



# Programozás I.

3.

Blokkok , logikai kifejezések,  
elágazások, ciklusok,  
tömbök

1

---

Blokkok

# Blokk létrehozása

- A blokk a forráskód egy, valamilyen szempontból összefüggőnek tekinthető része.
- A blokk '{'-tól '}'-ig tart
- A blokk nem utasítás, nem kell a végére ';'
- Pl.:

```
{  
    int a;  
    scanf("%d", &a);  
    printf("%d\n", a*6+2);  
}
```

# Blokkok használata

- Blokkon belül is lehet blokk

```
{  
    int b;  
    {  
        int a;  
        scanf("%d", &a);  
        printf("%d\n", a*6+2);  
    }  
}
```

# Változók élettartama

- Egy blokkon belül létrehozott változó a blokk végén megszűnik

```
{
```

```
    int a; // az a változó létrehozása
```

```
    scanf("%d", &a); // és használata
```

```
    printf("%d\n", a*6+2); // itt még létezik
```

```
} // itt megszűnik
```

```
printf("%d\n", a*2); // így itt már nem létezik
```

# Változók azonos névvel

- Különböző blokkokban lehetnek azonos nevű változók:

```
int a=3; // a változó a külső blokkban
{
    int a=2; // a változó a belső blokkban
    printf("%d\n", a); // ez a belső a, értéke 2
}
printf("%d\n", a); // ez a külső a, értéke 3
```

# Elágazás

# Mi az elágazás

- Az elágazás egy olyan blokk, melynek tartalma csak akkor hajtódik végre, ha egy megadott feltétel teljesül.
- Pl.: Ha a szám páros, osszuk le kettővel.



# Logikai kifejezések

- Az elágazások feltétele egy logikai kifejezés, amely vagy igaz, vagy hamis.
- Pl.: A szám páros.
- Ezeket a kifejezéseket a fordító számára érthető formában kell megadni.

# Egyszerű logikai kifejezések

- $a==b$  // igaz, ha **a** és **b** értéke megegyezik, egyébként hamis
- $a!=b$  // igaz, ha **a** és **b** értéke különbözik, egyébként hamis
- $a<b$  // igaz, ha **a** értéke kisebb, mint **b** értéke, egyébként hamis
- $a>b$  // igaz, ha **a** értéke nagyobb, mint **b** értéke, egyébként hamis
- $a\leq b$  // igaz, ha **a** értéke kisebb, vagy egyenlő, mint **b** értéke, egyébként hamis
- $a\geq b$  // igaz, ha **a** értéke nagyobb, vagy egyenlő, mint **b** értéke, egyébként hamis

# Összetett logikai kifejezések

- Amik más logikai kifejezésekből jönnek **tagadással**, **és**-, **vagy**-, illetve **kizáró vagy** műveletekkel.
- Pl.: A szám nagyobb, mint 2, **és** kisebb mint 6.
- Pl.: A szám kisebb, mint 4, **vagy** nagyobb, mint 12.

# Összetett logikai kifejezések

- $(k1) \ \&\& \ (k2)$  //  $k1$  **és**  $k2$ : igaz, ha mind  $k1$  és  $k2$  is igaz
- $(k1) \ || \ (k2)$  //  $k1$  **vagy**  $k2$ : igaz, ha a kettő közül legalább az egyik igaz
- $(k1) \ \wedge \ (k2)$  //  $k1$  **xor**  $k2$  (kizáró vagy): igaz ha pontosan az egyik igaz
- $!(k1)$  // **nem**  $k1$ : igaz, ha  $k1$  hamis

# Összetett logikai kifejezések

- Pl.:
- $((a > b) \ \&\& \ (b \neq 3)) \ || \ (\!((c > 2) \ || \ (c < 3)) \ )$
- A műveleti sorrend miatt bizonyos zárójelek elhagyhatók úgy, hogy a kifejezés jelentése nem változik:
- $(a > b \ \&\& \ b \neq 3) \ || \ !(c > 2 \ || \ c < 3)$

# Oszthatóság

- Maradékos osztás: % operátor
  - $a \% 3$  // Visszaadja  $a$  3-mal való osztási maradékát (Pl.:  $5 \% 3 = 2$ )
- A maradék 0, ha osztható vele
- Pl.: Az  $a$  szám osztható 5-tel:
  - $(a \% 5) == 0$
- Pl.: Az  $a$  szám nem osztható 7-tel:
  - $(a \% 7) != 0$

# Elágazás létrehozása

- C-ben elágazást az **if** kulcsszóval hozhatunk létre
- A feltétel a kulcsszó után zárójelben jön
- Majd a végrehajtandó utasítások **blokkja**
- Pl.: Ha az **a** szám osztható 2-vel, adjunk hozzá 5-öt:

```
if (a%2==0)
{
    a+=5;
}
```

## Ha ..., egyébként ...

- Sok esetben előfordul, hogy nem csak akkor kell műveleteket végezni, ha a feltétel igaz, hanem akkor is, ha hamis, viszont ezek a műveletek eltérhetnek.
- Pl.: Ha  $a$  szám osztható 2-vel, akkor adjunk hozzá 5-öt, egyébként vonjunk ki 4-et.



## if {...} else {...}

- A feltétel hamis ágát az **else** kulcsszóval tudjuk definiálni, ami után szintén egy blokk következik:

```
if (a%2==0)
{
    a+=5;
}
else
{
    a-=4;
}
```

# Egyszerűsítések

- Amennyiben a feltétel bármely ágában lévő blokk csak egy utasításból áll, magát a blokkot nem kell külön jelölni:

```
if (a%2==0)
```

```
    a+=5;
```

```
else
```

```
    a-=4;
```

# Egyszerűsítések

- Ha a logikai kifejezés az  $a==0$  vagy  $a!=0$  feltételt használja, nem kell az összehasonlítást kiírni. Ha egy számot logikai kifejezésként értelmezünk, amennyiben a szám értéke 0, akkor a jelentése hamis, egyébként pedig igaz.
  - `if (a)` // ugyanaz, mint: `if (a!=0)`
  - `if (!a)` // ugyanaz, mint: `if (a==0)`
  - `if (!(a%2))` // ugyanaz, mint: `if (a%2==0)`

## Egyébként ha

- Egy feltételnek akár kettőnél több ága is lehet az **else if** használatával:

```
if (a<2) {...}
```

```
else if (a<4) {...}
```

```
else if (a<10) {...}
```

```
else {...}
```

# Gyakorló feladatok

## ◦ 2.1 Feladat

- Olvasson be egy számot
- Írassa ki a szám abszolútértékét

## ◦ 2.2 Feladat

- Olvasson be két számot
- Számolja ki a két szám hányadosának felső egész részét

## ◦ 2.3 Feladat

- Olvasson be három számot
- Tárolja el egy új változóban a három szám közül a legnagyobbat
- Írassa ki ezt a számot

# Ciklus

# Mi a ciklus?

- A ciklus egy olyan kódrészlet (blokk), amely egymás után többször is lefut. A futás mennyisége egy feltételtől függ, amely minden végrehajtás előtt (vagy után) eldönti, szükség van-e a következő végrehajtásra.


## while ciklus

- A **while** kulcsszó használatával előltesztelős ciklust lehet létrehozni.
- A ciklusmag (a blokk) minden lefutása előtt a program ellenőrzi a feltételt.
- Amennyiben a feltétel igaz, a ciklusmag lefut még egyszer, egyébként a ciklus leáll és a program a következő sornál folytatódik.




# while ciklus szintaktika

Bentmaradási  
feltétel

**while** (...) 

{

...  Ciklusmag

}

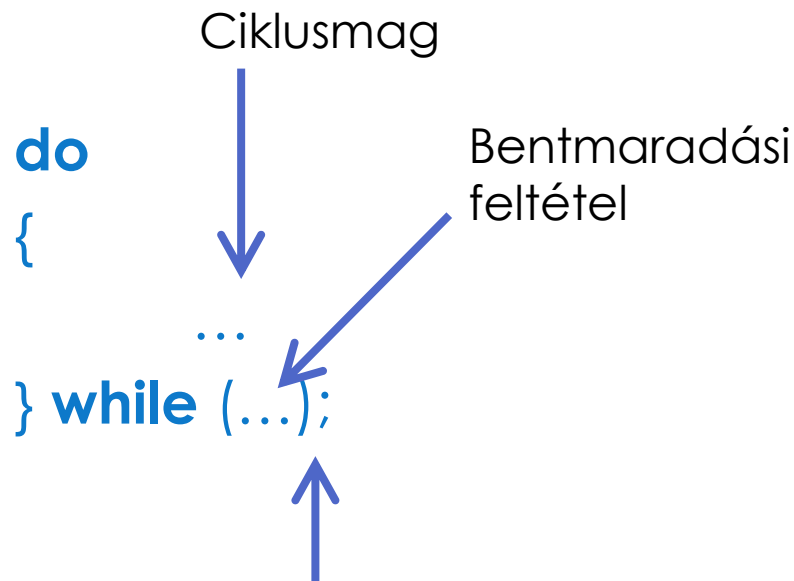
## while ciklus példa

```
int a=0;
scanf("%d", &a);
while (a%2==0) // amíg a osztható 2-vel
{
    a/=2; // addig osszuk le kettővel
}
printf("%d\n", a);
```

## do-while ciklus

- A while ciklus hátultesztelős változatához a **do** kulcsszót használhatjuk.
- Mivel hátultesztelős ciklus, így a ciklusmag egyszer lefut, és csak azután ellenőrzi a program, hogy fusson-e le még egyszer.

# do-while ciklus szintaktika



**FIGYELEM!** Ide kell a ';', mert nem a '}' jelnél van a ciklus vége, hanem a feltétel után.

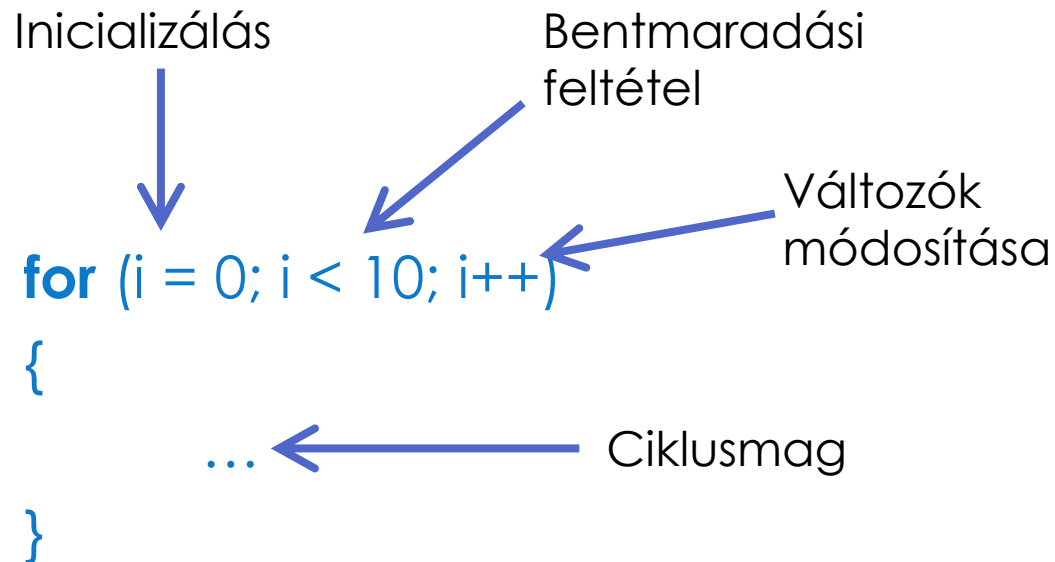
## do-while ciklus példa

```
int a;  
do // csináljuk azt, hogy  
{  
    scanf("%d", &a); // bekérünk egy számot  
    printf("%d\n", a); // majd kiíratjuk  
} while (a<10); // és ha a szám kisebb, mint 10,  
akkor ismételjük
```

## for ciklus

- Szintén előtesztelős a **for** kulcsszóval létrehozható ciklus.
- Ennek tulajdonsága, hogy jellemzően ciklusváltozót használó ciklusoknál alkalmazzuk.
- A szintaktika nem csak a bentmaradási feltételt tartalmazza, hanem az inicializálást és a ciklusváltozó módosítását is.

# for ciklus szintaktika



## for ciklus példa

```
int i;  
for (i=0; i<5; i++) // csináljuk meg 5-ször, hogy  
{  
    int a;  
    scanf("%d", &a); // bekérünk egy számot  
    printf("%d\n", a); // és kiírjuk  
}
```



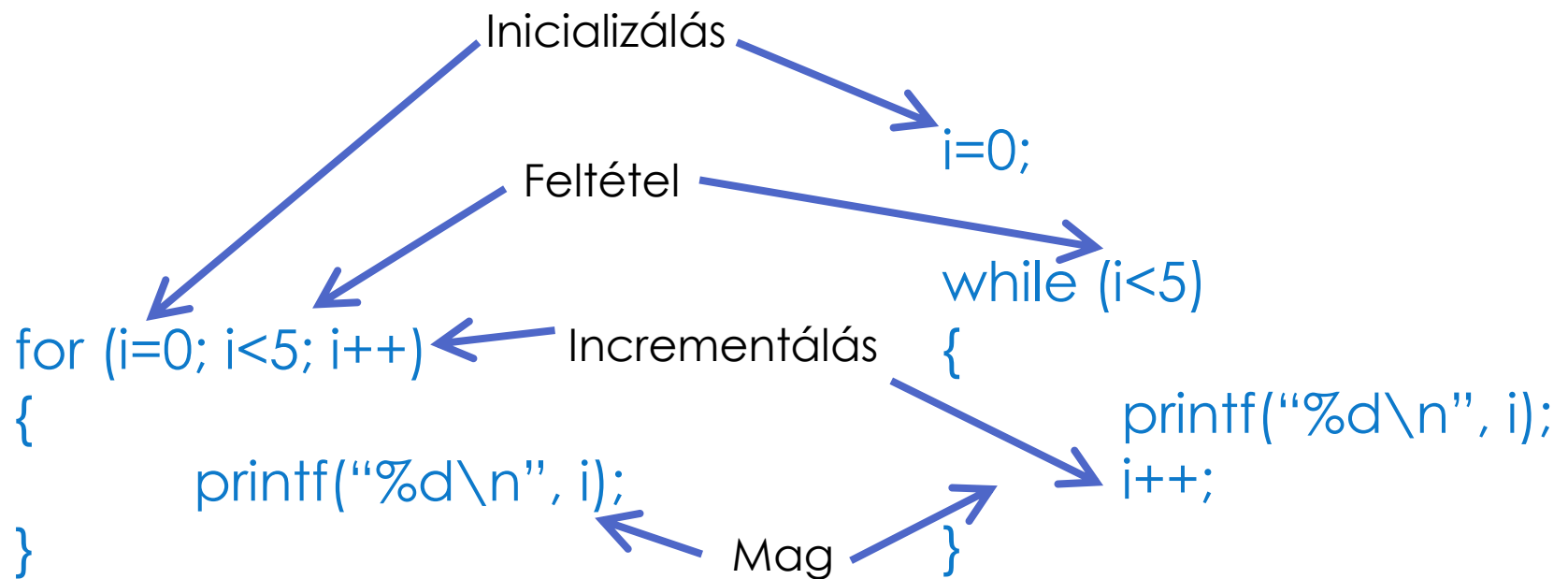
## for ciklus több művelettel

- A for ciklus megadásában az inicializáló és incrementáló rész több utasítást is tartalmazhat, ‘,’-vel elválasztva:

```
int i, j;  
for (i=0, j=0; i<5; i++, j+=3)  
{  
    printf("%d %d\n", i, j);  
}
```

# for ciklus ↔ while ciklus

- A for és a while ciklusok egymásnak megfeleltethetők:



# Egyszerűsítés

- A blokk itt is elhagyható, ha csak egy utasítás van:

```
int i, j;
```

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

```
    for (j=0; j<10; j++)
```

```
        if (i<j)
```

```
            printf("%d*%d=%d\n", i, j, i*j);
```

# Gyakorló feladatok

## ◉ 2.6 Feladat

- ◉ Ciklus segítségével ötször olvasson be egy számot, majd írja ki annak négyzetét

## ◉ 2.7 Feladat

- ◉ Olvasson be egy számot addig, amíg nem kap pozitív értéket

## ◉ 2.9 Feladat

- ◉ Olvasson be számokat addig, amíg a beolvasott számok értékének összege kisebb, mint 100
- ◉ Ha a beolvasott szám osztható kettővel, de hárommal nem, írassa ki az értékét

# Matematikai függvények

- Matematikai függvények használatához:
  - `#include <math.h>`
- Gyökvonás:
  - `sqrt( szám )`  
`double c = sqrt( a*a + b*b );`
- Abszolútérték:
  - `fabs( szám )`  
`double tavolsag = fabs( a - b );`
- Fordítás:
  - `-lm` kapcsoló  
`gcc -o Program main.c -lm`

# Tömbök

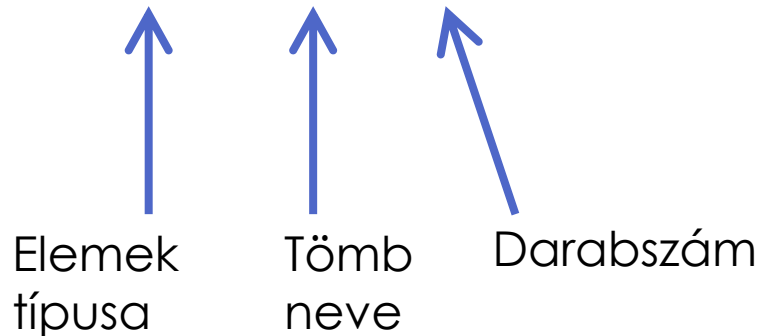
# Mi a tömb?

- A tömb több azonos típusú változót képes tárolni egymás mellé téve.
- A tömbnek csak a változók típusát és számát kell megadni.
- Akkor jó, ha több, ugyanolyan célt szolgáló értéket szeretnénk eltárolni.
  - Pl.: 10 darab hosszúságérték tárolását kellene megoldani.
  - Lehetne 10 külön változó, de ha később kiderül, hogy inkább 20 kell, akkor gondban vagyunk.

# Tömb létrehozása

- Típus + név + darabszám:

- `int tomb[10];`



0x20	0x24	0x28	0x2C	0x30	0x34	0x38	0x3C	0x40	0x44
142	-5	6	897	-23	12	42	-11	-9	123



# Tömb indexelése

- A tömb elemeit külön el lehet érni, mintha különböző változók lennének.
- A változók egymás mellett helyezkednek el sorrendben.
- Minden elemnek van egy indexe a tömbön belül (hányadik a sorban).
- Az első elem indexe mindig 0.
- Az utolsó elem indexe  $n$  elem esetén  $n-1$ .

# Elemek elérése

- Adott egy tömb:
  - `int tomb[10];`
- Az első (0-s indexű) elem elérése:
  - `tomb[0]=42;`
  - `scanf("%d", &tomb[0]);`
  - `printf("%d\n", tomb[0]);`
  - `int a=tomb[0];`
- Egy tetszőleges (pl. 4-es indexű) elem elérése:
  - `tomb[4]=11;`

# Tömb indexelése

- Egy  $n$  elemű tömb esetén az indexek 0-tól  $n-1$ -ig mennek. A program nem veszi észre, ha érvénytelen a megadott index, de a program futása helytelen lesz tőle.

```
int tomb[10];
```

tomb[0] 

tomb[4] 

tomb[9] 

tomb[10] 

tomb[-1] 

# Elemek elérése

- Az eléréshez megadott index egy másik változó, vagy akár egy kifejezés is lehet:

```
int tomb[10];
```

```
int i=5;
```

```
tomb[i]=3;
```

```
tomb[i+1]=5;
```

```
tomb[2*i-1]=5;
```

0x20   0x24   0x28   0x2C   0x30   0x34   0x38   0x3C   0x40   0x44

142	-5	6	897	-23	12	42	-11	-9	123
-----	----	---	-----	-----	----	----	-----	----	-----

# Elemek elérése

- Változók segítségével végig tudjuk járni a tömb elemeit:

```
int tomb[10], i;  
for (i=0; i<10; i++)  
{  
    tomb[i]=i*i*i;  
}
```

# Gyakorló feladatok

## ◦ 3.1 Feladat

- Hozzon létre egy ötelemű tömböt
- Olvasson be egymás után öt számot és tárolja el a tömbben
- Írassa ki a tömb elemeit

## ◦ 3.2 Feladat

- Hozzon létre egy hételemű tömböt
- Töltse fel a tömböt billentyűzetről beolvasott értékekkel
- Szorozza meg a tömb minden elemét annak indexével
- Írassa ki a tömb elemeit

# String

- A szöveg reprezentálása egy karakter tömb segítségével valósítható meg
- A karaktersorozat végét egy '\0' speciális karakter jelzi
- A karaktertömb mérete a karaktersorozat hosszánál legalább egyel nagyobb kell, hogy legyen

0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19

'a'	'l'	'm'	'a'	'f'	'a'	'\0'			
-----	-----	-----	-----	-----	-----	------	--	--	--

# String

```
char text[10] = { 'a', 'l', 'm', 'a', 'f', 'a', '\0' };  
scanf("%s", text);           //ide nem kell & jel  
printf("%s", text);         //6 karaktert jelenít meg
```

0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19

'a'	'l'	'm'	'a'	'f'	'a'	'\0'			
-----	-----	-----	-----	-----	-----	------	--	--	--



# Gyakorló feladatok

## ◉ 3.6 Feladat

- ◉ Olvasson be egy legfeljebb 30 karakter hosszú városnevet
- ◉ Jelenítse meg a beolvasott városnevet

## ◉ 3.7 Feladat

- ◉ Olvasson be egy vezetéknév- és keresztnévet
- ◉ Jelenítse meg a beolvasott nevet a következő formátumban:
  - ◉ keresztnév, vezetéknév