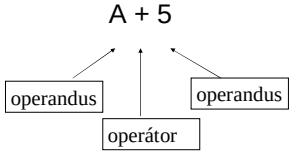
Vezérlési szerkezetek/ Utasítások

Egyszerű utasítások:

- Értékadás
  változó értékadásjel kifejezés
  vagy fordítva?
  értékadásjel: := ill. = de van: -> is
  szemantikája

Kifejezések

- operandusokból és operátorokból áll
- minden operandus lehet egy újabb kifejezés



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, ADA95, Eiffel, …
- C, C++, Java, …

Van-e többszörös értékadás?

A COBOL eszköztára 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.
```

most már lehet:

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


Pascal:

Változónév := Kifejezés;

A változó és a kifejezés típusa megegyező, de legalább kompatíbilis kell legyen.

m: integer; ⇒

m:= m+1;

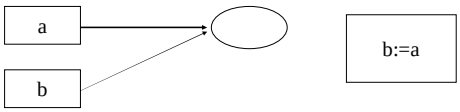
vagy:

p := keres(gyoker, x);

Többszörös értékadás nem megengedett.

CLU:

referenciákat használ => értékadás hatására két változó hivatkozhat ugyanarra az objektumra.



Többszörös értékadás megengedett:
pl. *x, y := y, x.*

ADA95:

- A változó és a kifejezés típusa megegyező kell legyen. Fordítási időben ellenőrzi.
- Altípusnál, ha nem jó aktuálisan: Constraint_Error futáskor.

I: Integer range 1..10;
J: Integer range 1..20:=20;

I:=J; -- Constraint_Error

J:=I; -- mindig jó

- Teljes rekordok és tömbök értékadása is megengedett.

C++:

Számos értékadó operátor jobbról balra feldolgozva:
A=B=C* jelentése: *A=(B=C);

Értékadó operátorok :
= *= /= %= += -= >>= <<= &= ^= !=

ero += Megeszi(taplalek);

Java:

- A C++ -hoz hasonló, számos értékadó operátor: = *= /= %= += -= >>= <<= >>>= &= ^= !=.

  E1 op= E2 jelentése: *E1= (T)((E2) op(E1))*, ahol *T* az *E1* típusa.
- Összetett értékadó operátoroknál mindkét operandus primitív típusú kell legyen (kivéve: +=, ha a bal operandus *String* típusú),
- Implicit cast előfordulhat!

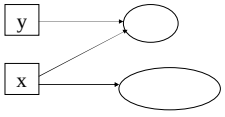
Pl.: *short x=3; x+=4.6;* eredménye: *x* ==7!

- 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?

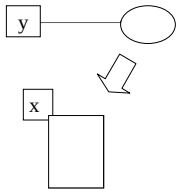
1. TX, TY referencia:



x:=y

referencia újrahozzárendelés

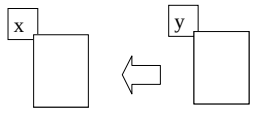
2. TX kiterjesztett, TY referencia:



x:=y

másolás

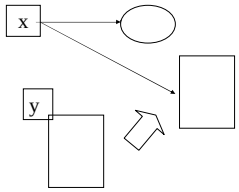
3. TX kiterjesztett, TY kiterjesztett:



x:=y

másolás

4. TX referencia, TY kiterjesztett:



x:=y

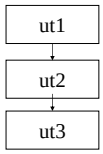
klónozás

Üres utasítás

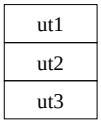
- **Pascal:** megengedett (case)
- **ADA95:** null; (case)
- **C++, Java:** ";" használható - pl. ciklusnak lehet üres törzse.
- **Eiffel:** ";" használható (nincs valódi szerepe)

Összetett utasítások:

- Szekvencia:



ut1; ut2; ut3;



Szekvencia

- Lehet-e üres?
- Terminátor vagy utasításelválasztó-e a ';'?
- Lehet-e blokkutasítást létrehozni?
- Elhelyezhető-e a blokkutasításban deklaráció?

- **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!
- ADA95: a ";" lezárja az utasítást
- C++, Java: a ";" lezárja az utasítást (de nem mindet, pl. a blokk utasítást nem).
- Eiffel: nincs szükség elválasztójelre, de a ";" megengedett.

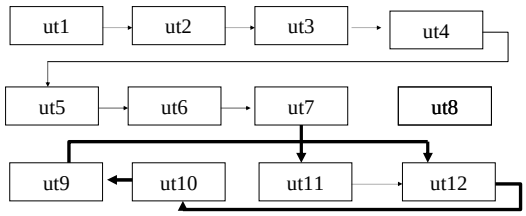
Blokk utasítások

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 (pl. Pascal, C++, Java).

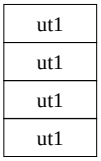
- **Pascal:** *begin utasítássorozat end*
- **ADA-95:**
  [declare
  deklarációk]
  begin utasítássorozat [kivételkezelő rész] end;
  Pl.: Csere:
  declare
  Temp : Integer;
  begin
  Temp := I; I := J; J := Temp;
  end Csere;
- **C++, Java:** "{" és "}" között.
- **Eiffel:** nincs explicit blokk utasítás.

Feltétel nélküli vezérlésátadás

- „goto"



Feltétel nélküli vezérlésátadás



javaslat : goto nélkül

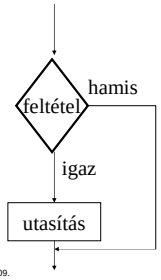
Feltétel nélküli vezérlésátadás

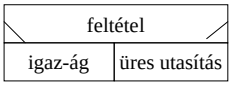
Ha mégis – Pascal, C stb. :
cimke:
.... utasítások ...
goto cimke;

Nincs: Modula-3, Java

Elágazások:

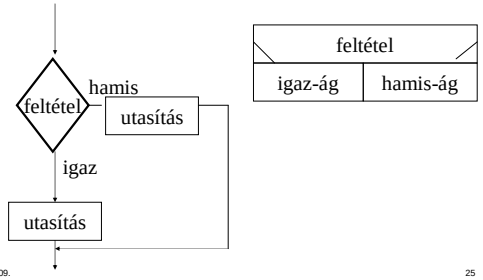
- Feltétel értékétől függően valamilyen utasítás(csoport) végrehajtása





Elágazások:

- Feltétel 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)

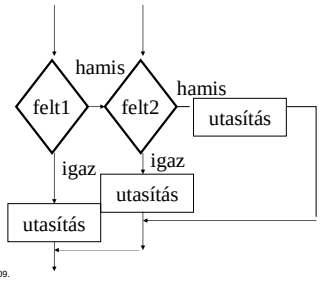
vagy

if felt1 then (if felt2 then ut1) else ut2 ?

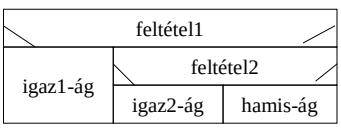
Elágazások

- 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;

if felt1 then s1;

elsif felt3 then s3;
stb.

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
```



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


Vigyázat!

if expr then st1
nem ugyanaz, mint:

if expr then ; st1
ez ekvivalens:

if expr then begin end; st1

=> ha

if expr then S1 else S2 -ben
S1 után beszúrunk egy ” ;”-t, az hiba!

CLU:

- Van egy záró *end.*

Megengedett az *elseif* is:

if <expr>$_1$ then
<statm>$_1$
{elseif <expr>$_2$
<statm>$_2$}
[else
<statm>$_n$]
end

ADA95:

- *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 0: *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
```

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
```

Többirányú elágazások

- valamilyen kifejezés értékétől függően:

kifejezés					
é1	é2	é3	é4	…	…
ig az1-ág	ig az2-ág	ig az3-ág	ig az4-ág	ig az … -ág	ig az … -ág

- **Többirányú elágazás:**
  - **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
* const$_1$: st$_1$;*
* const$_2$: st$_2$;*
* …*
* const$_n$: st$_n$*
end

- 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?**
  - **kell adni egy “*others*” ágat, hogy minden lehetőséget lefedjünk**
  - **a kifejezés típusa diszkrét kell legyen (egész vagy felsorolási típ).**

Pascal:

- A szelektor típusa lehet: integer, character, boolean vagy tetszőleges felsorolási vagy intervallum típus.
- "*else*" megengedett, de nem kötelező (skip).
- *case var of*
  val_1: $statm_1$;
  val_2.: $statm_2$;
  ... val_i…val_j : $statm_i$;
  [else statm]
  end;

```
var Age: Byte;
case Age of
   0..13  : Write('Child');
   14..23 : Write('Young');
   24..69: Write('Adult');
   else     Write('Old');
end
```

CLU:

- *s : T...*
  tagcase s
  tag n_1 : <statm>$_1$
  tag n_i : <statm>$_i$
  tag n_j (idn : Tj): <statm>$_j$
  others : <statm>$_n$
  end
- A kifejezés oneof vagy variant lehet.
- *others* kötelező, ha nincs minden lefedve.

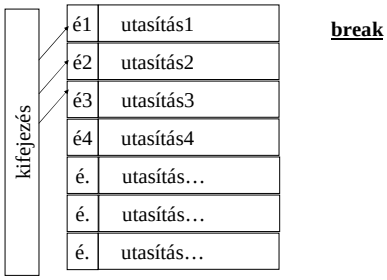
ADA95:

case expr is
when $choice_1$ => $statms_1$;
when $choice_2$ => $statms_2$; ...
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++:



C++:

switch (intexpr) {
case $label_1$: $statm_1$; break;
case $label_2$: $statm_2$; 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:

- A változó típusa *INTEGER* vagy *CHARACTER.*

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

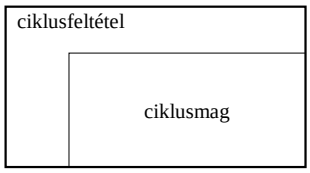
Exception lép fel, ha nem gondoskodtunk a megfelelő választék-elemről (null utasítás szükséges lehet).

Eiffel:

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


Ciklusok

- utasítások ismételt végrehajtása
  - valamilyen feltételtől függően



Ciklusok:

- **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:

- **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:

- **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;
- CLU:
  end zárja, így a ciklusmag utasítássorozat is lehet.
  while expr do body end

- ADA95:
  end loop zárja, így a ciklusmag utasítássorozat 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 utasítássorozat is lehet.

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

Ciklushelyesség kezelése

pl.:

```
from
  Result.start
until
  Result.off
loop
  Result.put(item);
  forth;
  Result.forth;
end;
```


Hátultesztelő “Repeat-until” ciklusok

- amíg egy feltétel igaz nem lesz.
- ciklusmag egyszer biztosan lefut

```
Repeat
 utasításssorozat
until ciklusfelt
```

nincs szükség blokk utasításra - a ciklusmag vége meghatározott

`while E do S` jelentése:

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


- **Pascal:**

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

- **CLU, ADA95**: nincs
- **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 <name> := <init-expr> to <limit-expr> do <statm>;

for <name> := <init-expr> downto <limit-expr> do <statm>;

```
for i := 1 to n do sum := sum + i;

for c := ‘z’ downto ‘a’ do   write(c);
```

CLU iterátor:

```
for d : int in all_primes() do ….

all_primes = iter () yields ( int )
  own prime_table : array[int] := array[int]$[2]
  i,p : int
  for pe : int in prime_table!elements do  yield (pe)     %yields 2
  end    %for
  p := prime_table!top + 1
  while true do
    i := 1
    while i <= prime_table!high cand p //prime_table[i] ~= 0 do
      i := i + 1
    end %while
    if i > prime_table!high then  prime_table!addh(p)  yield (p)
    end %if
    p := p + 1
  end %while
end all_primes
```

- **Ada95:**

  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++

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

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` equivalens: `while(1)`.
Ha a `for-init-stm` egy deklaráció, akkor a deklarált nevek hatásköre a for utasítást tartalmazó blokk

- **Java:**

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

```
public class ForDemo {
   public static void main(String[] args) {
      int[] arrayOfInts = { 32, 87, …., 622, 127 };

      for (int i = 0; i < arrayOfInts.length; i++) {
         System.out.print(arrayOfInts[i] + " ");
      }
      System.out.println(); …
   }
}
```


- **Java 5.0-ban**

  – 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();
  }
}
```


– Gyűjteményekkel (Collection) és tömbökkel használható

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

– Még egy példa:

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

- **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
  - 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 folyt.:

```
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 folyt.:

```
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);
    }
  }
}
```


“Végtelen” ciklusok

ADA95:

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** - VESZÉLYES!!!

- CLU:
  Nincs explicit goto utasítás, de számos “strukturált” vezérlésátadási lehetőség:
  - **continue**
  - **break**
  - **return**
  - **yield - iterátor törzsében- felfüggeszti az iterátor végrehajtását, visszaad egy értéket**
- **ADA95:**
  - **exit**
  - **return**
  - **goto.**

- C++:
  - **break - ciklusban, vagy switch utasításban**
  - **continue - csak ciklusban**
  - **return**
  - **goto**
- **Java:**
  - **break, continue, return mint a C++-ban**
  - **NINCS goto!**
- **Eiffel:**
  - **NINCS: break, continue, return, goto.**