetLength(0));
      }
     public int Current {
         get {
            return(collection.items[nIndex]);
         }
      }
   }
}
```

C#

foreach gyűjteményekre – példák

```
public class MainClass
{
  public static void Main()
  {
     MyCollection col = new MyCollection();
     Console.WriteLine("Values in the collection are:");

     // Display collection items:
     foreach (int i in col)
     {
        Console.WriteLine(i);
     }
  }
}
```

SWIFT

For ciklus

- ```
  var firstForLoop = 0
  for i in 0..<4 {
      firstForLoop += i
  }
  ```
- ```
  var secondForLoop = 0
  for var i = 0; i < 4; ++i {
      secondForLoop += i
  }
  ```

A `..<` használható arra, hogy az `i` `0` és `3` közötti értékeket veszi fel, ha `0` és `4` közötti értékeket szeretnénk, akkor `i in 0 … 4` a használandó szintaktika

“Végtelen” ciklusok

ADA

```
loop
  <statm>1; …
     exit when <feltétel>;
  <statm>2; …
end loop;

loop
  get(Current_Char);
  exit when Current_Char =’*’;
end loop;
```

Vezérlésátadó utasítások

`break`: kiugrás a vezérlési szerkezetből - pl.:

```
while  feltétel do
  if speciális-eset then kezeld le; break; end if;
 kezeld a normális eseteket;
end while
```

`continue`: ciklusfeltétel újrakiértékelése

alprogram hívás

`return` alprogramból hívóhoz visszatérés.

`goto` – !!! Veszélyeket rejt

Példák

C++:

- `break` - ciklusban, vagy switch utasításban
- `continue` - csak ciklusban
- `return`
- `goto`

Java:

- `break`, continue, `return` mint a C++-ban
- NINCS goto!

Absztrakció

Absztrakció – Mi az?

Gondolkodásmód: az általános elvekre koncentrálunk, nem ezeknek a specifikus megjelenési formáira.

- Filozófia,
- Matematika,
- Számítástudomány

Absztrakció a rendszer analízisben:

- A feladat lényeges aspektusai
- Eltekintünk a kevésbé lényegestől
- Például: légi közlekedés vezérlése
  - lényeges: helyzete, sebessége stb.
  - lényegtelen: színe, utasok neve…
    - (ez a feladattól függ!)

Alprogramok mint absztrakciók

Absztrakció a programozásban

- különböztessük meg a szinteket
  - Mit csinál a program?
  - Hogyan van implementálva?

Minden programozási nyelv a gépi kód absztrakciója.

Magasabb szintű absztrakciók

- Eljárás: Mit csinál az eljárás?
  - Hogyan csak akkor fontos, amikor implementáljuk ☺.
- Eljárást hívó eljárások:
  - Az absztrakció akárhány szükséges szintje bevezethető.

Alprogramok mint absztrakciók

Absztrakciós mechanizmus:

- a programozási nyelvi konstrukció, ami megengedi, hogy a programozó megragadja az absztrakciót
  - És a program részeként reprezentálja – egyfajta számítási mintaként
  - Egyszerű példa:
    - Eljárások és függvények
  - Egyéb példák
    - Később...

A legegyszerűbb absztrakciós mechanizmus

Eljárások és függvények: egységek, melyek számításokat tartalmaznak

- Függvény-absztrakció: egy kiszámítandó kifejezést tartalmaz
- Eljárás-absztrakció: egy végrehajtandó parancsot tartalmaz.

A tartalmazott számítás mindig végre lesz hajtva, amikor egy absztrakciót hívunk.

Az érdekeltségek szétválasztása:

- A Hívó azzal törődik, mit csinál a számítás.
- Az Implementáló azzal is törődik, hogy hogyan kell a számítást végrehajtani természetesen a „mit” szerint ☺

A hatékonyságot a paraméterezéssel javítjuk.

Függvény-absztrakció

Egy kiértékelendő kifejezés – amikor hívjuk, egy értéket ad vissza

Ada példa

```
function Kerulet (R : Float) return Float is
begin
    return 2.0*R*3.14;
end;
```

ML példa

```
fun power( x: real; n: int) =
    (* felt. hogy  n > 0 *)
    if n = 1 then
        x
    else
        x * power (x, n- 1)
    end
```

Mi a függvény definíció?

```
function I (FP1 ; ... ; FPn) return T is
body
```

Egy I azonosítót hozzákapcsol egy adott függvény-absztrakcióhoz.

A függvény-absztrakció a megfelelő aktuális paraméterekkel való híváskor eredményt ad vissza.

- A függvényhívás felhasználói szemlélettel egy leképezés az argumentumok és az eredmény között.
- Megvalósítói szemlélet: a függvénytörzs kiértékelése.
  - Az algoritmus változása csak a megvalósítóra tartozik.

Eljárás-absztrakció

Egy végrehajtandó parancsot testesít meg.

- A felhasználó csak a változók megváltozását érzékeli.

Sok nyelvben nem lehet létrehozni egy eljárás-absztrakciót név nélkül. Az általános formátum:

- `procedure I ( FP 1 ; ... ; FP n )`
  `body`

Felhasználói szemlélet:

- `type Dictionary = array [...] of Word;`
- `procedure sort( var words: Dictionary);`
- `...`
- `sort(a); (* érzékeljük ‘a’ változását *)`

Megvalósítói szemlélet: a kódolt algoritmus.

Absztrakciós elv

Összefoglalás:

- Függvény-absztrakció: egy kifejezés absztrakciója.
  - Egy függvényhívás egy kifejezés, amikor hívjuk, kiértékeli, és egy értéket ad vissza.
- Eljárás-absztrakció: egy parancs absztrakciója.
  - Egy eljáráshívás egy parancs, ami az eljárástörzs végrehajtása során frissíti/frissítheti a változók értékét.

Absztrakciós elv

Általánosítás

- Tetszőleges szintaktikai osztály fölött létrehozhatunk absztrakciókat, feltéve, hogy ez valamifajta számítást specifikál.
  - Absztrakt adattípusok
  - Generic (később)

Alprogramok és paramétereik

Alprogramok használata

- az egyik legelső programozási eszköz.

Charles Babbage – Analytical Engine – 1840-ben már tervezte, hogy lyukkártyák egy csoportját fogja használni nagyobb számítások gyakrabban ismételt részeinél

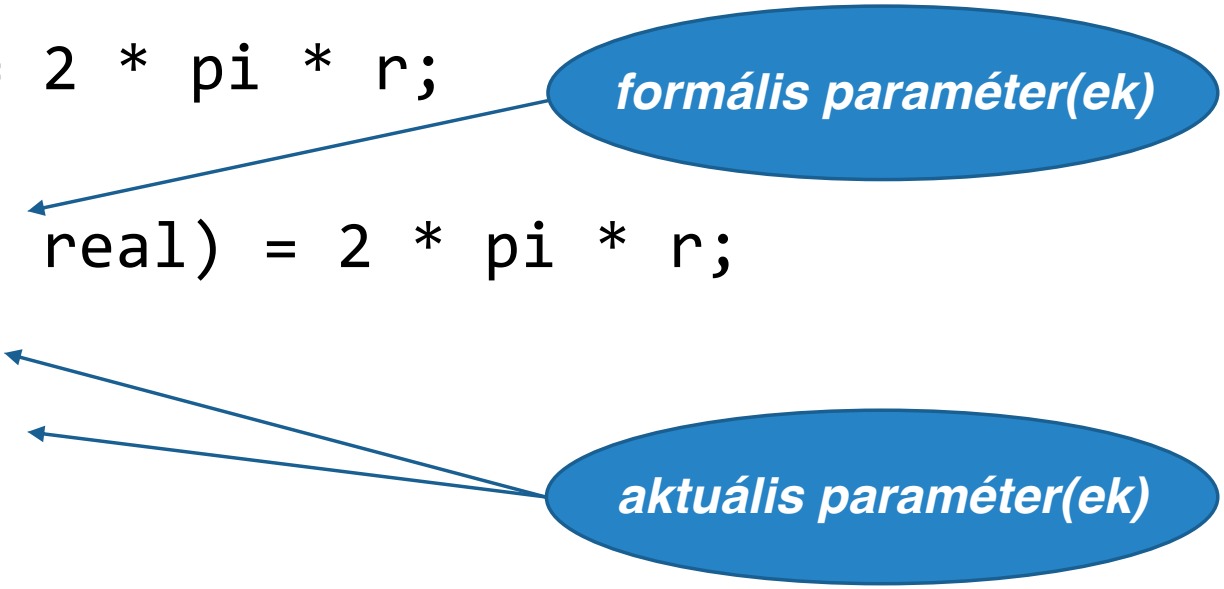
Alprogramok használatával nevet adhatunk egy kódrészletnek és paraméterezhetjük a viselkedését.

Paraméter átadása

Absztrakció általánosítása

```
val pi = 3.14159;
val r = 1.0;
fun circumference() = 2 * pi * r;

fun circumference (r: real) = 2 * pi * r;
circumference (1.0);
circumference(a + b);
```



Alprogram

Programegység

Végrehajtás kezdeményezése: meghívással

A program darabolásának eszköze

Különböző számítások elkülönítése egymástól

Újrafelhasználhatóság

Mire való

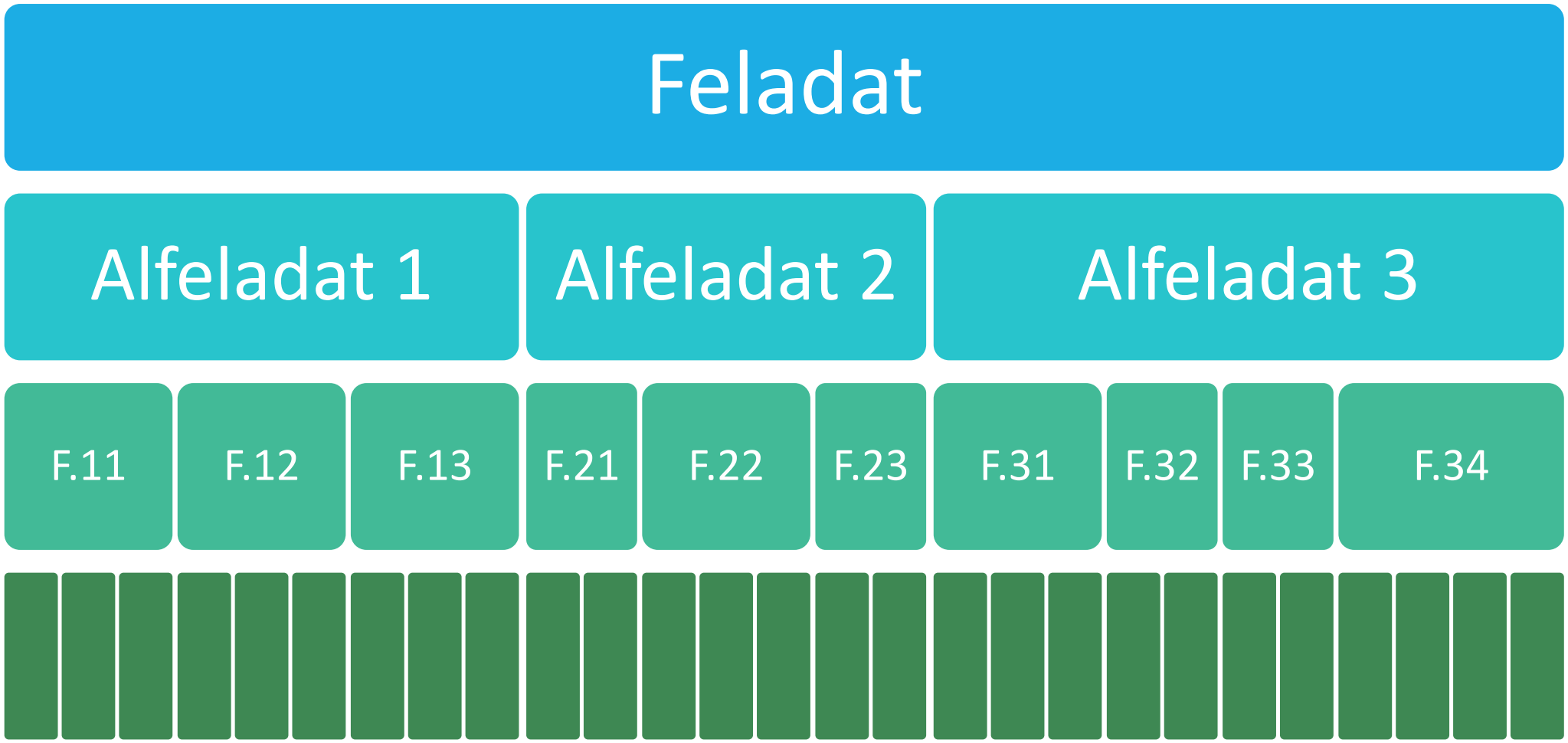
Nagyobb feladat egy részfeladatának megoldása

Algoritmus kódolása

Típusművelet megvalósítása

Matematikai függvény kiszámítása

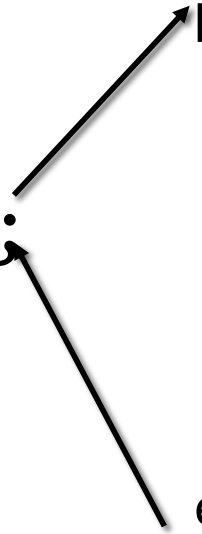
Eljárás-absztrakció



Alprogram hívása

```
Kar: Char;
…
begin
…
  while Meret<> 0 do
    begin
      KarOlvas (Kar);
      Teglalap;
      …
    end;
  end;
```

```
procedure Karolvas(Var C:Char);
 begin
  GotoXY(20,112);
  write('Kerem a karaktert');
  ReadLn(C);
end;
```



Alprogram

alprogram=(név, paraméterek, környezet, törzs).

- az alprogram definíciójakor a formális paraméterekkel írjuk le az adatcsere elemeit
  - a külvilággal való kommunikáció alterének komponenseit
- az alprogram használatakor pedig az aktuális paraméterek kerülnek ezek helyére
  - ez a paraméterátadás

Függvény

```
function Faktoriális ( N: Natural ) return
Positive is
    Fakt: Positive := 1;
begin
    for I in 1..N loop
        Fakt := Fakt * I;
    end loop;
    return Fakt;
end Faktoriális;
```

Eljárás

```
procedure Cserél ( A, B: in out Integer ) is
    Temp: Integer := A;
begin
    A := B;
    B := Temp;
end Cserél;
```

Eljárás – függvény

Eljárás végrehajtása: utasítás

Eljárás neve: ige

- `Egyenest_Rajzol(Kezdopont,Vegpont);`

Függvény végrehajtása: kifejezés értékének meghatározása.

Függvény neve: főnév vagy melléknév.

- `if Elemek_Száma(Halmaz) > 0 then ...`

Paraméterek, visszatérési érték

Információ átadása / átvétele alprogramhívásnál

- paramétereken keresztül
- visszatérési értéken keresztül

Paramétereknél az információ áramlása

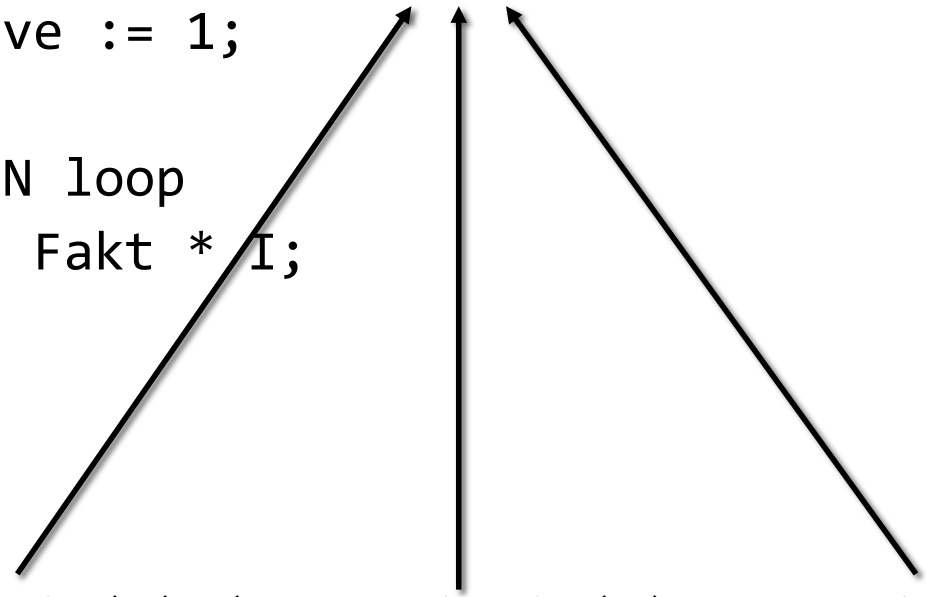
- Merre: a paraméterek módja
- Hogyan: a paraméterátadás technikája (paraméterátadás módja)

Alprogram hívásakor a formális paramétereknek aktuális paramétereket feleltetünk meg.

Paraméterek

```
function Faktoriális ( N: Natural ) return Positive is
    Fakt: Positive := 1;
begin
    for I in 1..N loop
         Fakt := Fakt * I;
    end loop;
    return Fakt;
end Faktoriális;

L_Al_K:=Faktoriális(L)/(Faktoriális(K)*Faktoriális(L-K))
```

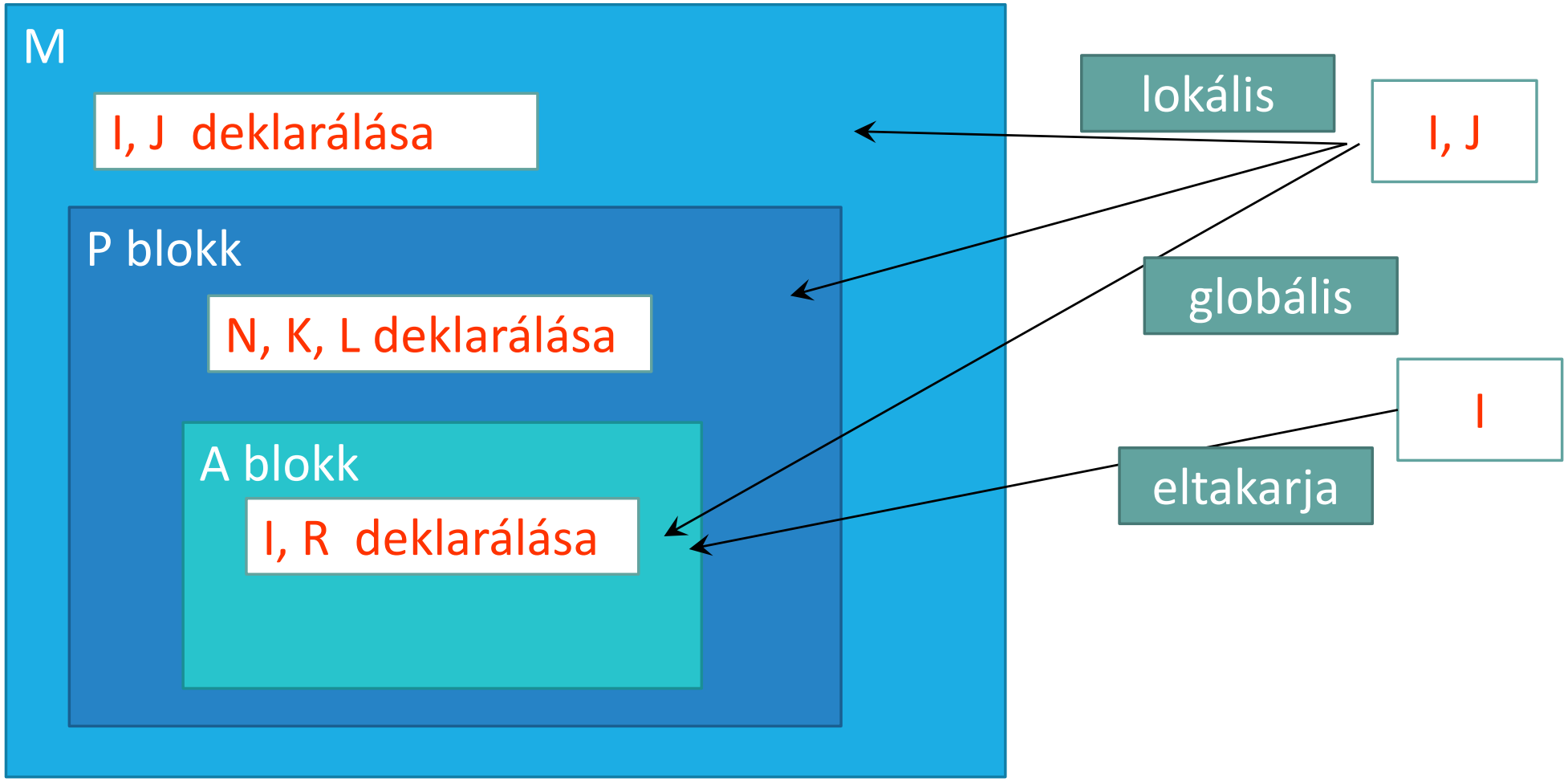


Alprogram beágyazható

Sok programozási nyelvben:

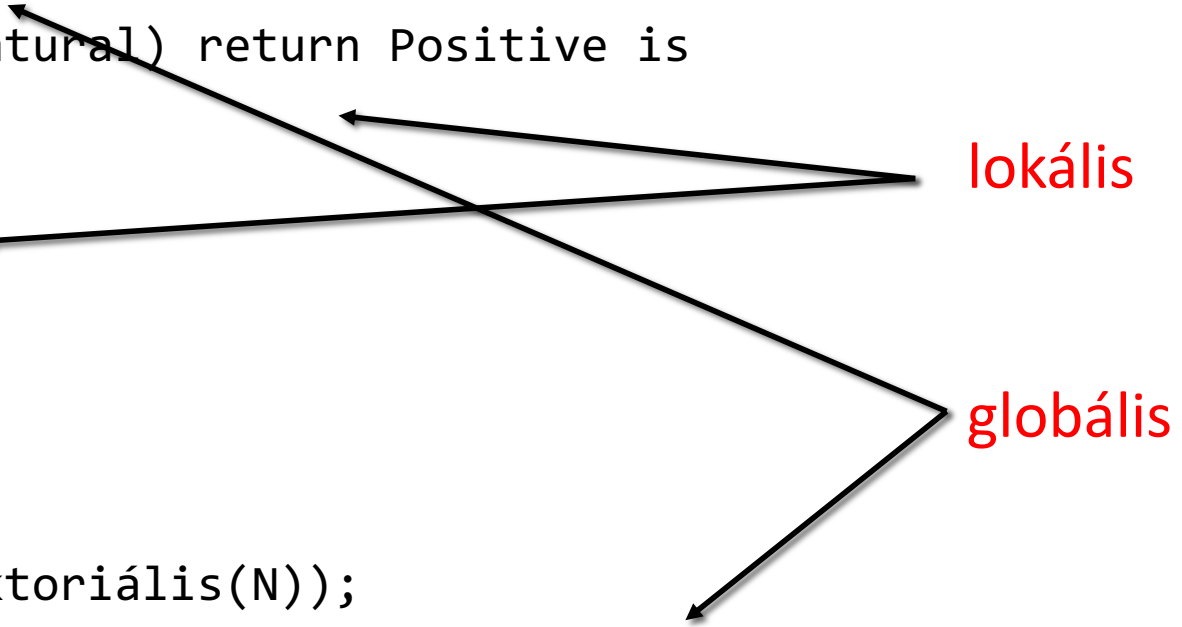
- Blokkszerűen
- Deklarációs részbe
- Lokális - globális deklarációk
- Hatáskör, láthatóság, élettartam

Változók


M
I, J deklarálása
P blokk
N, K, L deklarálása
A blokk
I, R deklarálása
lokális
I, J
globális
I
eltakarja

Példa

```
with Ada.Integer_Text_IO, Text_IO;
procedure Faktoriálist_Számít is
   N: Integer;
   function Faktoriális (N: Natural) return Positive is
      Fakt: Positive := 1;
   begin
      for I in 1..N loop
          Fakt := Fakt * I;
      end loop;
      return Fakt;
   end Faktoriális;
begin
   ...
   Ada.Integer_Text_IO.Put(Faktoriális(N));
   ...
end Faktoriálist_Számít;
```



Paraméterek fajtái

Az információáramlás iránya szerint

- Input: hívó $\Rightarrow$ alprogram
- Output: hívó $\Leftarrow$ alprogram
- Update: hívó $\Leftrightarrow$ alprogram

Paraméterátadási technikák

A paraméterátadás módja

- Különféle nyelvekben különféle módon adják át a paramétereket
- Legismertebb paraméterátadási módok:
  - érték szerint
  - cím szerint
  - eredmény, érték-eredmény szerint
  - név szerint

Érték szerinti paraméterátadás

A formális paraméter az alprogram egy lokális változója, aminek az aktuális paraméter adja a kezdőértéket

- in módú átadásra alkalmas

Az aktuális paraméter tetszőleges kifejezés, kiértékelésére egyszer, az alprogram végrehajtásának megkezdése előtt kerül sor.

Az alprogram végrehajtása közben sem az aktuális paraméter értékének esetleges megváltozása nincs hatással a formális paraméter értékére, sem a formális paraméter értékének megváltoztatása nincs hatással az aktuális paraméterre.

Érték szerinti paraméterátadás

Példa C-ben

```
int lnko( int a, int b ){
  while( a != b )
    if( a > b )   a -= b;
    else          b -= a;
  return a;
}
```

```
int x,y,z; ….
x = 10; y = 5;
z = lnko(x,x+y+1);


a = 10
b = 16
```

Érték szerinti paraméterátadás

Példa C-ben

```
int lnko( int a, int b ){
  while( a != b )
    if( a > b )   a -= b;
    else          b -= a;
  return a;
}
```

```
int x,y,z; ….
x = 10; y = 5;
z = lnko(x,x+y+1);
```

z értéke 2 lesz,
x marad 10

Érték szerinti paraméterátadás

Csak egyszer értékelődik ki

C példa:

```
int x = 1;
int f( int a ){
  ++x;
  return a+x;
}
int main(){
  int y = f(x);
}
```

a = 1

x= 2

y = 3 !

Cím szerinti paraméterátadás

Az aktuális paraméter vagy egy változó, vagy egy változó komponensét meghatározó kifejezés – balérték - (x[i+2*j]) lehet.

Az aktuális paraméter kiértékelése a hozzá rendelt memóriaterület címének meghatározását jelenti.

Az aktuális paraméter kiértékelésére az alprogram végrehajtásának megkezdése előtt kerül sor, s az aktuális paraméter memóriaterülete rendelődik hozzá.

Az alprogramban hivatkozhatunk is a formális paraméter értékére, és adhatunk is neki új értéket (update módú átadásra is alkalmas)

Cím szerinti paraméterátadás

Példa Pascalban:

```
procedure Csere(var a, b:
Integer );
var temp: Integer;
begin
  temp := a;
  a := b;
  b := temp;
end;
```

Hívás:

```
…
x: Integer;
y: Integer;
…
x:=3;
y:=6;
Csere(x,y);
```

x=6, y=3

Cím szerinti paraméterátadás

```
procedure s (var a   : integer)…
     -- kiírja és lenullázza a paraméterét

procedure p (var a, b: integer);
     -- kiírja és lenullázza a paramétereit

begin
  s(a);
  s(b);
end;
```

Mit csinál majd `p(x,x)`?

Cím szerinti paraméterátadás

„Alias” – amikor a program egy pontján két különböző változó ugyanazt az egyedet jelenti

```
var  globalis: Integer;
procedure r (var lokalis: Integer);
  begin
    globalis := globalis + 1;
    lokalis := lokalis + globalis;
end r;
... globalis := 1;
r(globalis);
```

Mennyi lesz `globalis` értéke?

Eredmény szerinti paraméterátadás

A kimenő paraméterek megvalósításához.

- Az alprogram formális paraméterében kiszámított eredményt helyezi vissza az aktuális paraméterbe.

A formális paraméter, csakúgy, mint az érték szerinti paraméterátadás esetén, az alprogram lokális változója, melynek értéke az alprogram befejeződésekor kimásolódik az aktuális paraméterbe.

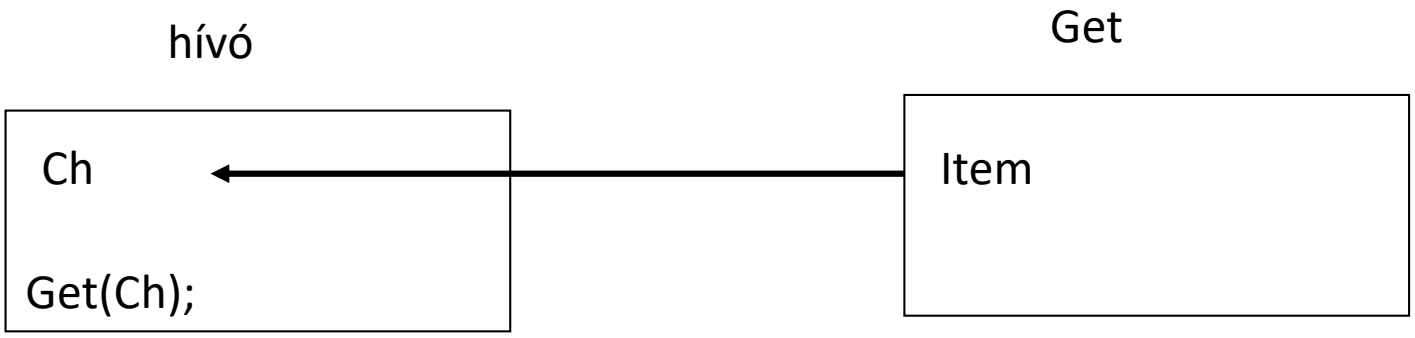
Az aktuális paraméter balérték kell legyen.

A formális paraméter nem kapja meg az aktuális paraméter értékét az alprogram hívásakor, ezért az információáramlás egyirányú (out módú átadásra alkalmas).

Eredmény szerinti paraméterátadás

Ada példa:

- `procedure Get(Item: out Character );`



Object Pascal példa:

- `procedure GetInfo(out Info: SomeRecordType);`

Érték/Eredmény szerinti paraméterátadás

Az alprogram befejezésének pillanatáig megegyezik az érték szerinti paraméterátadással.

Az alprogram végrehajtásának befejezésekor az aktuális paraméter felveszi a formális paraméter pillanatnyi értékét (update módú átadásra alkalmas).

Az aktuális paraméter csak „balérték” lehet

Miben különbözik a cím szerinti átadástól?

Érték/Eredmény szerinti paraméterátadás

```
procedure Paramproba is
…   A,B:Integer;….
  procedure Parcsere(X,Y: in out Integer) is
      Temp:Integer;                                  <-------  X=1,Y=2
   begin
      Temp:=X; X:=Y; Y:=Temp;
      A:=12; B:=99; -- …                             <-------  X=2,Y=1
   end;
begin
   A:=1;B:=2;
   Parcsere(A,B);                                    <-------  A=2,B=1
end;
```

Érték/Eredmény szerinti paraméterátadás

```
procedure s (a:    in out Integer)…
      – kiírja és lenullázza a paraméterét
procedure p (a, b: in out Integer)
      - kiírja és lenullázza a paramétereit
begin
  s(a);
  s(b);
end;
```

Mit csinál majd `p(x,x)`?

Érték/Eredmény szerinti paraméterátadás

```
globalis: Integer;
procedure r (lokalis: in out Integer);
  begin
    globalis := globalis + 1;
    lokalis := lokalis + globalis;
end r;
...
globalis := 1;
r(globalis);
```

itt mennyi lesz a `globalis`?

Név szerinti paraméterátadás

Az aktuális paraméterként leírt teljes kifejezés adódik át és minden használatkor (dinamikusan!) kiértékelődik;

- például az a[i] hivatkozás változhat, ha menet közben az i értéke változik(!)
- (update módú átadásra is alkalmas)

Lusta kiértékelés…

Példa

```
program
  I: integer;
  A: integer array;
  SWAP_BY_NAME: procedure(X: name, Y: name)
   TEMP: integer;
   begin
    TEMP := X;  X := Y;   Y := TEMP;
  end;
 begin
  I := 3;
  A[I] := 6;
  output I, A[3];
  SWAP_BY_NAME(I, A[I]);
  output I, A[3];
 end;
```

Output

```
I = 3   A[3]  = 6
I = 6   A[3]  = 6 de A[6] = 3 lesz!
```

Egy híres példa

Jensen's device kifejezések kiértékelésére – Algol60

```
real procedure sum (expr, i, low, high);
   value low, high;
   real expr;        -- név szerint az expr és az i,
                     -- ez a default Algolban!
    integer i, low, high;
    begin
     real rtn;
     rtn := 0;
     for i := low step 1 until high do
        rtn := rtn + expr;
     sum := rtn;
end sum
y := sum (3*x*x - 5*x + 2, x, 1, 10);
```

Mit csinál?

Alprogramok paraméterei

Az alprogramok paramétereinek száma általában kötött, de lehet változó is.

Egy szintig szimulálható a változó számú paraméter egy tömb átadásával, de ezzel nem mindig oldható meg eltérő típusú paraméterek átadása.

Paraméterek száma

A programnyelvek kezelhetik a túl sok vagy túl kevés aktuális paraméter megadását.

A túl kevés paraméter megadása esetén több módszer alkalmazható:

- alapértelmezett értékek adhatók meg, amelyek a hiányzó aktuális paraméterek helyébe lépnek (pozíció vagy név szerint)
- vesszőket kell tenni a kihagyott paraméterek helyett
- amíg nincsenek alapértelmezett értékek, addig kötelező az aktuális paramétereket megadni, tehát az alapértelmezések csak a paraméterlista végén lehetnek

Formális-aktuális paraméterek megfeleltetése

```
procedure Get_Line ( File : in  File_Type;
          Item : out String; Last : out Natural )
```

pozícionális formában: az aktuális paramétereket abban a sorrendben kell felsorolni, ahogy a formális paraméterek voltak az alprogram specifikációjában:

```
Get_Line(F,S,N);
```

névvel jelölt formában: a paraméterek sorrendje tetszőleges:

```
Get_Line(File=>F, Last=>L, Item=>S);
```

kevert formában: mindig a pozícionálisan megadott paramétereknek kell elöl állniuk:

```
Get_Line(F,Last=>L,Item=>S);
```

Paraméterek feltételezett értéke

Az in módú paraméterekhez rendelhetünk feltételezett bemenő értéket.

Az alprogram specifikációs része tartalmazza ezt a feltételezett értéket, amit akkor vesz fel a formális paraméter, ha neki megfelelő aktuális paramétert nem adunk meg.

Példa

```
procedure New_Line (File : in File_Type;
                    Spacing : in Positive_Count := 1 )
```

Hívhatjuk úgy, hogy megadjuk a Spacing értékét, de úgy is, hogy nem: ilyenkor a feltételezett értéket használja az eljárás.

```
New_Line(F);       -- 1 sort emel az F fájlban
New_Line(F,1);     -- az előzővel egyenértékű
New_Line(F,42);    -- 42 sort emel
```

Paraméterek feltételezett értéke

Lehet több alapértelmezett értékkel rendelkező paraméter is. Ilyenkor bármelyik aktuális paramétert el lehet hagyni, akár többet is (segít a névvel jelölt hívás).

```
procedure Egyenest_Rajzol (
                Kezdőpont, Végpont: in Pont;
                Vastagság: in Positive := 1;
                Szín: in Színskála := Fekete )
Egyenest_Rajzol(P1,P2);
Egyenest_Rajzol(P1,P2,3);
Egyenest_Rajzol(P1,P2,Szín=>Piros);
```

Alprogramok túlterhelése (átlapolása)

```
procedure Cserél ( A, B: in out Integer );

procedure Cserél ( A, B: in out Boolean );
```

Azonos név – paraméterek száma és/vagy típusa különböző

A használatból (a hívásból) fog kiderülni, - kell kiderüljön! - hogy melyikre gondolunk

Kérdések

Mellékhatás – mi az?

```
z:= sin(x);
y:= sin(x);
z = y?
```

mi az elvárásom?

```
zz:= rnd();
xx:= rnd();
zz = xx
```

és most?

Kérdések

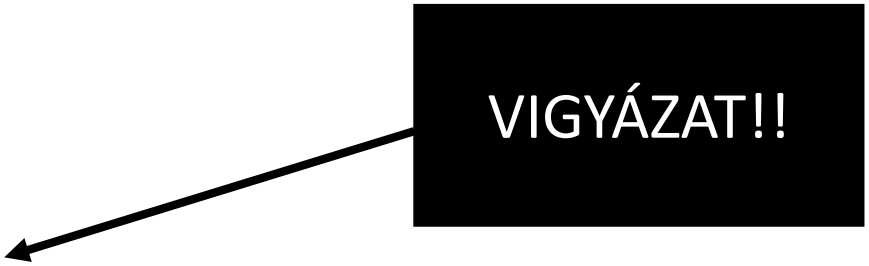
Mellékhatás 1.: globális változók használata

```
with Text_Io; use Text_Io;
procedure Globprobe is
     I:Integer;
     function Glob(J:Integer) return Integer is
     begin
          I:=I+1;  return I+J;
     end;
begin
     I:=2;
     Put_Line(Integer'Image(Glob(1)));
                  …
     Put_Line(Integer'Image(Glob(1)));
end;
```

Kérdések

Mellékhatás 2.: függvény paraméter is változik

```
int f(int val, int& ref){
   val++;
   ref++;
   return val+ref;
};
void main(){
   int i=1;
   int j=1;
   int k;
   k=f(i,j);
}
```



Operátorok

Infix alakban is írható műveletek

```
A+42        ”+”(A,42)
```

Operátorok is átlapolhatók:

```
function ”+” ( A, B: Mátrix ) return Mátrix;
```

Rekurzió

Közvetlenül vagy közvetve önmagát hívó alprogram

```
function Faktoriális ( N: Natural ) return Positive
is
begin
    if N > 1 then
            return N * Faktoriális(N-1);
    else return 1;
    end if;
end Faktoriális;
```

Rekurzió

A rekurzív alprogramok paraméterátadás szempontjából nem különböznek a szokásos alprogramoktól.

Ezért a paramétereket rekurzív alprogramoknak cím szerint átadni csak elővigyázatosan szabad, mert a sorozatos hívások interferálhatnak.

Fontosabb kérdések

Van-e eljárás?

Van-e függvény?

Mely paraméterátadási módok léteznek?

- érték szerinti
- eredmény szerinti
- érték-eredmény szerinti
- cím szerinti / konstans cím szerinti
- név szerinti

További fontos kérdések

Meghatározható-e a paraméterek információátadásának iránya?

Adható-e a formális paramétereknek alapértelmezett érték?

Túlterhelhetők-e az alprogramnevek?

Az operátorok átlapolhatók-e?

Definiálhatók-e új operátorok?

Típus-e az alprogram? Lehet-e alprogrammal paraméterezni?

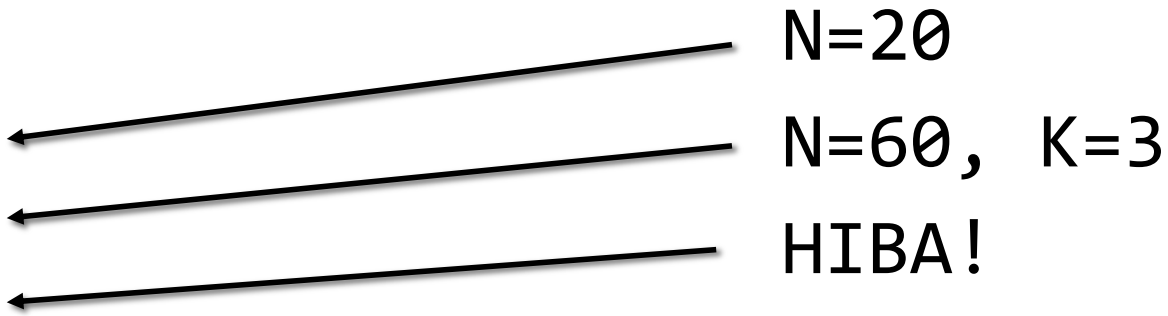
Pascal

A paraméterek alapértelmezésben érték szerint adódnak át, de a VAR kulcsszóval cím szerinti átadás érhető el:

```
program A;
   procedure Megszoroz(Var Mit: Integer;Szorzo:Integer);
      begin
        Mit:= Mit * Szorzo;
   end;

Var N, K :Integer;

begin
   N:=5; K:=3;
   Megszoroz(N, 4);
   Megszoroz(N, K);
   Megszoroz(5, N);
end.
```



C/C++

csak függvény van – void visszatérési érték, ha eljárást szeretnénk

csak érték szerinti paraméter átvétel,
cím szerinti: mutatók vagy referencia kell.
Lehet a paraméterek számára is kezdőértéket adni.

```
void f(int val, int& ref){
  val++;
  ref++;
}
```

```
void g(){
  int i=1;
  int j=1;
  f(i,j);
}
```

i==1,
j==2 lesz

C++

const& paraméter – nem változtatható a törzsön belül

```
float fsqrt(const float&);
...
float r=fsqrt(x);
```

Tömb paraméterek

- a tömb első elemére hivatkozó mutató adódik át
- vagyis nem érték szerint adódnak át
- méretük nem adódik át

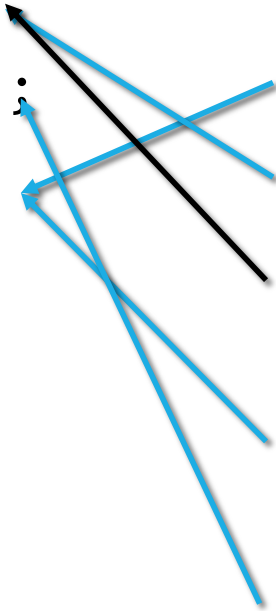
```
int strlen(const char*);
void f(){
  char v[]='egy tomb';
  int i=strlen(v);
}
```

A T[] típusú paraméter T* típusúvá lesz átalakítva ekkor.

C++

Túlterhelés lehetséges:

```
void print(int);
void print(const char*);
void print(char);
…
```



```
void h(char c, short s, int i){
 print(c); //Pontos illeszkedés
 print(i); //Pontos illeszkedés
 print(s); //Konverzió

 print(‚a’);//Pontos illeszkedés

 print(„a”);
           //Pontos illeszkedés
}
```

Java

Csak valamely osztály metódusa lehet

Az aktuális objektum attribútumaira hivatkozik

Paraméterátadás

- primitív típusok érték szerint
- összetett típusú paraméterek referencia szerint
  - mutable-immutable lehetőség!

Smalltalk

Üzenetküldésekkel érjük el a metódusokat, így a paraméterátadás érték szerintinek tekinthető – referenciák figyelembevételével

- A következő sor az a objektumnak küld üzenetet, két paraméterrel a két paraméter megnevezésével és aktuális értékével
- `a at:3 put:5`

Ada

`in, out, in out`

Az összetett típusú értékek esetén a fordítóprogram választhat az érték-eredmény, illetve a cím szerinti átadás között.

A függvényeknek csak in paramétereik lehetnek.

Az in paramétereknek lehet alapértelmezett értékük is.

Az alprogramok határozatlan méretű tömb típust is elfogadnak a formális paramétereik között.

SWIFT

Konstans paraméter, inout paraméter, kevert formában történő átadás, feltételezett érték létezik, függvényt is át lehet adni paraméterként, változó számú paraméter átadásának lehetősége

```
func greet(name: String, day: String) -> String {
    …
}
```

Több visszatérési értéke is lehet:

```
func calculateStatistics(scores: [Int]) ->
    (min: Int, max: Int, sum: Int ) {
  …
  return (min, max, sum)
}
```