Mire használható a `sizeof`?

1. Típusok méretének lekérdezése

Ez a leggyakoribb:

```
sizeof(int)           // pl. 4
sizeof(double)        // pl. 8
```

---

2. Változók méretének lekérdezése

```
int x;
sizeof(x);            // ugyanaz, mint sizeof(int)
```

A fordító ebből tudja, mekkora hely kell a memóriában.

---

3. Tömbök elemszámának meghatározása

Ez **nagyon fontos és hasznos**!

```
int tomb[10];
int meret = sizeof(tomb) / sizeof(tomb[0]);  // = 10
```

Így megtudod, hány elemből áll egy **statikus tömb**, még akkor is, ha nem tudod fejből.

---

4. Struktúrák méretének lekérdezése

```
struct Ember {
    int kor;
    double magassag;
};

printf("Méret: %zu\n", sizeof(struct Ember));
```

Ez segít optimalizálni, vagy ha fájlba írsz binárisan adatot (pl. mentés, hálózati adatcsomag).

---

5. Memóriafoglalás (malloc) esetén

```
int *tomb = malloc(10 * sizeof(int));  // 10 int méretű helyet foglalunk
```

Így biztosan a típushoz passzoló méretet foglalsz, **portábilis és biztonságos**.

---

6. Függvényparamétereknél: nem mindig az, amire számítasz

Ha ezt írod:

```
void printMeret(int t[]) {
    printf("%zu\n", sizeof(t));  // NEM a tömb mérete lesz, hanem pointer
mérete!
}
```

Ott csak a **pointer méretét** kapod meg (`sizeof(int *)`), mert a tömb decay-el pointerré.

Ezért mindig **csak statikus tömböknél** működik jól:

```
int tomb[100];
sizeof(tomb) / sizeof(tomb[0]); // OK
```

---

7. Padding és alignment elemzéséhez

A `sizeof(struct)` megmutatja, hogy a struct mekkora – még ha padding bájtokat is tartalmaz.

---

8. Makrókban is használható

Pl. biztonságos ciklushoz:

```
#define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
```

Összefoglalás

Használat	Mire jó?
`sizeof(típus)`	Típus mérete
`sizeof(változó)`	Változó tényleges típusa alapján
`sizeof(tömb) / sizeof(elem)`	Statikus tömb elemszáma
`sizeof(struct)`	Struktúra teljes mérete
`malloc(n * sizeof(típus))`	Biztonságos memórialefoglalás
`sizeof(pointer paraméter)`	Pointer mérete, **nem tömb**
