

---

# **A C programozási nyelv V. Struktúra Dinamikus memóriakezelés**

Ficsor Lajos

Miskolci Egyetem

Általános Informatikai Tanszék

# A struktúra deklarációja 1.

- A struktúra olyan összetett típus, amelynek elemei különböző típusúak lehetnek.

- Adott szerkezetű változók deklarációja:

```
struct {elemek deklarációja}  
    változólista
```

A változólista elemei adott szerkezetű struktúrák.

# A struktúra deklarációja 2.

- **Struktúra típus létrehozása**

`struct név {elemek deklarációja}`

- Létrehoz egy **név** nevű típust, amely a megadott szerkezetű struktúra. Ezután a **struct név** típusnévként használható deklarációkban.
- **Ez a forma az ajánlott.**

# A struktúra deklarációja 3.

- Az elemek deklarációja pontosvesszővel elválasztott deklarációlista
- Struktúra tagja lehet
  - tetszőleges típusú változó vagy tömb
  - tetszőleges típusú mutató
  - már definiált struktúra (vagy annak mutatója)
  - nem lehet tagja önmaga
  - lehet viszont tagja önmaga mutatója (önhivatkozó struktúra).

# A struktúra deklarációja 4.

Példa struktúra változók deklarálására:

```
struct {  
    int év;  
    char honap[12];  
    int nap;  
} szulinap, ma, tegnap;
```

# A struktúra deklarációja 5.

Példa struktúra típus deklarálására:

```
struct datum {  
    int év;  
    char honap[12];  
    int nap;  
};  
  
struct datum szulinap, ma,  
    tegnap, *pd;
```

# Műveletek a struktúrával

A struktúrákkal az alábbi műveleteket végezhetjük:

- az **&** operátorral képezhetjük a címét
- hivatkozhatunk valamelyik elemére a **"."** (pont) és a **"->"** operátorral
- megengedett két azonos típusú struktúra közötti értékadás
- struktúra lehet függvény paramétere vagy visszatérési értéke

# Hivatkozás a struktúra tagjaira

- Mezőhivatkozás struktúra nevével  
`struktúra_azonosító.mezőnév`

Például:

```
szulinap.ev = 1951;
```

- Hivatkozás struktúra-mutatóval

```
(*pd).mezőnév
```


vagy

```
pd -> mezőnév
```

– A fenti két forma egyenértékű.



# Statikus memória foglалás

- Deklarált objektum
  - a hatáskörébe belépve lefoglalódik a hely
  - a teljes élettartam alatt foglalva marad, akkor is, ha nem használjuk
  - a tömb elemszáma már fordítási időben ismert kell legyen  helypazarlás
- Megoldás: helyfoglalás és a feleslegessé vált memória felszabadítása **futásidőben**

# Dinamikus memória kezelés 1.

**`void* malloc(size_t meret)`**

- Lefoglal **`meret`** byte nagyságú memóriaterületet.
- Visszatérési értéke a lefoglalt terület kezdőcíme, vagy **`NULL`**, ha a helyfoglalás sikertelen volt.
- A visszaadott pointer **`void*`** típusú, ha tehát a memóriaterületet tömb-szerűen akarjuk használni, azt a tárolni kívánt adatok típusára kell konvertálni.
- A **`size_t`** előre definiált típus (**`stdlib.h`**)

# Dinamikus memória kezelés 2.

`void free(void* blokk)`

- Felszabadítja a **blokk** kezdőcímű memóriaterületet.
- A paraméterének előzőleg **malloc**, **calloc** vagy a **realloc** függvény segítségével kellett értéket kapnia.
- További kapcsolódó függvények:  
**calloc**, **realloc**

# Dinamikus memória kezelés (Példa 1.)

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct idopont {
    int ora;
    int perc;
};
```

```
static int nap[]={0,31,28,31,30,31,30,31,
                 31,30,31,30,31};
```

# Dinamikus memória kezelés (Példa 2.)

```
int main()  
{  
    int i;  
    int honap;  
    struct idopont* munkaido;  
  
    scanf("%d",&honap);  
    munkaido = (struct idopont*)  
        malloc(honap*sizeof(struct idopont));
```

# Dinamikus memória kezelés (Példa 3.)

```
for (i = 1; i <= nap[honap]; i++)  
{  
    scanf("%d", &munkaido[i].ora);  
    scanf("%d", &munkaido[i].perc);  
    printf("\n%d %d %d\n",  
           i, munkaido[i].ora,  
           munkaido[i].perc);  
}  
return 0;  
}
```