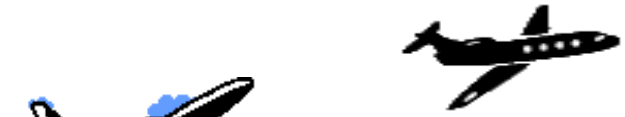
Absztrakció

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 lénye**
  - **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?
- Hogy 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.

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.

Pl. Ada:

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

vagy ML:

```
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* (*FP 1* ; ... ; *FP n*) return T **is** *E*

- 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*) *B*

- 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

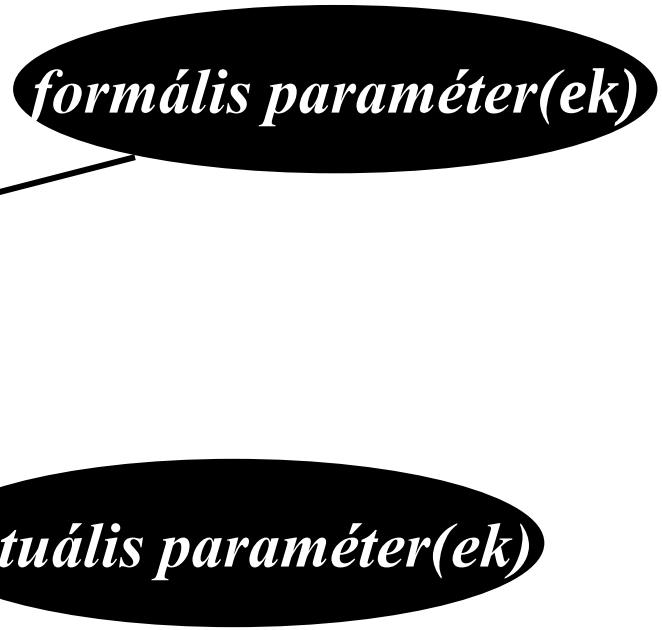
Később

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.

Paraméterek adása: absztrakció általánosítása

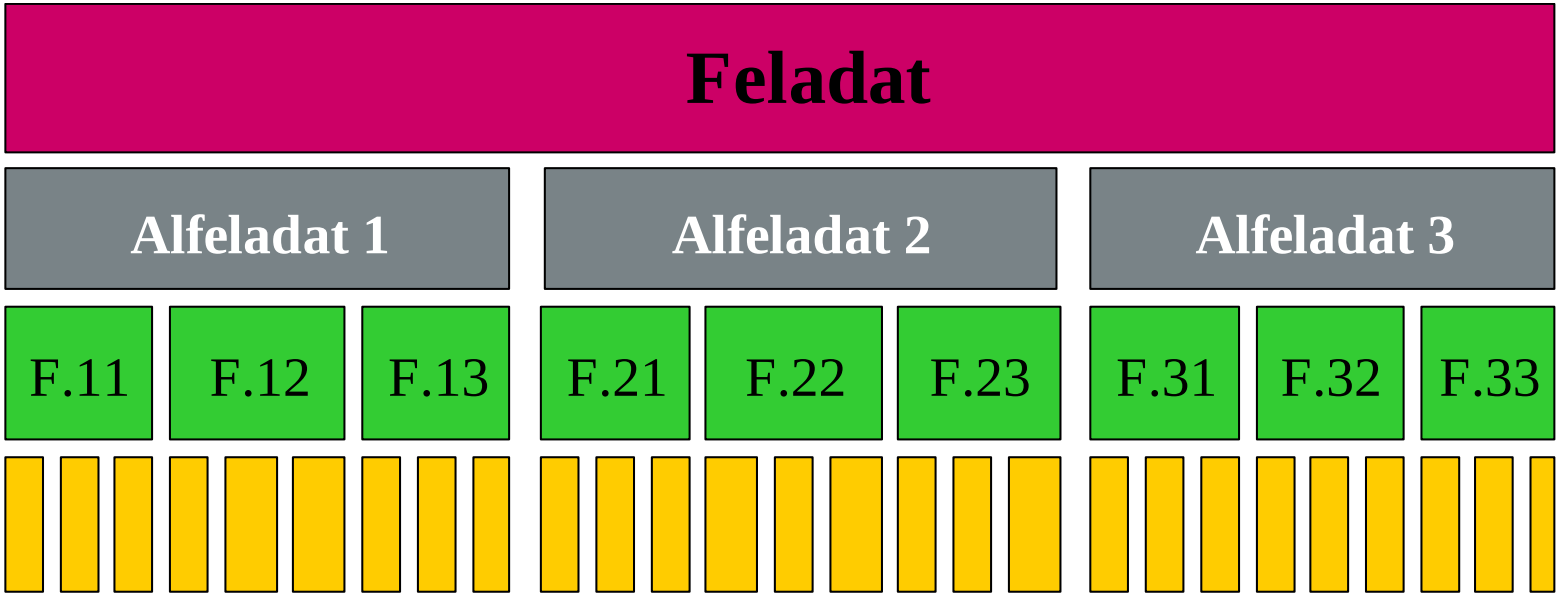
```
val pi = 3.14159;
val r = 1.0;
fun perimeter() = 2 * pi * r;

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



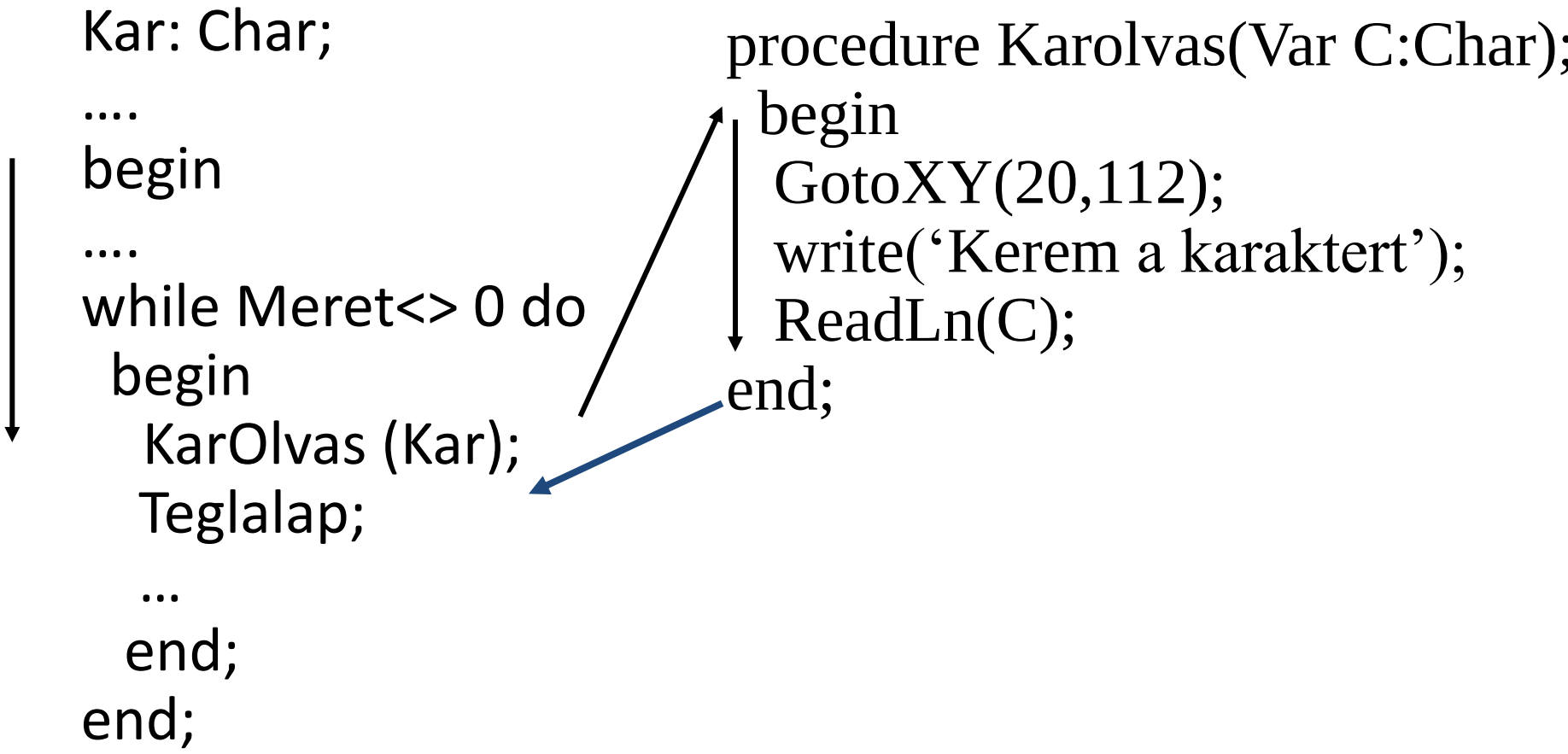
Alprogram

- Programegység
- Végrehajtás kezdeményezése: meghívással
- A program darabolásának eszköze
- Különböző számítások elkülönítése egymástól
- Újrafelhasználhatóság

Eljárás-absztrakció



Alprogram hívása:



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

$\Rightarrow$ 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 végrehajtása: utasítás.
  Eljárás neve: ige.

  Egyenest_Rajzol(Kezdopont,Vegpont);

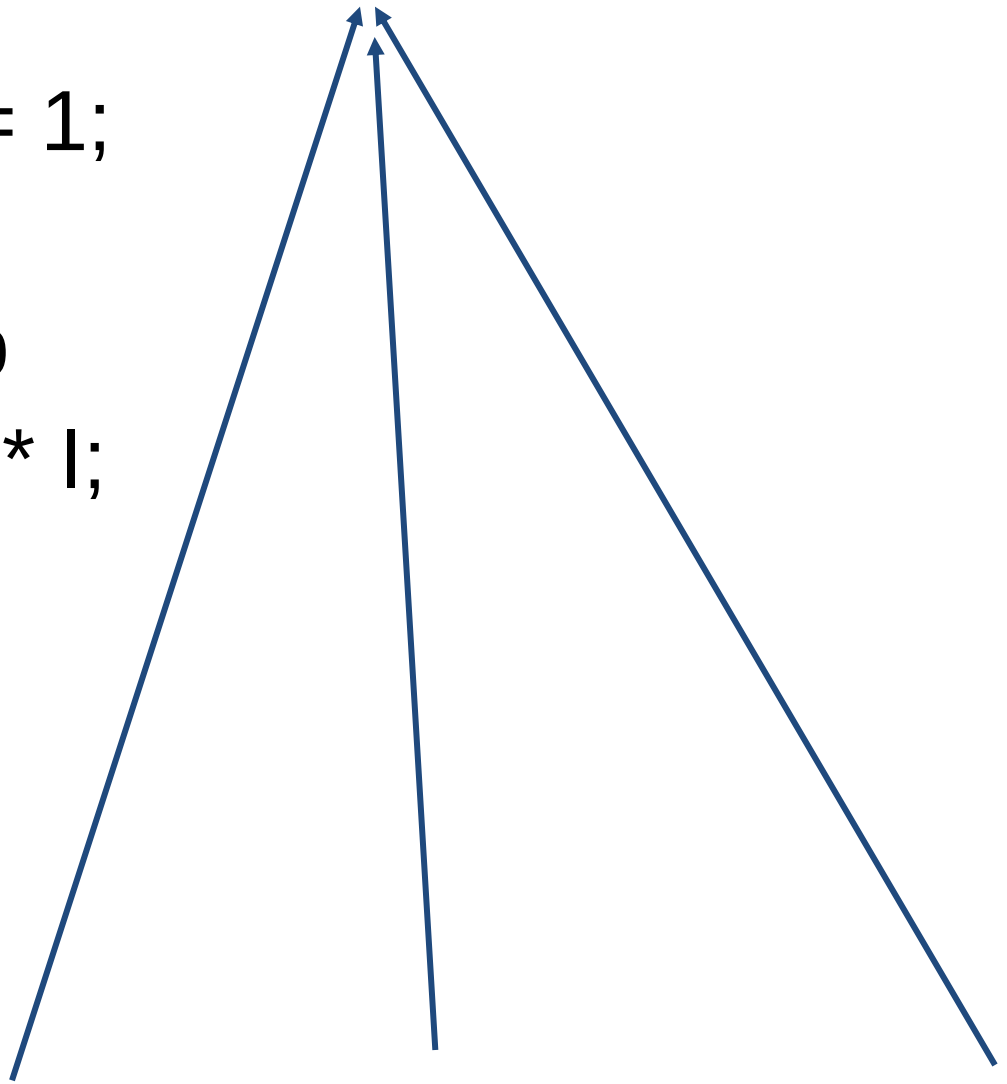
- Függvény végrehajtása:
  kifejezés értékének meghatározása.
  Függvény neve: főnév vagy melléknév.

  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.

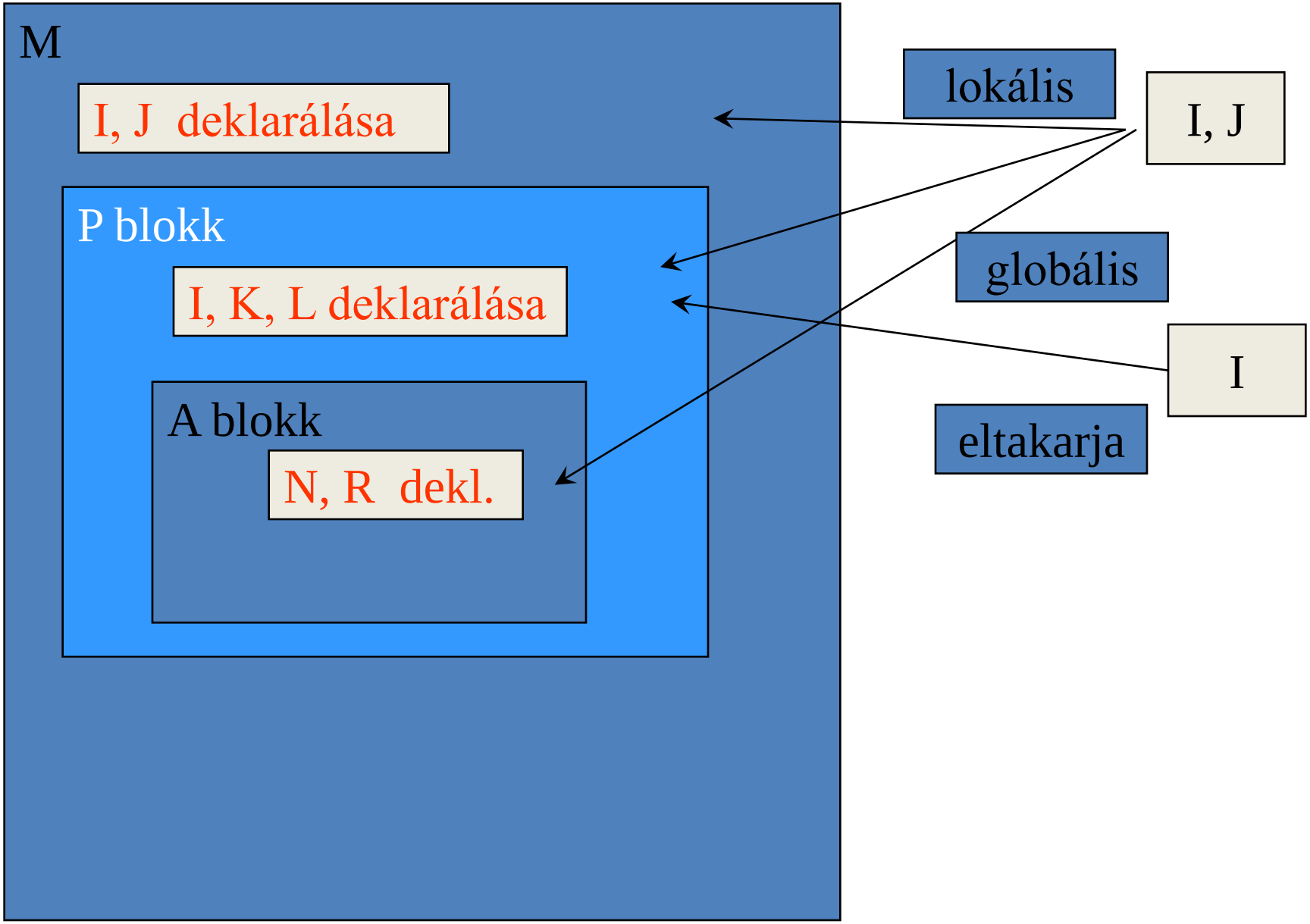
```
function Faktoriális ( N: Natural ) return Positive
   is
    Fakt: Positive := 1;
begin
    for I in 1..N loop
        Fakt := Fakt * I;
    end loop;
    return Fakt;
end Faktoriális;
```



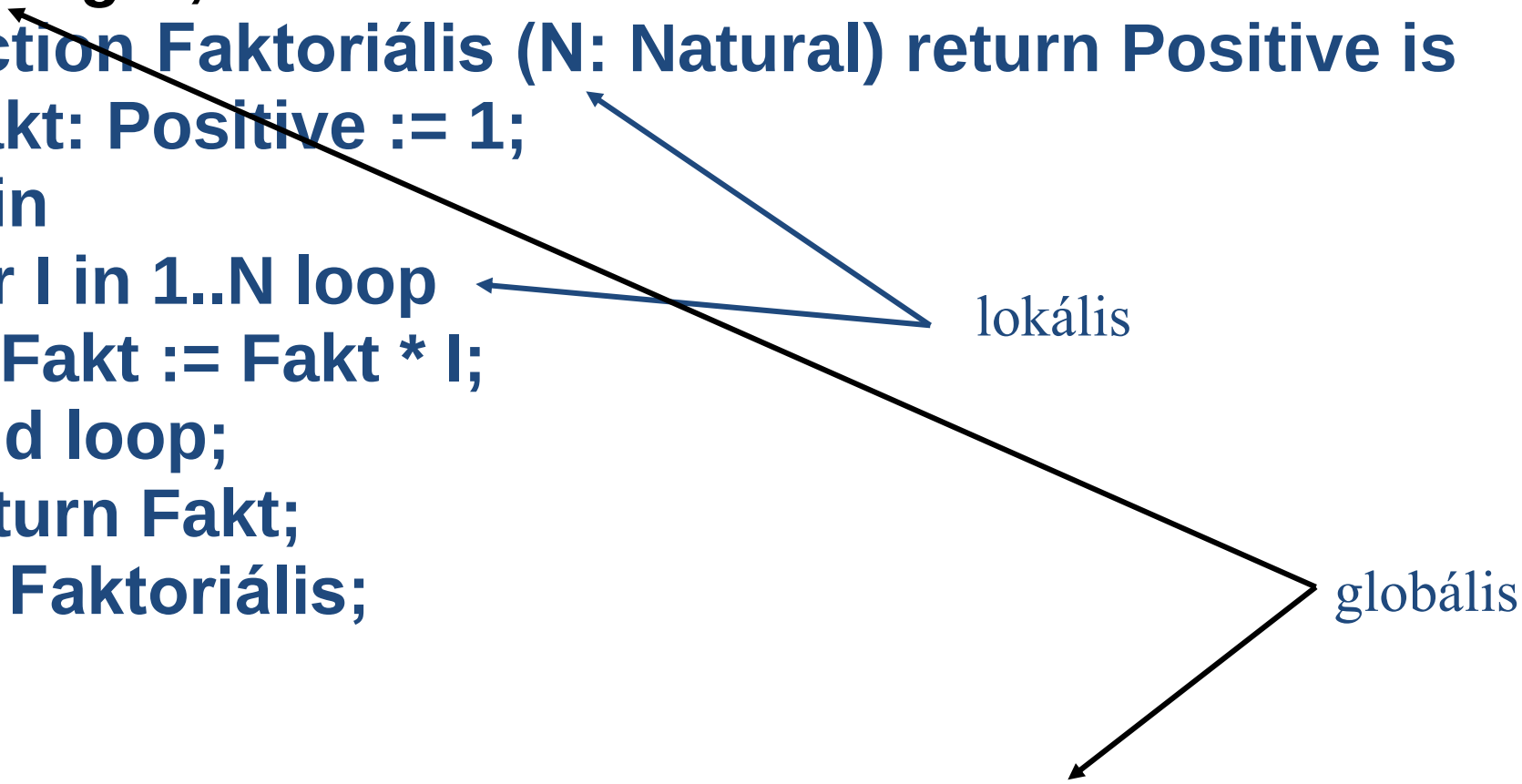
```
L_Al_K:=Faktoriális(L)/(Faktoriális(K)*Faktoriális(L-K))
```

Alprogram beágyazható

- Sok programozási nyelvben:
  - Blokkszerűen
  - Deklarációs részbe
  - Lokális - globális deklarációk
  - Hatáskör, láthatóság, élettartam


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

```
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, azaz 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

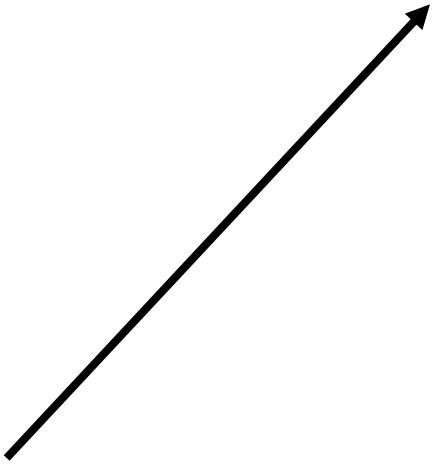
b=16

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);
```



x marad 10,

z értéke 2 lesz

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:

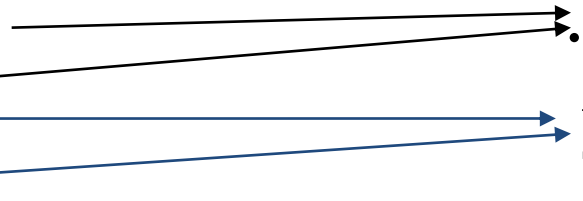
- Az aktuális paraméter vagy egy változó, vagy egy változó komponensének címét meghatározó kifejezés – balérték - (pl. x[i+2*j]) lehet.
- Az aktuális paraméter kiértékelése a hozzá rendelt memóriaterület címének meghatározását jelenti.
- Az aktuális paraméter kiértékelésére az alprogram végrehajtásának 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 (2)

```
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 (3)

„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?

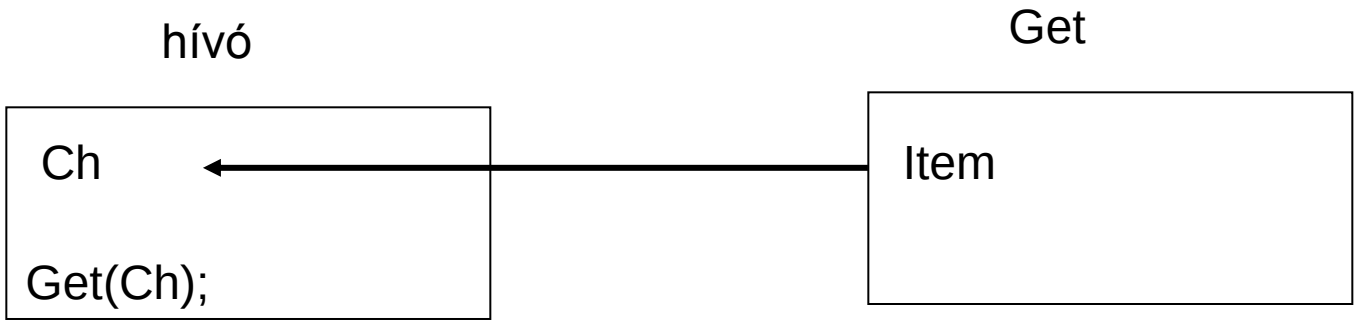
Eredmény szerinti paraméterátadás:

- 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:

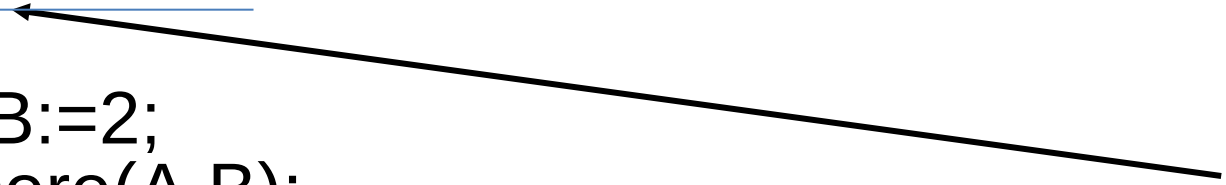
- Az alprogram befejezésének pillanatáig megegyezik az érték szerinti paraméterátadással.
- Az alprogram végrehajtásának befejezésekor az aktuális paraméter felveszi a formális paraméter pillanatnyi értékét (update módú átadásra alkalmas).
- Az aktuális paraméter csak „balérték” lehet.
- Miben különbözik a cím szerinti átadástól?
- Lehetséges előnyei

Érték/Eredmény szerinti paraméterátadás

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

X=1,Y=2

X=2,Y=1



A=2, B=1

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…

```
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!
  integer i, low, high;
  begin
   real rtn;
   rtn := 0;
   for i := low step 1 until high do
      rtn := rtn + expr;
   sum := rtn;
end sum                         hívása pl.:
y := sum (3*x*x - 5*x + 2, x, 1, 10);  Mit csinál?
```

és y:=sum(a[i],i, 1,100) ?

- 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.

- 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.

```
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
```

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 (
        Kezdôpont, 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

Mellékhatás – mi az?

… z:= sin(x); y:=sin(x); …

z = y?

mi az elvárásom?

… zz=rnd(); xx:=rnd(); ….

zz = xx?

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;
```

VIGYÁZAT!!

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;
```

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**

- **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?**

- **Algol60-68:**

  Algol60: Az alprogramok paramétereit alapértelmezésben név szerint veszi át, de a `value` kulcsszó használatával érték szerinti lesz az átadás. 1966-ban lett eredmény szerinti paraméterátadás (`result`) is.

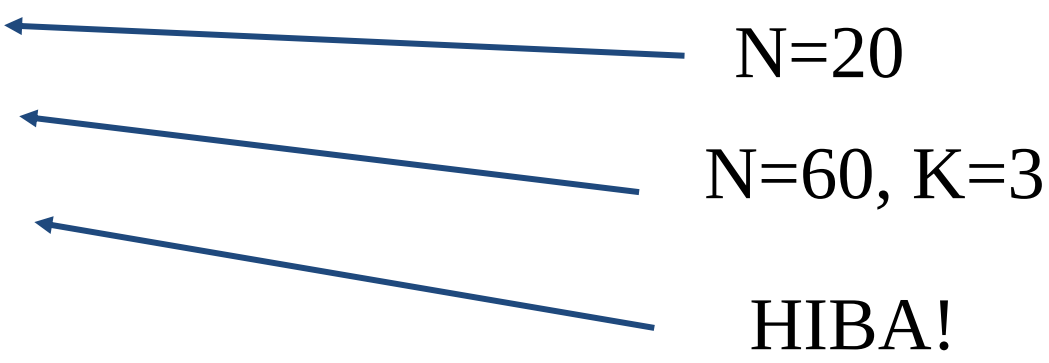
  Algol68: Az érték szerinti paraméterátadás lett az alapértelmezés, a cím szerintit pedig a `ref` kulcsszóval lehet elérni.
  Nincs se név, se eredmény szerinti.

  Van operátor átdefiniálás!
  (sőt, a precedenciák is változhatnak!)

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.
```

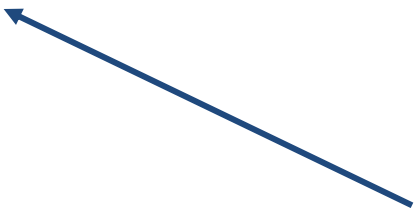


C/C++:

- 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

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! -- pl. külön paraméter lehet.

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

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

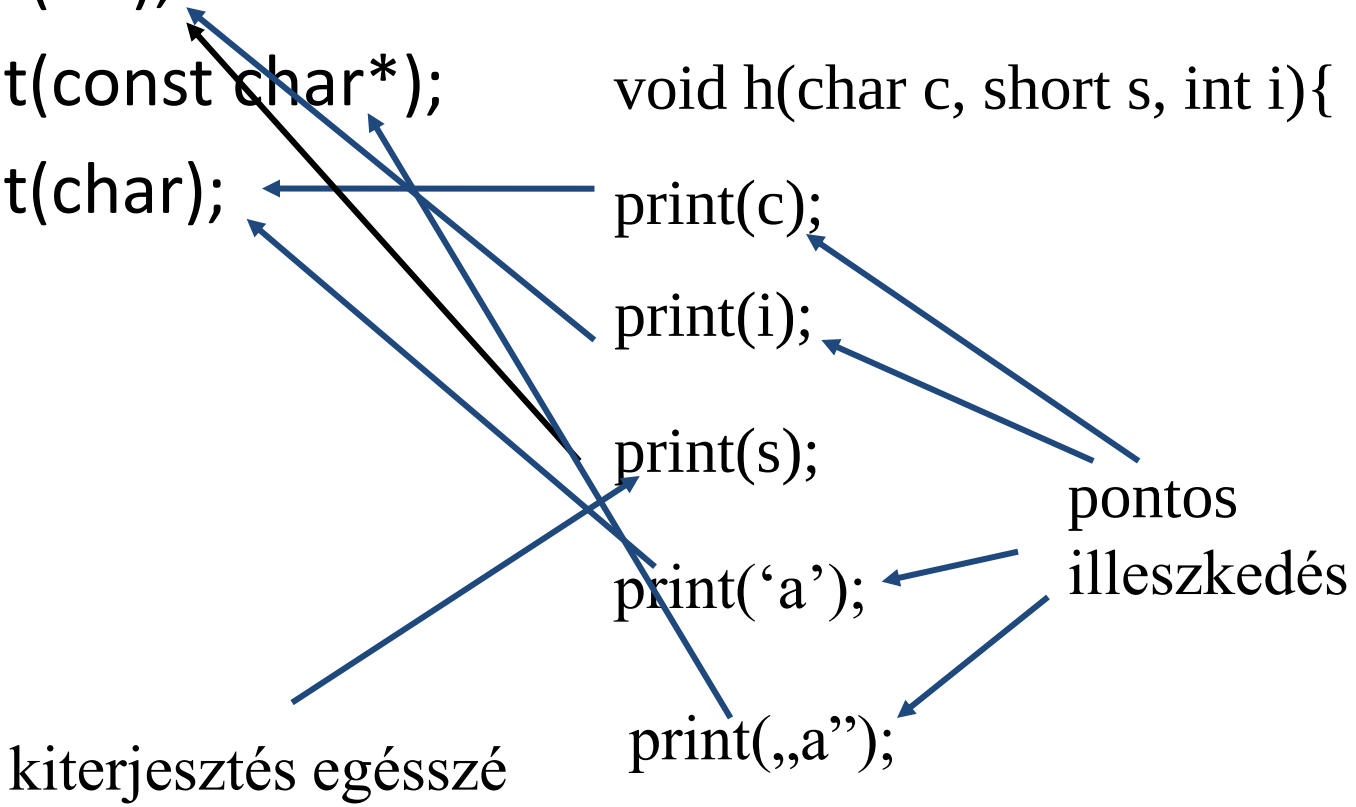
Túlterhelés lehetséges:

void print(int);

void print(const char*);

void print(char);

…



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 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.