Feladat 1: Próbáljuk ki a következő kódot és magyarázzuk meg a hiba okát.

```
#include<stdio.h>
int main()
{
   {
     int a = 10, b = 20;
   }
   {
     printf("a = %d, b = %d\n", a, b);
   }
}
```

Megoldás: az *a* és *b* változók nem léteznek a második blokkban.

Feladat 2: Figyeljük meg, hogyan kezeli a **C** nyelv a blokkokat. Két kapcsos zárójel közötti utasításokat 1 blokknak kell tekinteni. Mi lesz a kimenete a következő kódnak és miért?

```
#include <stdio.h>
int main()
{
    int n = 1;
    {
        int n = 2;
        printf("%d\n", n);
    }
    printf("%d\n", n);
}
```

Megoldás: hiába azonosak a változónevek *n*, a C nyelv a blokkon belül egyedileg kezeli, így a külső *n* más értéket fog tartalmazni.

Feladat 3: Miért nem növekszik az x minden ciklusban?

```
#include <stdio.h>
int main() {
    for (int i = 0; i < 3; i++) {
        int x = 0;
        x++;
        printf("%d ", x);
    }
```

```
    return 0;
}
```

Megoldás: mert a ciklusban int x = 0; mindig újra inicializálja.

Feladat 4: Milyen sorrendben írja ki a számokat?

```
#include <stdio.h>
int main() {
    int x = 1;
    {
        x = 2;
        {
            int x = 3;
            printf("%d ", x);
        }
        printf("%d ", x);
    }
    printf("%d\n", x);
}
```

Megoldás: 3, 2, 2

Feladat 5: Miért kapunk hibás eredményt?

```
#include <stdio.h>
int main() {
    int n = 5;
    {
        int n = n + 1;
        printf("%d\n", n);
    }
}
```

Megoldás: undefined behavior (nem definiált viselkedés). A C szabályai szerint az új változó inicializálása előtt még nem létezik, tehát a *n* a jobb oldalon még nem a külső *n*-t, hanem az éppen most definiálás alatt álló (még nem inicializált) *n*-t jelenti

---

Feladat 6: Mi lesz a két kiírt szám és miért?

```
#include <stdio.h>
int main() {
    int a = 5;
    int b = 10;
    {
        int b = a + 1;
        printf("%d ", b);
    }
    printf("%d\n", b);
}
```

Megoldás: 6, 10

Feladat 7: Miért változik meg az a értéke, pedig a módosítás egy belső blokkban történt?

```
#include <stdio.h>
int main() {
    int a = 5;
    {
        int *p = &a;
        *p = 10;
    }
    printf("%d\n", a);
}
```

Megoldás: a pointer *p* közvetlenül a memóriába ír.

Feladat 8: Melyik sor melyik x-re hivatkozik?

```
#include <stdio.h>
int x = 100;

int main() {
    int x = 10;
    {
        extern int x;
        printf("%d ", x);
    }
    printf("%d\n", x);
```

```
}
```

Megoldás: extern int x; a globális változóra vonatkozik

Ciklusok a C nyelvben

A ciklus segítségével egy utasítást vagy egy egész blokkot többször is végrehajthatunk. Az ismételt részt **ciklusmagnak** nevezzük. A C nyelvben három ciklusutasítás van: `for`, `while` és `do … while`.

Feladat 1: Milyen számokat ír ki a program? Figyeljük meg, mikor vizsgálja meg a feltételt, és mikor növeli az *i* értékét.

```
#include <stdio.h>
int main(void)
{
    for (int i = 1; i <= 4; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

Megoldás: a kimenet `1 2 3 4`. Az `int i = 1` csak egyszer fut le. A program minden kör előtt megvizsgálja az `i ⇐ 4` feltételt. Ha igaz, kiírja az *i* értékét, majd az `i++` eggyel növeli azt. Amikor az *i* értéke 5 lesz, a feltétel már hamis, így az 5-öt nem írja ki.

Feladat 2: A programnak az 1-től 5-ig terjedő egész számok összegét kellene kiszámítania. Miért csak 10-et ír ki? Javítsuk ki a ciklust egyetlen jel hozzáadásával!

```
#include <stdio.h>
int main(void)
{
    int osszeg = 0;

    for (int i = 1; i < 5; i++) {
        osszeg += i;
    }
    printf("%d\n", osszeg);
    return 0;
}
```

Megoldás: az `osszeg += i` hozzáadja az *i* aktuális értékét az összeghez. Az `i < 5` feltétel miatt csak az 1, 2, 3 és 4 kerül az összegbe. Ha a feltételt `i <= 5` alakra javítjuk, az 5 is bekerül, így az eredmény 15 lesz.

Feladat 3: Egy tartályban 12 liter víz van. Minden lépésben 3 litert engedünk ki belőle. Milyen értékeket ír ki a program? Megjelenik-e a 0 is?

```
#include <stdio.h>
int main(void)
{
    int liter = 12;

    while (liter > 0) {
        printf("%d ", liter);
        liter -= 3;
    }
    return 0;
}
```

Megoldás: a kimenet `12 9 6 3`. A kiírás mindig a víz leengedése előtt történik. Az utolsó körben a program kiírja a 3-at, majd 0-ra csökkenti a *liter* értékét. A következő feltételvizsgálat már hamis eredményt ad. Ha a maradék vízmennyiséget szeretnénk látni minden leengedés után, cseréljük fel a ciklusmag két utasítását. Ekkor a kimenet `9 6 3 0`.

Feladat 4: Mindkét ciklus ugyanazt a feltételt vizsgálja, és az *n* értéke végig 0. Melyik szöveg jelenik meg a képernyőn?

```
#include <stdio.h>
int main(void)
{
    int n = 0;

    while (n > 0) {
        printf("A\n");
    }

    do {
        printf("B\n");
    } while (n > 0);

    return 0;
}
```

Megoldás: csak a B jelenik meg. A `while` a ciklusmag előtt vizsgálja meg a feltételt (**elöltesztelő ciklus**), ezért az első blokk egyszer sem fut le. A `do … while` a ciklusmag után ellenőrzi a feltételt (**hátultesztelő ciklus**), így a második blokk egyszer mindenképpen lefut. Figyeljük meg a pontosvesszőt a `do … while` végén!

Feladat 5: A programnak addig kellene növelnie a hőmérsékletet, amíg az el nem éri a 25 fokot. Miért írja ki újra és újra ugyanazt a számot? Először futtatás nélkül keressük meg a hibát!

```
#include <stdio.h>
int main(void)
{
    int homerseklet = 20;

    while (homerseklet < 25)
        printf("%d\n", homerseklet);
        homerseklet++;

    return 0;
}
```

Megoldás: kapcsos zárójelek nélkül csak a közvetlenül következő utasítás tartozik a ciklushoz. A behúzás megtévesztő: a `homerseklet++` már a ciklus után áll. A hőmérséklet így végig 20 marad, a feltétel mindig igaz, és **végtelen ciklus** alakul ki. A javított ciklus 20-tól 24-ig írja ki az értékeket:

```
    while (homerseklet < 25) {
        printf("%d\n", homerseklet);
        homerseklet++;
    }
```

Feladat 6: Három mérési ponton két-két mérést végzünk. Hányszor fut le a `printf`, és milyen sorrendben jelennek meg a mérések?

```
#include <stdio.h>
int main(void)
{
    for (int pont = 1; pont <= 3; pont++) {
        for (int meres = 1; meres <= 2; meres++) {
            printf("P%d/M%d ", pont, meres);
        }
        printf("\n");
```

```
    }
    return 0;
}
```

Megoldás: a mérés azonosítóját kiíró `printf` összesen $3 \times 2 = 6$ alkalommal fut le. Minden új mérési pontnál elölről indul a belső ciklus, ezért a *meres* ismét 1-ről kezdődik. A sortörés a belső ciklus után áll, így minden mérési pont külön sorba kerül:

```
P1/M1 P1/M2
P2/M1 P2/M2
P3/M1 P3/M2
```

Feladat 7: Az alábbi programban mit ír ki a ciklus, és mi lesz az *i* értéke a ciklus után?

```
#include <stdio.h>
int main(void)
{
    int i;

    for (i = 1; i <= 10; i++) {
        if (i == 4) {
            break;
        }
        printf("%d ", i);
    }
    printf("\ni = %d\n", i);
    return 0;
}
```

Megoldás: az első sorban `1 2 3`, a másodikban `i = 4` jelenik meg. Amikor az *i* eléri a 4-et, a `break` azonnal kilépteti a programot a ciklusból. Ebben a körben már sem a cikluson belüli kiírás, sem az `i++` nem fut le. Az *i* változót a ciklus előtt deklaráltuk, ezért utána is kiírhatjuk az értékét.

Feladat 8: Mi változik, ha a `break` helyett `continue` utasítást használunk? Kimarad-e az 5 is a kimenetből?

```
#include <stdio.h>
int main(void)
{
    for (int i = 1; i <= 6; i++) {
        if (i == 4) {
            continue;
```

```
        }
        printf("%d ", i);
    }
    return 0;
}
```

Megoldás: a kimenet `1 2 3 5 6`. Amikor az *i* értéke 4, a `continue` átugorja a ciklusmag hátralévő részét, így a kiírást is. A `for` ciklus léptetése ilyenkor is megtörténik: az `i++` 5-re növeli az *i* értékét. A következő körben tehát az 5 már megjelenik.

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:szamitastechnika:blokkok_-_ciklusok

Last update: **2026/11/03 19:16**