ELTE | IK

PROGRAMOZÁS

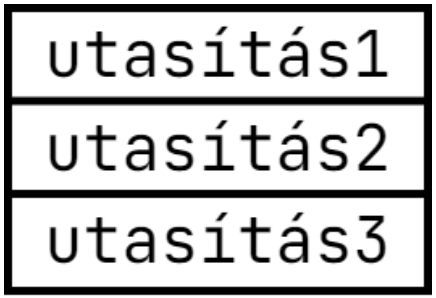
Adatszerkezetek

Horváth Győző

Ismétlés

Szekvencia

Algoritmus:

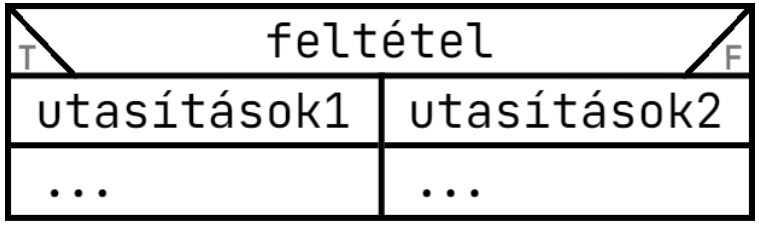


Kód:

```
utasítás1;
utasítás2;
utasítás3;
```

Elágazások

Algoritmus:



kétirányú

feltétel1	feltétel2	...	egyébként
utasítások1	utasítások2	...	utasítások

többirányú

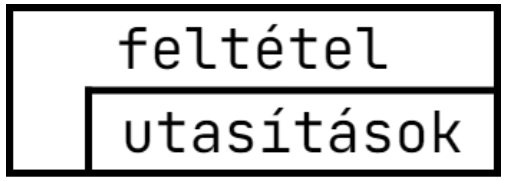
Kód:

```
if (feltétel) {
  utasítások1;
} else {
  utasítások2;
}
```

```
if (feltétel1) {
  utasítások1;
}
else if (feltétel2) {
  utasítások2;
}
// ...
else {
  utasítások;
}
```

Ciklusok

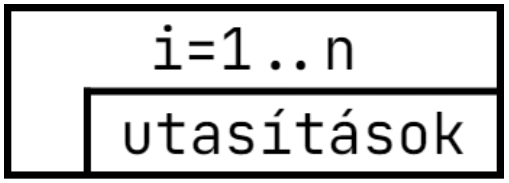
Feltételes ciklus:



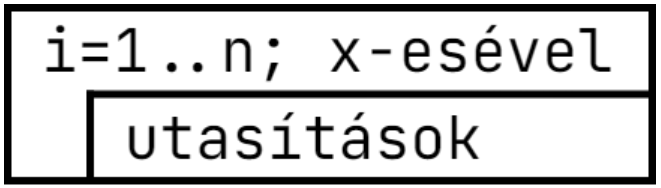

```
while (feltétel){
  utasítások
}
```


```
do{
  utasítások
} while (feltétel);
```

Számlálós ciklus:




```
for (int i=1;i<=n;++i){
  utasítások
}
```




```
for (int i=1;i<=n;i+=x){
  utasítások
}
```

Rekord

Rekord

Példák:
x=3,3; y=2,1 → sn=1
x=3,3; y=-2 → sn=4
…

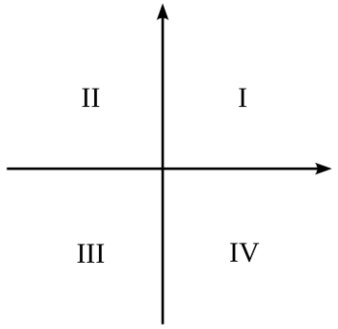
Feladat:

Adjuk meg, hogy egy síkbeli pont melyik síknegyedbe esik!

Specifikáció$_1$ és algoritmus$_1$:

```
Be: x∈R, y∈R
Ki: sn∈N
Ef: -
Uf: ((x>=0 és y>=0) -> sn=1) és
    ((x<0  és y>=0) -> sn=2) és
    ((x<0  és y<0)  -> sn=3) és
    ((x>=0 és y<0)  -> sn=4)
```

x≥0 és y≥0	x<0 és y≥0	x<0 és y<0	x≥0 és y<0
sn:=1	sn:=2	sn:=3	sn:=4



Be: x∈R, y∈R
Ki: sn∈N

Rekord

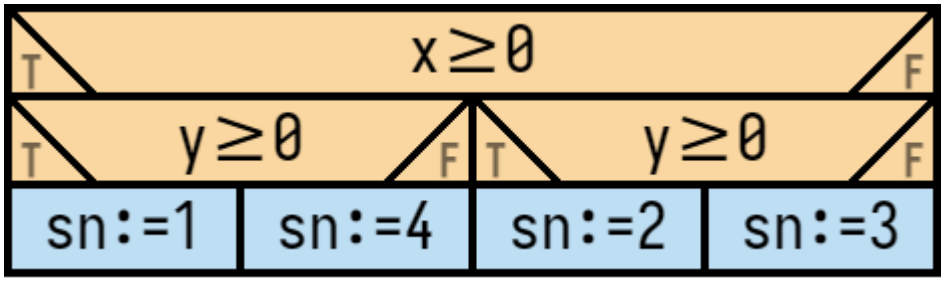
x≥0 és y≥0	x<0 és y≥0	x<0 és y<0	x≥0 és y<0
sn:=1	sn:=2	sn:=3	sn:=4

```
// 1. deklarálás
double x, y;
int sn = 0;
// 2. beolvasás
Console.Write("x = ");
double.TryParse(Console.ReadLine(), out x);
Console.Write("y = ");
double.TryParse(Console.ReadLine(), out y);
// 3. feldolgozás
if (x >= 0 && y >= 0) { sn = 1; }
else if (x < 0 && y >= 0) { sn = 2; }
else if (x < 0 && y < 0) { sn = 3; }
else if (x >= 0 && y < 0) { sn = 4; }
// 4. kiírás
Console.WriteLine("Síknegyed = {0}", sn);
```

Rekord

Specifikáció$_2$ és algoritmus$_2$:

```
Uf: (x>=0 -> (y>=0 -> sn = 1 és
                y< 0 -> sn = 4))
    és
    (x<0  -> (y>=0 -> sn = 2 és
                y< 0 -> sn = 3))
```




```
îğ  y      ◦
  îğ  ù      ◦     şŋ    ,
  êľşê    şŋ    _

êľşê
  îğ  ù      ◦     şŋ    ,
  êľşê    şŋ    ‘
```

Rekord

- Jelenlegi megoldás:
  - nincs x és y között szemantikus kapcsolat
  - pedig nemcsak két szám, hanem **együtt** jelölnek egy koordinátapárt
- **Rekord:**
  - *különböző funkciójú adatok egybezárása*
  - szemantikus egység létrehozása
  - „funkció”: mit *jelent* az adat

Rekord

Példák:
(x=3,3; y=2,1) → sn=1
(x=3,3; y=-2) → sn=4

Specifikációbeli jelölés:

- egy adat
  - x∈R
- adattöbbes
  - p∈Pont, Pont=(x:R x y:R)
  - x: direkt szorzat
  - hivatkozás a **nevükkel**
    - p.x (p.x∈R)
    - p.y (p.y∈R)

p∈RxR

új halmaz definíciója

Direkt szorzat példa:
a∈[1..2], b∈[4..5]

a	b
1	4
1	5
2	4
2	5

A direkt szorzat felsorolja az összes lehetséges adatpárt.

Rekord

Példák:
(x=3,3; y=2,1) → sn=1
(x=3,3; y=-2) → sn=4

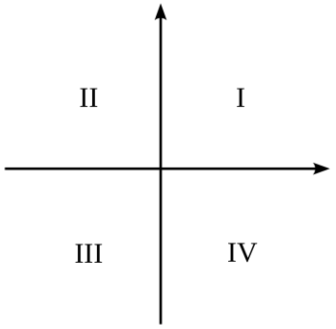
Feladat:

Adjuk meg, hogy egy síkbeli pont melyik síknegyedbe esik!

Specifikáció$_3$ és algoritmus$_3$:

```
Be: p∈Pont,
    Pont=(x:R x y:R)
Ki: sn∈N
Ef: -
Uf: ((p.x>=0 és p.y>=0) -> sn=1) és
    ((p.x<0  és p.y>=0) -> sn=2) és
    ((p.x<0  és p.y<0 ) -> sn=3) és
    ((p.x>=0 és p.y<0 ) -> sn=4)
```

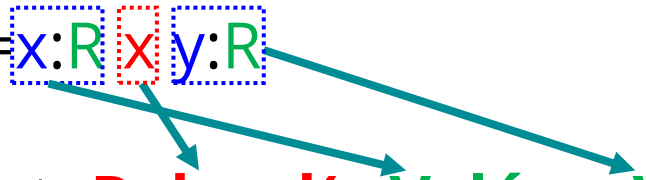
p.x≥0 és p.y≥0	p.x<0 és p.y≥0	p.x<0 és p.y<0	p.x≥0 és p.y<0
sn:=1	sn:=2	sn:=3	sn:=4



Rekord

Specifikáció → algoritmus$_{\text{adatleírás}}$:

- p∈Pont, Pont=x:R x y:R



Típus Pont=**Rekord(x:Valós,y:Valós)**

Változó p:Pont

Típus Pont=**Rekord(x,y:Valós)**

Tehát a Pont egy új **adattípus.**

- A rekordok **összetett** adatszerkezetek, a részeikre „**nevük**” által meghatározott szelektorokkal hivatkozunk: p=(p.**x**, p.**y**).

Rekord – típusdefiniálás kódban

Specifikáció → algoritmus → kód:

- **Típus**

Pont=Rekord(x,y:Valós)

Típus**definíció**

C# típusdefiníció:

```
struct Pont
{
    public double x,y;
}
```

típusazonosító

mezőtípus

hozzáférési jog

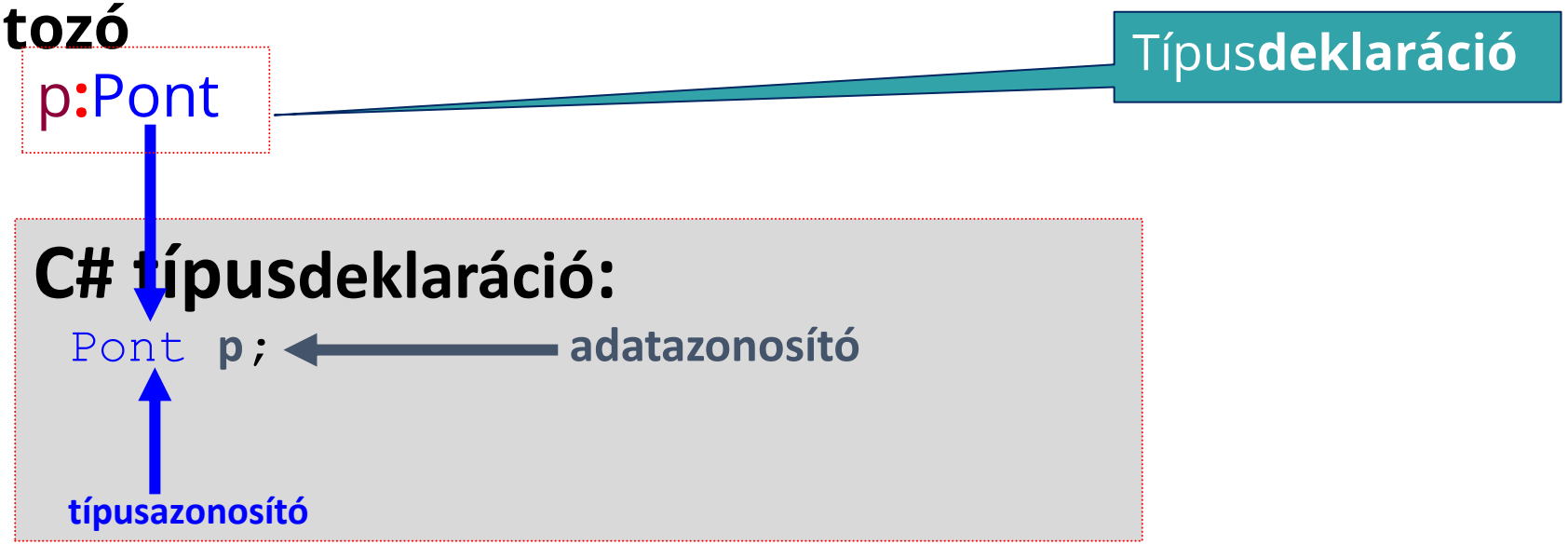
mezőazonosítók

```
struct Pont
{
    public double x;
    public double y;
}
```

Rekord – típusdeklarálás kódban

Specifikáció → algoritmus → kód:

- **Változó**



Kód

p.x≥0 és p.y≥0	p.x<0 és p.y≥0	p.x<0 és p.y<0	p.x≥0 és p.y<0
sn:=1	sn:=2	sn:=3	sn:=4

```
struct Pont {
  public double x, y;
}
static void Main(string[] args) {
  // 1. deklarálás
  //double x, y;
  Pont p;
  int sn = 0;
  // 2. beolvasás
  Console.Write("x = ");
  double.TryParse(Console.ReadLine(), out p.x);
  Console.Write("y = ");
  double.TryParse(Console.ReadLine(), out p.y);
  // 3. feldolgozás
  if (p.x >= 0 && p.y >= 0) { sn = 1; }
  else if (p.x < 0 && p.y >= 0) { sn = 2; }
  else if (p.x < 0 && p.y < 0) { sn = 3; }
  else if (p.x >= 0 && p.y < 0) { sn = 4; }
  // 4. kiírás
  Console.WriteLine("Síknegyed = {0}", sn);
}
```

```
Be: p∈Pont,
    Pont=(x:R x y:R)
Ki: sn∈N
Ef: -
Uf: ((p.x>=0 és p.y>=0) -> sn=1) és
    ((p.x<0  és p.y>=0) -> sn=2) és
    ((p.x<0  és p.y<0 ) -> sn=3) és
    ((p.x>=0 és p.y<0 ) -> sn=4)
```

Tömb

Sok adat

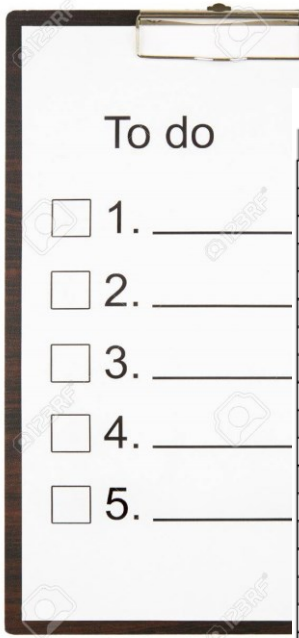
- Vannak olyan feladatok, amelyekben az adatleírás nehezen vagy egyáltalán nem végezhető el elemi típusokkal vagy rekorddal.
- Ezek, amikor **sok egyforma adat**tal dolgozunk
- Példa: melyik a legnagyobb?
  - `Be: a∈Z, b∈Z -> Ki: max∈Z`
  - `Be: a∈Z, b∈Z, c∈Z -> Ki: max∈Z`
  - `Be: a∈Z, b∈Z, c∈Z, d∈Z -> Ki: max∈Z`
- Gond:
  - ez a megoldás nem skálázódik jól a számossággal
  - az utófeltétel egyre bonyolultabb
  - ennek megfelelően az algoritmus is egyre bonyolultabb
  - változik az egész megoldás a számossággal (=új spec+alg+kód)

Tömb

- Szükségünk van egy olyan adatszerkezetre, amely
  - sok adat kezelésére alkalmas,
  - jól kezeli a bemenet változó számosságát, és
  - könnyű vele dolgozni (spec, alg, kód)
- **Tömb:**
  - *ugyanolyan funkciójú adatok sokasága*
  - szemantikus egység létrehozása
  - „funkció”: mit *jelent* az adat

Hétköznapi példák

- Amikor sok dolog van…



Tömb vs táblázatkezelő oszlopa

Excel

	A
1	zöld
2	piros
3	sárga
4	fehér
5	fekete
6	
7	

hivatkozás:
A2 → piros

Tömb

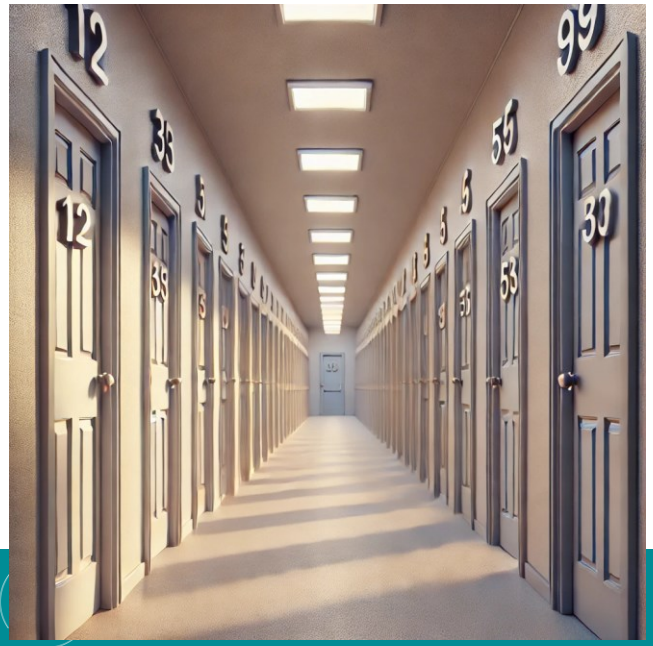
	szín
0	zöld
1	piros
2	sárga
3	fehér
4	fekete

hivatkozás:
szín[2] → sárga

Számozott felsorolás,
hozzáférés a sorszámon keresztül

Tömb: index → érték

- A tömb egy eleméhez az indexén (sorszámán) keresztül férünk hozzá: szín[2]
- Azaz az értékek *közvetlenül* nem érhetők el, csak *közvetve* az indexeken (sorszámokon) keresztül
- Az indextartomány tehát nagyon fontos!

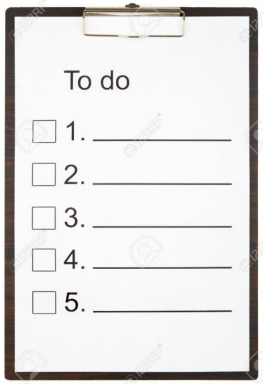


	szín
0	zöld
1	piros
2	sárga
3	fehér
4	fekete

Tömb – specifikáció

Azonos funkció → azonos halmazbeli

- **Sorozat:** azonos funkciójú elemek egymásutánja, az elemei sorszámozhatók.
- Példa a definiálásra (`Be, Ki`):
  - `teendők∈S[1..5]`
  - `totó∈K[1..14]`
  - `n∈N, vendégek∈S[1..n]`
  - `dobozok∈S[1..] ≡ dobozok∈S[]`
- Példa a használatra (=hivatkozás egy elemre, `Ef, Uf`)
  - `teendők[1]`
  - `vendégek[n-2]`



Az aktuális elemszám lekérdezhető: hossz(dobozok)

Az összes olyan véges 5 hosszú sorozat halmaza, amely a szöveg alaphalmaz elemeiből áll.

Kérdés: az elemek lehetnek sorozatok, azaz van-e sorozatok sorozata?

Tömb – algoritmus

- **Tömb:** véges hosszúságú *sorozat algoritmikus párja*, amelynek i-edik tagjával végezhetünk műveleteket.
- Példa a definiálásra:
  - `teendők:Tömb[1..5:Szöveg]`
  - `vendégek:Tömb[1..n:Szöveg]`
  - `Konst MAXN=100, Változó n:Egész vendégek:Tömb[1..MAXN:Szöveg]`
  - `dobozok:Tömb[1..:Szöveg]`
- Példa a használatra:
  - hivatkozás: `todo[2]`
  - értékadás: `vendégek[1]:="Miklós atya"`

Specifikáció:
`teendők∈S[1..5]`
`totó∈K[1..14]`
`n∈N, vendégek∈S[1..n]`
`dobozok∈S[1..]`

adott a legkisebb és a legnagyobb index, vagy az elemszám

Tömb – C# kód

Algoritmus:
- `teendők:Tömb[1..5:Szöveg]`

- Statikus tömb – ismert méret
  - `string[] teendok=new string[5];`
  - `char[] toto=new char[14];`
- Statikus tömb – max.méret
  -
```
int n;
const int MAXN = 100;
string[] vendegek = new string[MAXN];
```
- Statikus tömb – igény szerinti méret
  -
```
int n;
int.TryParse(Console.ReadLine(), out n);
string[] vendegek = new string[n];
```

Algoritmus:
- `Konst MAXN=100, Változó n:Egész`
  `vendégek:Tömb[1..MAXN:Szöveg]`

	vendégek
1	Miklós atya
2	Józsi bácsi
3	Ili néni
n= 4	Gábor barátom
5	
6	
7	
8	
9	
MAXN= 10	

	vendégek
1	Miklós atya
2	Józsi bácsi
3	Ili néni
n= 4	Gábor barátom

Algoritmus:
- `vendégek:Tömb[1..n:Szöveg]`

Tömb – algoritmus → kód

C#-ban a tömbök 0-tól indexelődnek.

1. ötlet: ne használjuk a 0. elemet!

Algoritmus		Kód	
		0	?
1	a	1	a
2	b	2	b
3	c	3	c

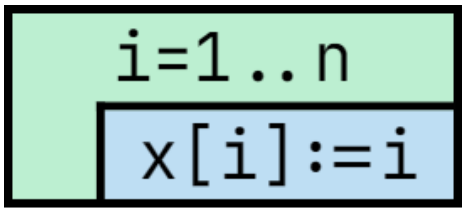
Deklarációs példa:

```
x:Tömb[1..n:Valós]
```

C# kód:

```
float[] x=new float[n+1];
```

Algoritmus:



C# kód:

```
for(int i=1;i<=n;++i){
  x[i]=i;
}
```

Tömb – algoritmus → kód

C#-ban a tömbök 0-tól indexelődnek.

2. ötlet: indexeltolás!

Algoritmus		Kód	
1	a	0	a
2	b	1	b
3	c	2	c

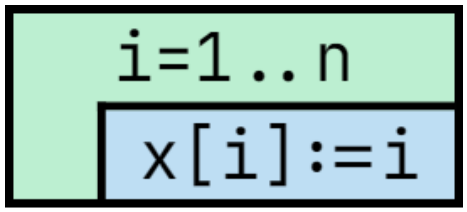
Deklarációs példa:

```
x:Tömb[1..n:Valós]
```

C# kód:

```
float[] x=new float[n];
```

Algoritmus:



C# kód:

```
for(int i=1;i<=n;++i){
  x[i-1]=i;
}
//------- vagy -------
for(int i=1-1;i<=n-1;++i){
  x[i]=i+1;
}
```

Tömb – algoritmus → kód

C#-ban a tömbök 0-tól indexelődnek.

3. ötlet: általános esetben!

Algoritmus		Kód	
e	a	0	a
e+1	b	1	b
u	c	2	c

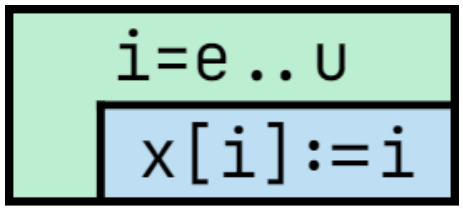
Deklarációs példa:

```
x:Tömb[e..u:Valós]
```

C# kód:

```
float[] x=new float[u-e+1];
```

Algoritmus:



C# kód:

```
for(int i=e;i<=u;++i){
  x[i-e]=i;
}
//------- vagy -------
for(int i=e-e;i<=u-e;++i){
  x[i]=i+e;
}
```

Tömb – algoritmus → kód

C#-ban a tömbök 0-tól indexelődnek.

3. ötlet: példa

Deklarációs példa:

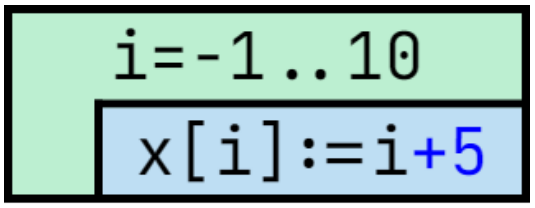
```
x:Tömb[-1..10:Valós]
```

Algoritmus		Kód	
-1	4	0	4
0	5	1	5
1	6	2	6

C# kód:

```
float[] x=new float[12];
```

Algoritmus:



C# kód:

```
for(int i=-1;i<=10;++i){
  x[i+1]=i+5;
}
//------- vagy -------
for(int i=0;i<=11;++i){
  x[i]=i+(-1)+5;
}
```

Konstans tömb

- **Specifikáció:**
  - `színek∈S[1..4]=`
    `["zöld","piros","tök","makk"]`
- **Algoritmus:**
  - `Konstans SZÍNEK:Tömb[1..4:Szöveg]=`
    `["zöld","piros","tök","makk"]`
- **Kód:**

```
string[] SZINEK = new string[4]
  { "zöld", "piros", "tök", "makk" };
// vagy
string[] SZINEK =
  { "zöld", "piros", "tök", "makk" };
```

Mátrix

- **Tömb:** azonos funkciójú elemek *egyirányú* sorozata
  - egy index egy elem kiválasztásához, pl. `x[i]`

	x
1	-4
2	2
3	5

- **Mátrix:** azonos funkciójú elemek *kétirányú* sorozata
  - két index egy elem kiválasztásához, pl. `x[i,j]`
  - specifikáció: `n∈N, m∈N, x∈Z[1..n,1..m]`
  - algoritmus: `x:Tömb[1..n,1..m:Egész]`
  - kód: `îŋʧ y ŋêx îŋʧ ŋ ŋ`

x	1	2	3
1	-4	3	2
2	2	10	11
3	5	4	-5

Példák konstans tömbökre

Feladat elágazásra – vagy más kell?

Feladat:

A japán naptár 60 éves ciklusokat tartalmaz, az éveket párosítják, s mindegyik párhoz valamilyen színt rendelnek (zöld, piros, sárga, fehér, fekete).

- 1,2,11,12, ...,51,52: zöld évek
- 3,4,13,14,...,53,54: piros évek
- 5,6,15,16,...55,56: sárga évek
- 7,8,17,18,...57,58: fehér évek
- 9,10,19,20,...,59,60: fekete évek

Tudjuk, hogy 1984-ben indult az utolsó ciklus, amely 2043-ban fog véget érni.

Írj programot, amely megadja egy M évről (1984≤M≤2043), hogy milyen színű!

Feladat elágazásra – vagy más kel

Példa: n=2024 → s=„zöld"

Specifikáció:

Be: év∈N
Sa: y∈N
Ki: s∈S
Ef: 1984<=év<=2043
Uf: y=((év-1984) mod 10) div 2 és
    y=0 -> s="zöld" és
    y=1 -> s="piros" és
    ...

- 1,2,11,12, ...,51,52: zöld évek
- 3,4,13,14,...,53,54: piros évek
- 5,6,15,16,...55,56: sárga évek
- 7,8,17,18,...57,58: fehér évek
- 9,10,19,20,...,59,60: fekete évek

év	év-1984	mod 10	div 2
1984	0	0	0
1985	1	1	0
1986	2	2	1
1987	3	3	1
1988	4	4	2
1989	5	5	2
1990	6	6	3
1991	7	7	3
1992	8	8	4
1993	9	9	4
1994	10	0	0
1995	11	1	0
1996	12	2	1
1997	13	3	1
1998	14	4	2
1999	15	5	2
2000	16	6	3
2001	17	7	3
2002	18	8	4
2003	19	9	4

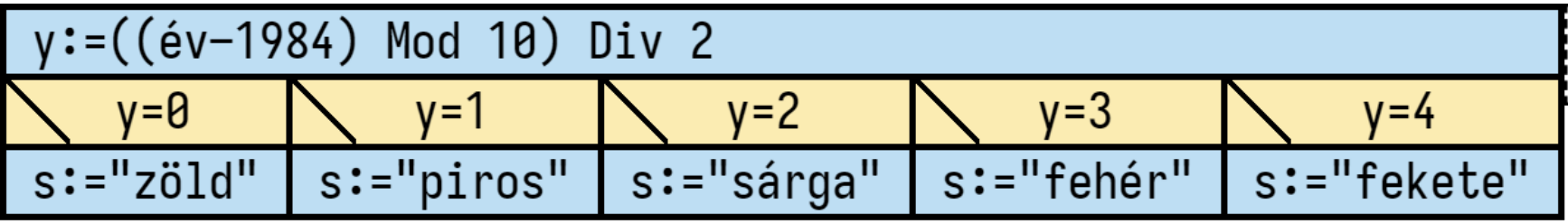
Állapottér bővítés

y	szín
0	zöld
1	piros
2	sárga
3	fehér
4	fekete

Feladat elágazásra – vagy más kell?

Segédadatnak megfelelő lokális változó deklarálása

Algoritmus:




```
Be: év∈N
Sa: y∈N
Ki: s∈S
Ef: 1984<=év<=2043
Uf: y=((év-1984) mod 10) div 2 és
    y=0 -> s="zöld" és
    y=1 -> s="piros" és
    ...
```

Kérdés:

Akkor is ezt tennénk, ha 5 helyett 90 ágat kellene írnunk?

→ **tömb**

y	szín
0	zöld
1	piros
2	sárga
3	fehér
4	fekete

Tömb

Összerendelés

y	szín
0	zöld
1	piros
2	sárga
3	fehér
4	fekete

Tömb

	szín
0	zöld
1	piros
2	sárga
3	fehér
4	fekete

hivatkozás:

szín[2] → sárga

Sorozat → tömb

	SZÍNEK
0	zöld
1	piros
2	sárga
3	fehér
4	fekete

Példa – színes évek:

A feladat specifikációjában bevezetünk egy szöveg konstansokból álló sorozatot:

```
színek∈S[0..4]=
        ["zöld","piros","sárga","fehér","fekete"]
```

Az algoritmusban tömbbel reprezentálhatjuk:

```
Konstans SZÍNEK:Tömb[0..4:Szöveg]=
        ["zöld","piros","sárga","fehér","fekete"]
```

Elágazás helyett tömb

Specifikáció (végleges):

```
Be: év∈N,
    színek∈S[0..4]=
      ["zöld","piros","sárga","fehér","fekete"]
Sa: y∈N
Ki: s∈S
Ef: 1984<=év<=2043
Uf: y=((év-1984) mod 10) div 2 és
    s=színek[y]
```

```
Be: év∈N
Sa: y∈N
Ki: s∈S
Ef: 1984<=év<=2043
Uf: y=((év-1984) mod 10) div 2 és
    y=0 -> s="zöld" és
    y=1 -> s="piros" és
    ...
```

	SZÍNEK
0	zöld
1	piros
2	sárga
3	fehér
4	fekete

Elágazás helyett tömb

Adatreprezentálás:

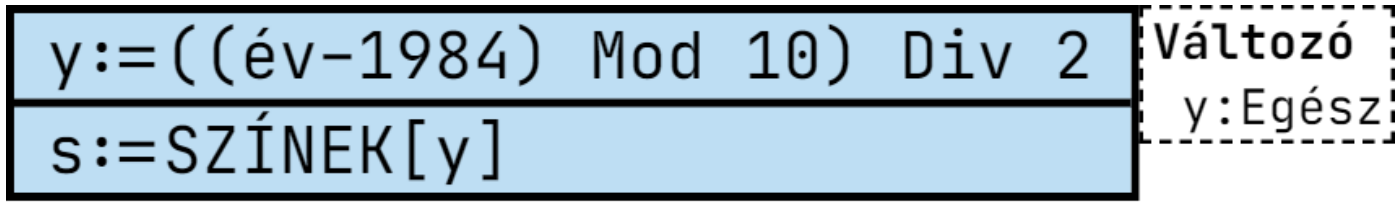
Programparaméterek deklarálása

```
Változó
   év:Egész
   s:Szöveg
Konstans
   SZÍNEK:Tömb[0..4:Szöveg]=
     ("zöld","piros","sárga","fehér","fekete")
```

```
Be: év∈N,
    színek∈S[0..4]=
      ["zöld","piros","sárga","fehér","fekete"]
Sa: y∈N
Ki: s∈S
```

Algoritmus:

```
Uf: y=((év-1984) mod 10) div 2 és
    s=színek[y]
```



Konstans tömb alkalmazása

Példa: n=42 → s=„negyvenkettő”

Feladat:

Írj programot, amely egy 1 és 99 közötti számot betűkkel ír ki!

Specifikáció:

Leglogikusabb helyre téve.
Az algoritmus szempontjából „adottság”, azaz bemenet...

```
Be: n∈N,
    egyes∈S[0..9]=["","egy",...,"kilenc"],
    tizes∈S[0..9]=["","tizen",...,"kilencven"]
Ki: s∈S
Ef: 1<=n<=99
Uf: n=10 -> s="tíz" és
    n=20 -> s="húsz" és
    (n≠10 és n≠20) ->
        s=tizes[n div 10]+egyes[n mod 10]
```

Konstans tömb alkalmazása

Algoritmus:

```
Változó n:Egész
Konstans EGYES:Tömb[0..9:Szöveg]=
                ("","egy",…,"kilenc")
         TIZES:Tömb[0..9:Szöveg]=
                ("","tizen",…,"kilencven")
Változó s:Szöveg
```

```
Be: n∈N,
    egyes∈S[0..9]=["","egy",...,"kilenc"],
    tizes∈S[0..9]=["","tizen",...,"kilencven"]
Ki: s∈S
```

```
Uf: n=10 -> s="tíz" és
    n=20 -> s="húsz" és
    (n≠10 és n≠20) ->
        s=tizes[n div 10]+egyes[n mod 10]
```

n=10	n=20	n≠10 és n≠20
s:="tíz"	s:="húsz"	s:=TIZES[n div 10]+ EGYES[n mod 10]

Konstans tömb alkalmazása

Példa:
h=9 → s="szeptember"

Feladat:

Írj programot, amely egy hónapnévhez a sorszámát rendeli!

Specifikáció:

Be: h∈S,
hónév∈S[1..12]=
["január",...,"december"]
Ki: s∈N
Ef: ∃i∈[1..12]:(h=hónév[i])
Uf: 1<=s<=12 és HÓNÉV[s]=h

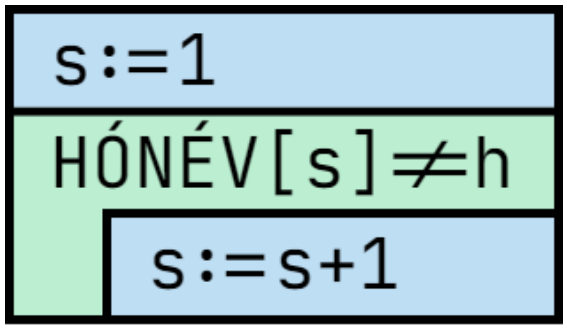
h∈hónév

	HÓNÉV
1	január
2	február
3	március
4	április
5	május
6	június
7	július
8	augusztus
9	szeptember
10	október
11	november
12	december

Konstans tömb alkalmazása

Algoritmus:

```
Változó  h:Szöveg, s:Egész
Konstans HÓNÉV:Tömb[1..12:Szöveg]=
           ("január",...,"december")
```




```
Be: h∈S,
    hónév∈S[1..12]=
      ["január",...,"december"]
Ki: s∈N
Ef: ∃i∈[1..12]:(h=hónév[i])
Uf: 1<=s<=12 és HÓNÉV[s]=h
```

Kérdés: mi lenne, ha az előfeltétel nem teljesülne?
Futási hiba? Végtelen ciklus?

	HÓNÉV
1	január
2	február
3	március
4	április
5	május
6	június
7	július
8	augusztus
9	szeptember
10	október
11	november
12	december

Konstans tömb – mit tárolunk?

Példa:
h=9, n=18 → s=261

Feladat:

Egy nap a nem szökőév hányadik napja?

Specifikáció$_1$:

```
Be: h∈N, n∈N,
    HÓ∈N[1..12]=[31,28,31,...,31]
Ki: s∈N
Ef: 1<=h<=12 és 1<=n<=HÓ[h]
Uf: s=SZUMMA(i=1..h-1, HÓ[i]) + n
```

	HÓ
1	31
2	28
3	31
4	30
5	31
6	30
7	31
8	31
9	30
10	31
11	30
12	31

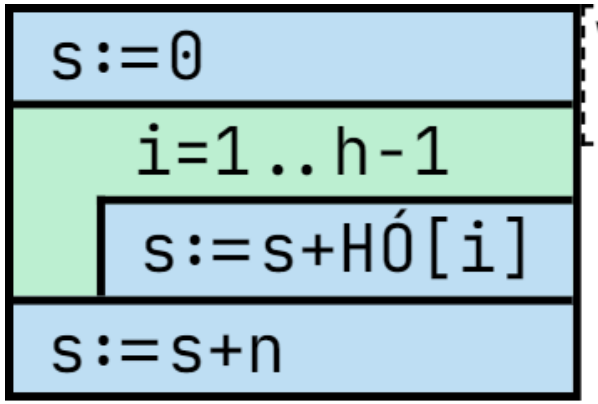
Konstans tömb – mit tárolunk?

Példa:
h=9, n=18 → s=261

Algoritmus:

Programparaméterek deklarálása

```
Változó h,n,s:Egész
Konstans HÓ:Tömb[1..12:Egész]=(31,28,31,…,31)
```




```
Változó
 i:Egész
```

```
Be: h∈N, n∈N,
    HÓ∈N[1..12]=[31,28,31,...,31]
Ki: s∈N
Ef: 1<=h<=12 és 1<=n<=HÓ[h]
Uf: s=SZUMMA(i=1..h-1, HÓ[i]) + n
```

Lokális változó deklarálása

	HÓ
1	31
2	28
3	31
4	30
5	31
6	30
7	31
8	31
9	30
10	31
11	30
12	31

Megjegyzés: szökőév esetén h≥3 esetén s-et 1-gyel meg kellene növelni! (És az előfeltétel is módosul.)

Konstans tömb – mit tárolunk?

Példa:
h=9, n=18 → s=261

Egy másik megoldás:

Tároljuk minden hónapra, hogy az előző hónapokban összesen hány nap van!

Specifikáció$_2$:

```
Be: h∈N, n∈N,
    HÓ∈S[1..12]=(0,31,59,90,...,334)
Uf: s=HÓ[h] + n
```

Kérdés: Ez jobb megoldás? Mi lesz az előfeltétellel?

	HÓ
1	0
2	31
3	59
4	90
5	120
6	151
7	181
8	212
9	243
10	273
11	304
12	334

Összefoglalás

Összefoglalás

- Adat
  - egy-szerű: elemi
  - több különböző: rekord
  - több egyforma: tömb
- Vezérlési szerkezetek
  - Szekvencia: `és`
  - Elágazás: `->`
  - Ciklus: `∀, ∃, Σ`

- Feladatmegoldás
  1. Példa
  2. Specifikáció (← példa)
     1. Adatok (Be, Ki)
     2. Megszorítás (Ef → Be)
     3. Összefüggés (Uf)
  3. Adat → Változó
  4. Algoritmus (← Uf)
  5. Kód (← Spec + Alg)

Megfeleltetések

Példa adat	Specifikáció halmaz	Algoritmus típus	Kód type
3	N	Egész	int
-3	Z	Egész	int
3,3	R	Valós	double
igaz	L	Logikai	bool
"alma"	S	Szöveg	string
"a"	K	Karakter	char
(név:"Győző", jegy: 5)	Név x Jegy, S x N	Rekord	struct
[3, 5, -6, 2]	Z[1..n]	Tömb	int[]

Ellenőrző kérdések

Ellenőrző kérdések

- Mikor érdemes rekordtípust használni?
- Milyen adatszerezettel írjuk le, ha több különböző funkciójú adatot szeretnénk egységbe foglalni?
- Milyen adatszerkezettel írjuk le, ha több ugyanolyan funkciójú adatot szeretnénk egységbe foglalni?
- Hogyan hívjuk a tömböt a specifikációban?