

Programozás módszertan

Vezérlési szerkezetek, utasítások

Pere László (pipas@linux.pte.hu)

PÉCSI EGYETEM TERMÉSZETTUDOMÁNYI KAR
INFORMATIKA ÉS ÁLTALÁNOS TECHNIKA TANSZÉK

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMÍTÁSTECHNIKAI ÉS AUTOMATIZÁLÁI KUTATÓINTÉZET
ADATBÁZIS ÉS ADATKEZELÉSI OSZTÁLY

Értékdás

Értékadás

Az imperatív nyelvekben kitüntetett fontosságúak a változók. A változók értékének megváltoztatására szolgáló elemi utasításokat hívjuk értékadásnak.

COBOL

A COBOL nyelvben eredetileg nem volt olyan értékadás, amelynek felhasználásával összetett aritmetikai kifejezés értékét kiszámolhatjuk.

Az újabb változatokban már készíthetünk kifejezéseket a négy alapművelet és a zárójelek felhasználásával.

COBOL

```
MOVE változó TO változó.  
MOVE konstans TO változó.
```

A MOVE szerkezettel változók és konstansok értékét másolhatjuk változókba.

COBOL

```
ADD ért TO ért GIVING vált.  
SUBTRACT ért FROM ért GIVING vált.  
MULTIPLY ért BY ért GIVING vált.  
DIVIDE ért BY ért GIVING vált.
```

Ezekkel a szerkezetekkel a négy alapművelet végezhető el változóként vagy konstansként megadott argumentumokon. A GIVING záradékkal megadható, hova kerüljön az eredmény.

COBOL

```
COMPUTE kifejezés.
```

A COBOL újabb változatai lehetővé teszik összetett aritmetikai kifejezések kiértékelését az értékadásban.
Példa:

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

Pascal

```
változó := kifejezés;
```

A Pascal értékadása összetett kifejezések értékének kiszámítását is lehetővé teszi. Az értékadás két lépésben történik: első lépésben ki kell számolni a kifejezés értékét, a második lépésben hozzárendelni a változóhoz az új értéket.

Példa:

```
elemek := elemek + 1;
```


C

```
változó = kifejezés;
```

A C nyelvben az értékadás összetett kifejezést is tartalmazhat. Előbb ennek a kifejezésnek az értékét számoljuk ki, majd hozzárendeljük a változóhoz az új értéket.

C

A C nyelvben az értékadás jelölésére használt `=` jel operátor, visszatérési értéke a jobb oldalon található kifejezés értéke. Ennek köszönhetően többszörös értékadás is készíthető:

```
a = b = b + 1;
```

C

Néhány más értékadó operátor is használható a C nyelvben kifejezések egyszerűbb írásának érdekében.

A következő két kifejezés egyenértékű:

```
a = a + 18;  
a += 18;
```

Smalltalk

```
változó := kifejezés.
```

A Smalltalk értékadásai hasonlítanak a Pascal értékadásaihoz, de azokat az objektumszemlélet miatt kissé máshogyan kell értelmeznünk.

CLU

$$vált1, vált2 := érték1, érték2$$

A CLU támogatja a többszörös értékadást. Az értékadás két lépésben végezhető el. Az első lépésben kiszámoljuk a jobb oldalon található kifejezések értékét, a második lépésben hozzárendeljük a bal oldalon található változókhoz az értékeket.

CLU

A következő értékadás megcseréli A és B változók értékét:

$$A, B := B, A$$

Szekvencia, blokkutasítás

Szekvencia, blokkutasítás

- A szekvencia több egymás után írt utasítás (pl. több értékadás). Az utasításszekvencia a legtöbb nyelvben egymás után végrehajtandó elemeket tartalmaz.
- A blokkutasítás valamilyen szempontból egységként kezelendő utasításszekvencia.

Utasítások elválasztása

Bizonyos imperatív programozási nyelvek előírják, hogy a szekvenciában egymás után következő utasításokat valamilyen karakterrel *elválasszuk*.

Ezeknél a nyelveknél az elválasztásra szolgáló jelet csak akkor kell használnunk, ha két utasítás követi egymást.

Utasítások lezárása

Bizonyos imperatív programozási nyelvek előírják, hogy az utasításokat valamilyen karakterrel *lezárjuk*.

Ezeknél a nyelveknél a lezárásra használt karaktert minden utasítás után használnunk kell.

Utasításválasztás

A következő Pascal programrészlet bemutatja, hogy miképpen használjuk az utasítások elválasztására a ; karaktert.

```
begin  
    a := 13;  
    b := a + 1  
end
```

Utasítás lezárása

A következő C programrészlet bemutatja miképpen használjuk az utasítások lezárására a ; karaktert.

```
{  
    a = 13;  
    b = a + 1;  
}
```

Pascal

A Pascal nyelvben bizonyos helyeken csak egy utasítás állhat. Ilyen helyeken azonban használhatjuk a blokkutasítást, amely egyetlen utasításnak számít.

A Pascal blokkutasításai szekvenciálisan, a programban való előfordulásuk sorrendjében hajtódnak végre.

Pascal

A Pascal blokkutasításainak szerkezete:

```
begin  
    utasítás1;  
    utasítás2;  
    utasítás3  
end
```

Mivel a blokkutasítás is utasítás, a blokkutasítások egymásba ágyazhatók.

Pascal

A Pascal nem teszi lehetővé a változók hatókörének korlátozását egyszerű blokkutasítás segítségével.

A Pascal nyelvben tehát nem definiálhatunk blokkutasítás elején lokális változót.

Ada

- Az Adában a blokkutasításnak nevet adhatunk, amelyet a blokkutasítás végén feltüntetve olvashatóbbá tehetjük a programot.
- A különleges helyeken használt blokkutasításokat kulcsszavakkal különböztetjük meg (end if, end loop stb.).
- A blokkokon belül lokális változókat hozhatunk létre.

C

A C nyelvben bizonyos helyeken csak egy utasítás állhat.

Mivel a blokkutasítás egy utasításnak számít, ilyen helyeken blokkutasítás segítségével helyezhetünk el több utasítást.

C

A C blokkutasítás szerkezete:

```
{  
    utasítás1;  
    utasítás2;  
    utasítás3;  
}
```

Mivel a blokkutasítás is utasítás, a { } jelek egymásba ágyazhatók.

C

A C nyelv (90) lehetővé teszi lokális változók deklarációját blokkutasításon belül. A blokkon belül a lokális változók létrehozásának meg kell előznie az első végrehajtható utasítást.

(A könyv szerint a C nyelvben csak az első szint tartalmazhat lokális változókat, de ez implementációfüggő lehet.)

Smalltalk

A Smalltalk esetében a blokkutasítás igen hatékony és sokat használt eszköz. A blokkutasítás segítségével hozhatunk létre metódusokat, névvel rendelkező végrehajtható objektumokat.

Feltétel nélküli vezérlésátadás

Ellenérvek

Értelmét veszíti a végrehajtási verem, a lokális változók kezelésére is használt eszköz.

A rekurzív hívást használó függvény melyik hívási mélységébe ugorjunk?

Ezeket a problémákat sok nyelv úgy kerüli meg, hogy a feltétel nélküli vezérlésátadást függvényen belülre korlátozza.

Ellenérvek

A helyességbizonyítási módszerek alkalmazását megnehezíti a vezérlésátadás, mert nem lehet kezelhető részekre szabdalni a programot.

Másrésről sokszor nincs szükségünk helyességbizonyításra.

Ellenérvek

A vezérlésátadás nincs összhangban az emberi gondolkodással, ezért igen nehéz az ilyen eszközzel készített programokat olvasni, karbantartani.

A legtöbb imperatív nyelven a feltétel nélküli vezérlésátadás (túlzott) használata azt mutatja, hogy a programozó nem gondolta át a problémát a megfelelő gondossággal.

COBOL

A feltétel nélküli vezérlésátadás formája a COBOLban:

```
GOTO név.
```

Az utasítás végrehajtása után a program végrehajtása a megadott nevű utasítástól folytatódik tovább.

COBOL

A feltétel nélküli vezérlésátadó utasítás kiszámított változatának (kiszámított GOTO) szerkezete a COBOLban:

```
GOTO név1, név2  
      DEPENDING ON vált.
```

Az utasítás végrehajtása után a változó értékétől függő helyen folytatódik a program végrehajtása.

Pascal

A Pascal nyelvben megtalálható a feltétel nélküli vezérlésátadás címkék és a goto utasítás segítségével.

Mivel a Pascal magasan struktúrált nyelv (utasításblokkok, függvények, eljárások) a feltétel nélküli vezérlésátadás használata korlátozott.

Modula-3

A Modula-3 azon programozási nyelvek egyike, amelyekben nincsen feltétel nélküli vezérlésátadó szerkezet.

FORTRAN

A FORTRAN nyelvben a GO TO utasítás numerikus címkékkel történik. A címkék hatóköre korlátozott, ezért a GO TO utasítás nem irányulhat utasításblokkon kívülre.

A FORTRAN régebbi változatai (FORTRAN77) igen intenzíven használják a GO TO utasítást, nagy odafigyelést és szakértelmet kívánva a programozótól.

C

A C nyelvben a feltétel nélküli vezérlésátadás címkék segítségével használható, hatóköre korlátozott.

A szabványos C könyvtár `setjmp()` és `longjmp()` függvényei blokkon (függvényen) kívüli helyre való ugrást tesznek lehetővé. Meglehetősen nagy odafigyelést igényel a használatuk.

C

A goto utasítás használata a C nyelvben:

```
címkenév:  
    utasítás1;  
    utasítás2;  
goto címkenév;
```

(Az ugrás történhet előre és hátra is, természetesen függvényen belül.)

Feltételes utasításvégrehajtás

Feltételes utasításvégrehajtás

A feltételes utasításvégrehajtás olyan vezérlőszerkezet, amelynek segítségével az utasítás vagy utasításblokk végrehajtását a változók értékének állapotához mint feltételhez köthetjük.

Mivel az utasítások végrehajtásának láncolatában a feltételes utasításvégrehajtás alternatívaként jelentkezik, szokás ezt a vezérlőszerkezetet elágazásnak nevezni.

FORTRAN

A FORTRAN77 nyelvben a feltételes utasításvégrehajtás valóban az utasítás feltételes végrehajtását jelenti (feltétellel kiegészített utasítás).

```
IF ( feltétel ) utasítás
```

Itt a *feltétel* egy logikai kifejezés. Az utasítás akkor hajtódik végre, ha a feltételként adott kifejezés értéke igaz.

Az *utasítás* helyén nem állhat több utasítás (utasításszekvencia, utasításblokk), nem állhat feltételes utasítás, állhat viszont feltétel nélküli vezérlésátadás.

FORTRAN

Több utasítás feltételes végrehajtása:

```
IF(feltétel) GOTO cím1  
GOTO cím2  
cím1  utasítás1  
      utasítás2  
cím2  további utasítás
```

Ebben a szerkezetben *utasítás1* és *utasítás2* csak akkor hajtódik végre, ha a *feltétel* igaz értéket képvisel.

FORTRAN

Egyszerűsített változat:

```
IF(feltétel) GOTO cím1  
utasítás1  
utasítás2  
cím1 további utasítás
```

Ebben az esetben *utasítás1* és *utasítás2* akkor hajtódik végre, ha a *feltétel* hamis értéket képvisel.

COBOL

A COBOL lehetővé teszi utasításszekvenciák feltételes végrehajtását.

```
IF feltétel THEN  
    utasítás1  
    utasítás2  
ELSE  
    utasítás3  
    utasítás4.
```

Az első utasításszekvencia a *feltétel* igaz, a második a *feltétel* hamis értéke mellett hajtódik végre.

A nyelv lehetővé teszi az ELSE és a hamis ág elhagyását.

Pascal

A Pascal a COBOL nyelvhez hasonló feltételes utasításvégrehajtással rendelkezik:

```
if feltétel then  
    utasítás1  
else  
    utasítás2;
```

Az igaz és a hamis ágba is egy utasítás lehet, a hamis ág az else kulcsszóval együtt elhagyható.

Pascal

A ; az utasítást választja el a következő utasítástól:

```
if feltétel then  
    utasítás1  
else  
    utasítás2;  
    utasítás3;
```

Láthatjuk, hogy ebben az értelemben a teljes if szerkezet egy utasítás.

Pascal

Ha elhagyjuk a hamis ágot, az if szerkezetet utasításként a következő utasítástól a következőképpen választjuk el:

```
if feltétel then  
    utasítás1;  
utasítás2;
```

Nyilvánvalóan nyelvtani okai vannak annak, hogy az igaz és hamis ágakban csak egy utasítás szerepelhet! Így garantálhatjuk, hogy mindenki számára egyértelmű, melyik utasítás része a szerkezetnek és melyik nem.

Pascal

Ha az igaz vagy a hamis ágba több utasítást szeretnénk írni, blokkutasítást kell használnunk:

```
if feltétel then  
begin  
    utasítás1;  
    utasítás2;  
end;  
utasítás3;
```

A begin és end kulcsszavak lehetővé teszik az ág végének jelzését.

Csellengő else

Több programozási nyelvben – így a Pascalban is – tisztázatlan a következő formájú kifejezések jelentése:

```
if feltétel1 then  
    if feltétel2 then  
        utasítás1  
    else  
        utasítás2;
```

Nem világos, hogy az else a külső vagy a belső elágazás hamis ágát jelzi.

Csellengő else

Több programozási nyelvben – így a Pascalban is – tisztázatlan a következő formájú kifejezések jelentése:

```
if feltétel1 then  
    if feltétel2 then  
        utasítás1  
else  
    utasítás2;
```

Nem világos, hogy az else a külső vagy a belső elágazás hamis ágát jelzi.

Modula-3

A Pascal esetében sokszor elfelejtették az igaz vagy hamis ág bővítésekor jelezni az utasításblokkot (begin–end).

```
IF feltétel
THEN
    utasítás1;
    utasítás2;
ELSE
    utasítás3;
END;
```

A több utasítás lehetőségének ára a kötelező END használat.

Modula-3

Bizonyos szerkezetek egyszerűbb írásának érdekében a nyelv bevezette az ELSIF kulcsszót.

```
IF feltétel1 THEN utasítás1  
ELSIF feltétel2 THEN utasítás2  
ELSE utasítás3  
END
```

Ezt a módszert később több más nyelv is átvette a csellengő else problémájának megoldásaképp.

C

A feltételes utasításvégrehajtás szerkezete a C nyelvben a következő:

```
if( feltétel )  
    utasítás1;  
else  
    utasítás2;
```

Az igaz és hamis ágban egyetlen utasítás állhat. Ha nincs szükségünk a hamis ágra, az else kulcsszóval együtt elhagyható.

C

Ha több utasítást szeretnénk elhelyezni az igaz vagy a hamis ágba, utasításblokkot kell létrehoznunk:

```
if( feltétel ) {  
    utasítás1;  
    utasítás2;  
} else  
    utasítás3;
```

Többirányú elágazás

A többirányú elágazás

A többirányú elágazás egy kifejezés értékétől függően kettőnél több utasítás vagy utasításblokk alternatív végrehajtását teszi lehetővé.

A legtöbb programozási nyelv ugrási táblázatot használ a többirányú elágazás kezelésére. Ez korlátot jelenthet az alternatívák számára nézve.

Pascal

A többirányú elágazás formája a Pascal nyelvben a következő:

```
case diszkrét kifejezés of  
    állandó1: utasítás1;  
    állandó2: utasítás2;  
    állandó3: utasítás3;  
    else utasítás4;  
end
```

Az egyes utasítások helyén a begin–end kulcsszavak segítségével utasításblokk is állhat.

A szerkezetből az else kulcsszó és az utána következő utasítás elhagyható.

Pascal

- A szelektorkifejezésnek egésznek vagy felsorolási típusnak kell lennie. Nem állhat tehát itt pl. lebegőpontos szám.
- Az ágak jelzésére használt állandók értéke már a fordításkor is ismert. Pascalban itt tartomány is állhat.
- Ha egyetlen ágnak sem feleltethető meg a szelektorkifejezés és nincs else ág sem, a programvégrehajtás a többirányú elágazás után folytatódik.

Modula-3

Ahogy a feltételes utasításvégrehajtásnál is, a többirányú elágazásnál is elhagyhatók a begin/end párok a Modula-3 esetében.

```
CASE diszkrét kifejezés OF
  állandó1 => utasításlista1;
  | állandó2 => utasításlista2;
  | állandó3 => utasításlista3;
  ELSE utasításlista4;
END
```

A | karakter jelzi az egyes ágak végét, akkor is, ha az ágot egyetlen utasítás alkotja.

Modula-3

- A szelektorkifejezés itt is kötelezően diszkrét értékű kifejezés.
- Az egyes ágakhoz tartozó értékek állandók, értékük már a fordításkor ismert. Egy ághoz több állandó vagy tartomány is tartozhat.
- Ha a szelektorkifejezés értékének megfelelő ág nem található – az ELSE ággal együtt –, futási hiba következik be.
Ilyenkor a program végrehajtása hibaüzenettel leáll.

C

A többirányú elágazás szerkezete a C nyelvben a következő:

```
switch( diszkrét kifejezés) {  
    case érték1:  
        utasításlista1;  
        break;  
    case érték2:  
        utasításlista2;  
        break;  
    default:  
        utasításlista3;  
}
```

- A szelektorkifejezés diszkrét típusú.
- Az egyes ágakhoz tartozó értékek olyan állandókból álló kifejezések, amelyek értéke már a fordításkor ismert. Nem használhatunk tartományt.
- Ha egyetlen ághoz sem rendelhető a diszkrét kifejezés (a default ághoz sem), a programvégrehajtás a többszörös elágazás után folytatódik.
- Több utasítást is elhelyezhetünk az egyes ágakban, a vezérlés a következő ágra csorog, ha nem használjuk a break kulcsszót.

C

```
switch (a+c) {  
    case 3:  
    case 4:  
    case 5:  
        utasítás1;  
    break;  
    case 6:  
        utasítás2;  
    case 7:  
        utasítás3;  
}
```


FORTRAN

FORTRAN nyelvben többirányú elágazásnak tekinthető az ún. numerikus IF szerkezet, melynek formája:

```
IF ( kif. ) címke1, címke2, címke3
```

Ha a numerikus kifejezés értéke negatív, a végrehajtás *címke1*-nél, ha 0, *címke2*-nél, ha pedig pozitív, *címke3*-nál folytatódik.

Ciklusszervező utasítások

Ciklusszervező utasítások

A ciklusszervező utasítások lehetővé teszik programrészek ismételt, többszörös végrehajtását, a változók aktuális értékétől függően.

Habár minden ciklikus jellegű feladat megoldható a feltételes utasításvégrehajtás és a feltétel nélküli vezérlésátadás segítségével, a ciklusszervező utasítások nagyon megkönnyítik a programozó munkáját és nagymértékben segítik a program megértését.

Ciklusok felosztása

- Elöltesztelő ciklusról beszélünk, ha a ciklus ismétlésének feltételét a ciklikusan ismétlendő utasítások végrehajtása előtt értékeljük ki. Ilyen szerkezetet használva előfordulhat, hogy egyszer sem hajtjuk végre a ciklusutasításokat.
- Hátultesztelő ciklusról beszélünk, ha a feltétel kiértékelése a ciklusutasítások végrehajtása után történik. Ilyen szerkezet esetében a ciklusutasítások egyszer mindenképpen végrehajtódnak.

COBOL

A COBOL támogatja a feltétel nélküli vezérlésátadást. A GOTO utasítás segítségével ciklusokat készíthetünk COBOL nyelven.

A COBOL PERFORM utasítása segítségével fejlettebb formájú ciklusokat készíthetünk.

COBOL PERFORM

A PERFORM utasítás többféleképpen használható.

```
PERFORM név.
```

```
PERFORM név1 THRU név2.
```

```
PERFORM név szám TIMES.
```

```
PERFORM név UNTIL feltétel.
```

```
PERFORM név VARYING változó  
FROM érték BY növ UNTIL feltétel.
```

Pascal

A while utasítás segítségével előtesztelő utasítást készíthetünk a Pascal nyelvben:

```
while feltétel do  
    utasítás;
```

Ha a ciklusmagban több utasítást szeretnénk elhelyezni, azokat begin–end pár közé kell írunk.

Pascal

Hátultesztelő ciklust a Pascalban a következőképpen készíthetünk:

```
repeat  
    utasítás1;  
    utasítás2  
until feltétel;
```

A while ciklushoz képest fordított a feltétel hatása: a ciklus addig folytatódik, amíg a feltétel hamis.

Pascal

Előre ismert lépésszámú ciklust Pascal nyelven a for utasítással készíthetünk:

```
for változó := ért1 to ért2 do  
  utasítás;
```

```
for változó := ért1 downto ért2 do  
  utasítás;
```

A lépésköz 1 ill. -1, a ciklus pedig addig ismétlődik, amíg az érték2-t túl nem haladtuk.

Pascal

A Pascal for utasításának sok vitatott tulajdonsága van:

- A ciklus csak $+1$ illetve -1 lépésközzel változhat.
- A ciklusváltozó értéke a ciklus végrehajtása után a használt fordítótól függ.
- A ciklusváltozó változtatása nem megengedett, de ezt a fordító nem ellenőrzi.
- A felső határ kiszámítására csak egyszer kerül sor.

Modula-3

A Pascalhoz képest a feltételes utasításvégrehajtáshoz hasonlóan a BEGIN–END használata kötelező a while esetben, a repeat–until pedig változatlan formában használható.

Modula-3

A for ciklus a Pascalhoz képest a következőképpen változott:

```
FOR vált := ért1 TO ért2 BY lépésköz  
DO  
    utasítás1;  
    utasítás2  
END
```

Modula-3

A for ciklus viselkedése Modula-3 esetében a következőképpen változott a Pascal nyelvhez képest:

- A ciklusváltozó csak a FOR szerkezeten belül érvényes, arra nézve lokális. A ciklusváltozó a ciklus után megszűnik.
- A ciklusváltozó deklarációja a FOR utasítással történik.
- A ciklusváltozó a cikluson belül nem módosítható, ezt a fordító ellenőrzi.

Modula-3

A Modula-3 külön szerkezetet használ a végtelen ciklus jelölésére:

```
LOOP
    utasítás1;
    utasítás2;
    ... EXIT;
    utasításn
END
```

Az ilyen ciklusból általában feltételes utasításvégrehajtás segítségével végrehajtott EXIT utasítással lépünk ki.

Ada

Az Ada for ciklusának szerkezetei a következők:

```
for változó in tartomány loop  
    utasítás1  
    utasítás2  
end loop;
```

```
for változó in reverse tartomány loop  
    utasítás1  
    utasítás2  
end loop;
```

Ada

- A ciklusváltozó a ciklus lokális konstansa.
- A ciklusváltozó a ciklusban definiált, az értéktartomány legszűkebb típusaként.
- A ciklusváltozó típusa diszkrét (nem folytonos) típus.
- Az értéktartomány kiszámítására csak egyszer kerül sor.
- A ciklusváltozó elfedi az hasonló névvel rendelkező külső változót.

C

Az előltesztelő utasítás szerkezete a C nyelvben:

```
while ( feltétel )  
    utasítás;
```

A feltétel egy aritmetikai, logikai értéket adó utasítás, kifejezés. A ciklusmagot végrehajtjuk, ha a feltétel igaz, 0-tól különböző az értéke.

Ha több utasítást szeretnénk elhelyezni a ciklusmagban, használnunk kell a { } jeleket.

C

A hátultesztelő utasítás szerkezete a C nyelvben:

```
do  
    utasítás;  
while ( feltétel );
```

A feltétel és értelmezése itt is az előzőek szerint alakul.

Ha több utasítást szeretnénk elhelyezni a ciklusban, a { } használata kötelező.

C

Az előre ismert lépésszámú ciklus készítésére szolgáló szerkezet a C nyelvben:

```
for ( ut1; ut2; ut3 )  
    utasítás;
```

Az *ut1* a ciklus első végrehajtása előtt, az *ut3* a ciklusmag végrehajtása után hajtódik végre. Az *ut2* a ciklusmag végrehajtása előtt hajtódik végre, igaz értéke esetén hajtjuk végre a ciklust.

Ha több utasítást szeretnénk elhelyezni a ciklusmagban, használnunk kell a { } jeleket.

C

A következő példa bemutatja a for ciklus egyszerű használatát:

```
for (i=0; i<10; ++i)
    printf("i=%d", i);
```

C

A for ciklusustasítás mindhárom tagjának elhagyásával végtelen ciklust készíthetünk:

```
for ( ;; )  
    printf("infinite... ")
```

Java

A Java nyelv lehetővé teszi, hogy a ciklusváltozót az utasításban deklaráljuk, ami lehetővé teszi, hogy az csak a ciklus törzsén belül létezzen. Ezt mutatja be a következő példa:

```
for (int i=0; i<10; ++i )  
    utasítás;
```

Iterátorok

Az iterátorok

Az iterátorok (bejárók) olyan ciklusszervező utasítások, amelyek megkönnyítik egy halmaz elemeinek végigjárását.

Ha a halmaz elemeire hivatkozni tudunk pl. egész számokkal (pl. tömbök), az egyszerű for ciklus is megfelelő lehet, de ha nem, bejáróciklust kell használnunk.

Iterátorok (példa)

A következő AWK program a bejárók használatára ad példát.

Az AWK igen praktikus eszköze az asszociatív tömb, ahol a tömb elemeinek címezéséhez nem csak számokat, számpárokat, n -eseket, hanem például karakterláncokat is használhatunk.

Az AWK nyelvben karakterlánccal címzett asszociatív tömböt csak bejáróciklussal járhatunk végig.

AWK (példa)

AWK/iterator.awk

```
#!/bin/awk -f

BEGIN {
    szavak["alma"] = "apple"
    szavak["világ"] = "world"
    szavak["csillag"] = "star"

    for (m in szavak)
        print m, szavak[m]
}
```

Iterátorciklust a C# nyelvben a következő formában készíthetünk:

```
foreach ( vált in tömb )  
    utasítás
```

Ha az iterátorciklus magjában több utasítást is el szeretnénk helyezni, használnunk kell a { } jeleket.

Valójában minden objektum lehet iterált ciklus alapja, ami implementálja a következő metódusokat:

MoveNext Következő elemre ugrás.

Current Jelenlegi elem.

Reset Első elemre ugrás.

A metódusok implementálásával a programozó is készíthet IEnumerable interfészt, olyan objektumot hozva létre, amely iterált ciklusok alapja lehet.