Programozás C nyelven (11. ELŐADÁS)

Sapientia EMTE

2020-21

Összetett típusok

- Hogyan tároljuk el a memóriában az összetartozó adatokat?
  - TÖMB-változók: azonos típusú összetartozó adatok
    - egész számsorozat elemeit, `int`-tömbben
    - karakterlánc karaktereit, `char`-tömbben
  - … és, ha az összetartozó adatok más-más típusúak?
    - például: egy személy neve (karakterlánc), életkora (egész szám), magassága (valós szám)
    - ezt a célt szolgálja a **`struct`** típus

Összetett típusok nevesítése **typedef**

- A typedef kulcsszó lehetővé teszi, hogy nevet adj egy típusnak
  - új nevet egy már létezőnek
  - nevet egy definiálandónak
- A typedef kulcsszó típus-definiálássá változtat egy változó definiálást
  - konvenció szerint, a felhasználó által definiált típusnevek legyenek csupa nagybetűsek

Szintaxisa:

```
typedef <típus> <azonosító>;
```

Példák:

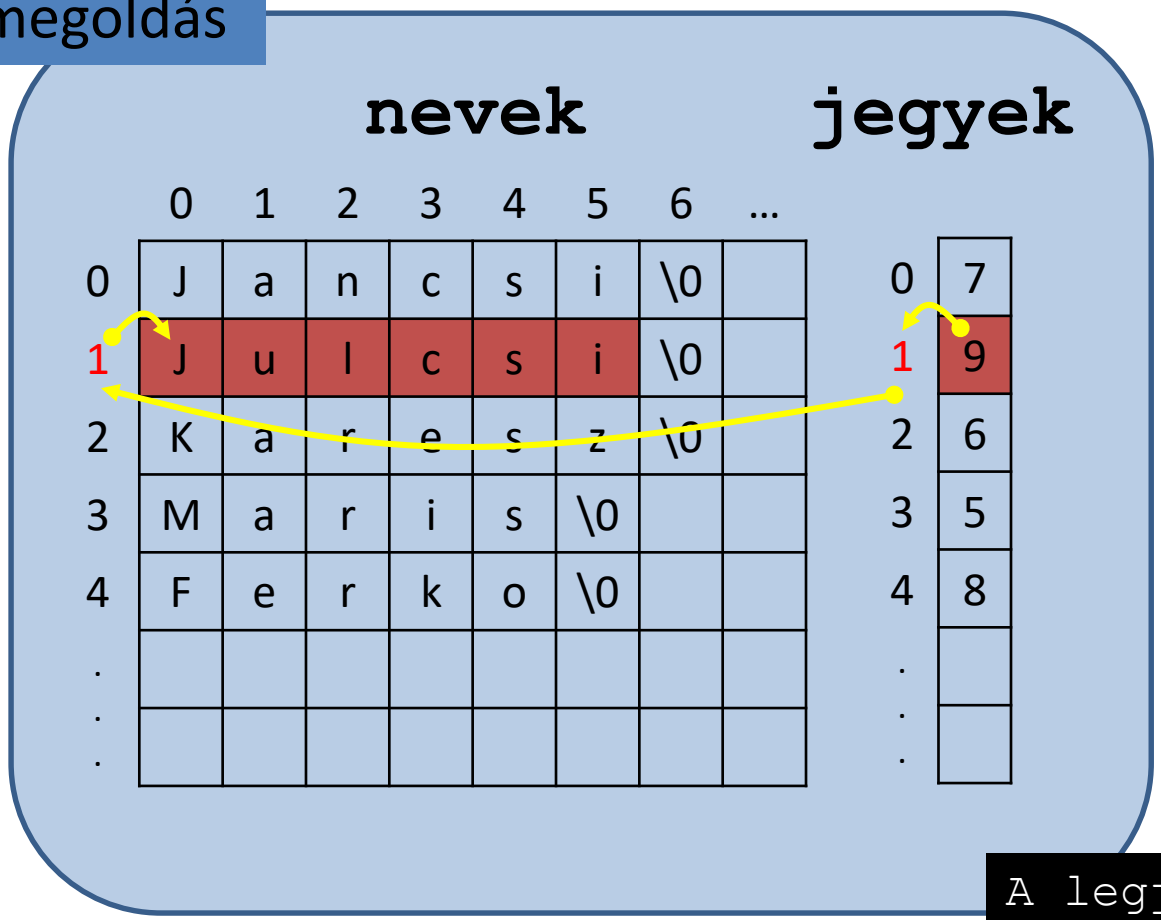
```
int EGESZ     /* a EGESZ egy int típusú változó */
typedef int EGESZ ;   /* az EGESZ egy típus, az int típus
    egy újabb neve */
EGESZ x; int y;   /* mind x, mind y int típusú változó */
typedef float * FLOATPTR;
FLOATPTR fp;   /* fp, float-pointer-változó */
typedef int IVEKTOR[10];
IVEKTOR a;   /* a, 10 elemű int egydimenziós tömbvál-
    tozó */
typedef double DMATRIX[10][20];
DMATRIX b;   /* b, 10 × 20-as double tömbváltozó */
```

Olvassunk be állományból n (n≤100) diák egyszavas nevét (max 20 karakter) és prog-I vizsgajegyét, majd írassuk ki a képernyőre egy „legjobb” diák nevét.

1. megoldás

progI.txt

```
5
Jancsi 7
Julcsi 9
Karesz 6
Maris 5
Ferko 8
```




```
A legjobb diak:
Julcsi_
```

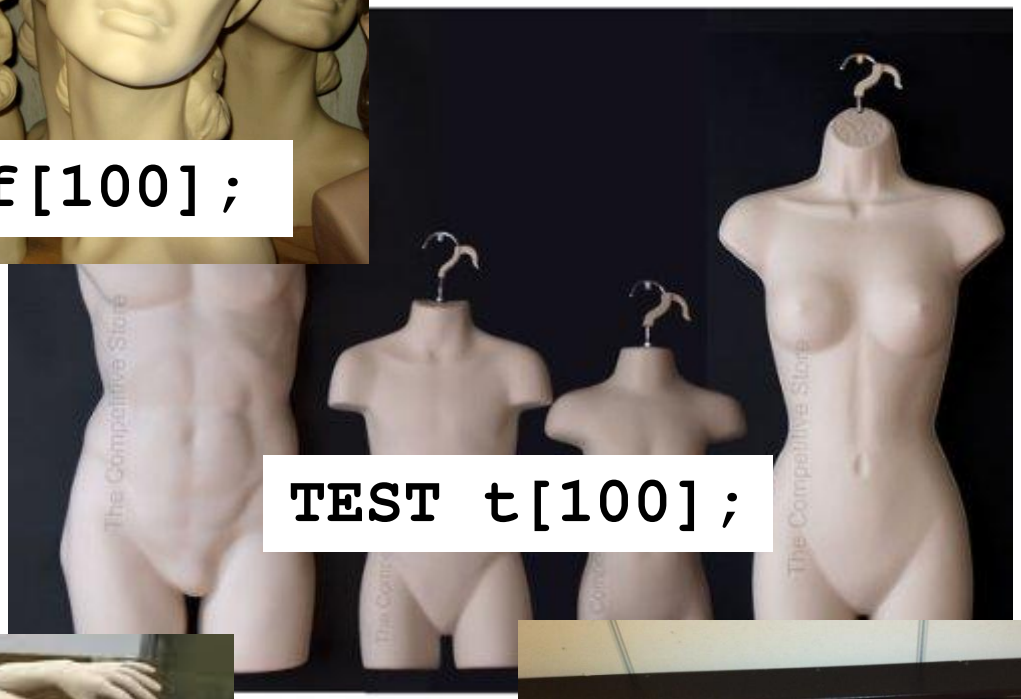
Olvassunk be állományból n (n≤100) diák egyszavas nevét (max 20 karakter) és prog-I vizsgajegyét, majd írassuk ki a képernyőre egy „legjobb” diák nevét.

1. megoldás

```
int n,jegyek[100],i,maxindex;
char nevek[100][21];
freopen("progI.txt", "r", stdin);
scanf("%i", &n);
for( i = 0 ; i < n ; ++i ){
   scanf("%s%i", nevek[i], &jegyek[i]);
}
maxindex = legjobbjegyindex(n,jegyek);
printf("A legjobb diak:\n%s", nevek[maxindex]);
```

Hiányosság: az adatábrázolásban nem tükröződik, hogy egy adott személy neve és jegye összetartozó adatok


FEJ f[100];


TEST t[100];


KEZ k[100];


LAB l[100];

`struct` típus

`m[1].f`

`MANOKEN m[100];`

```
typedef struct{
   FEJ f;
   TEST t;
   KEZ k;
   LAB l;
} MANOKEN;
...
MANOKEN m[100];
```

`m[4].l`

- Mivel `m` tömb, ezért a celláira indexeléssel hivatkozunk: `m[1]`
- Mivel `m[1]` struct, ezért a mezőire pont-tal hivatkozunk: `m[1].f`

```
typedef struct
{
  <típus_1> <mező_1>;
  <típus_2> <mező_2>;
  ...
  <típus_n> <mező_n>;
} <típus_azonosító>;
```

Olvassunk be állományból n (n≤100) diák egyszavas nevét (max 20 karakter) és prog-I vizsgajegyét, majd írassuk ki a képernyőre egy „legjobb” diák nevét.

2. megoldás

```
typedef struct{
    char nev[21];
    int jegy;
}DIAK;
...
DIAK d[100];
```

d

0	J	a	n	c	s	i	\0	…	7
1	J	u	l	c	s	i	\0	…	9
2	K	a	r	e	s	z	\0	…	6
3	M	a	r	i	s	\0	…		5
4	F	e	r	k	o	\0	…		8
…									

progI.txt

```
5
Jancsi 7
Julcsi 9
Karesz 6
Maris 5
Ferko 8
```

d[2].jegy

d[4].nev

```
A legjobb diak:
Julcsi_
```

Olvassunk be állományból n (n≤100) diák egyszavas nevét (max 20 karakter) és prog-I vizsgajegyét, majd írassuk ki a képernyőre egy „legjobb" diák nevét.

2. megoldás

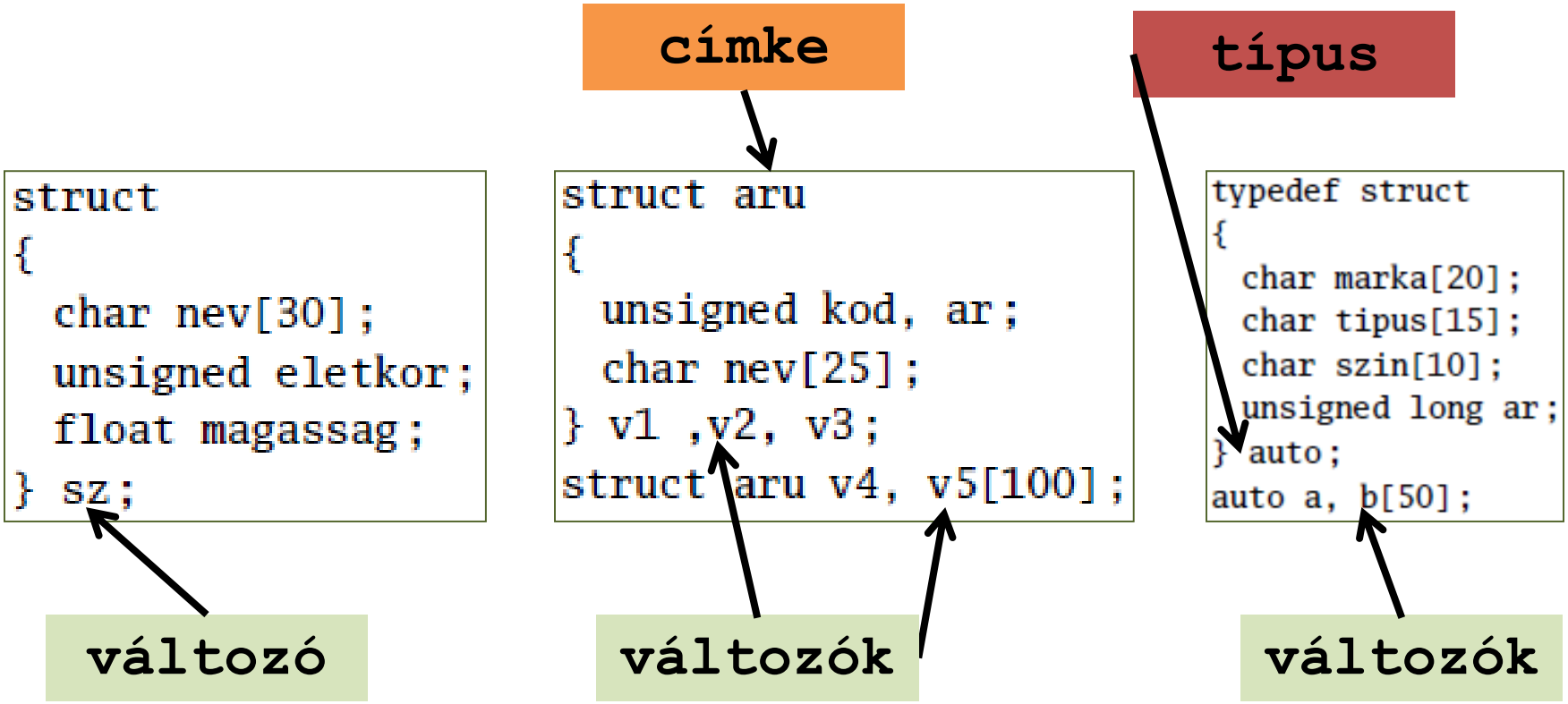
```
typedef struct{
   char nev[21];
   int jegy;
}DIAK;
void irdkilegjobbdiaknev(int,DIAK*);
int main(){
   int n,i; DIAK d[100];
   freopen("progI.txt", "r", stdin);
   scanf("%i", &n);
   for( i = 0 ; i < n ; ++i ){
      scanf("%s%i", d[i].nev, &d[i].jegy);
   }
   irdkilegjobbdiaknev(n,d);
   return 0;
}
```

Olvassunk be állományból n (n≤100) diák egyszavas nevét (max 20 karakter) és prog-I vizsgajegyét, majd írassuk ki az adatokat képernyőre ABC, illetve jegyek szerinti csökkenő sorrendben.

```
typedef struct{
    char nev[21];
    int jegy;
}DIAK;
void kiiras(int, DIAK*);
int nev_cmp(const void*, const void*);
int jegy_cmp(const void*, const void*);
int main(){
    int n,i; DIAK d[100];
    freopen("progI.txt", "r", stdin);
    scanf("%i", &n);
    for( i = 0 ; i < n ; ++i ){
        scanf("%s%i", d[i].nev, &d[i].jegy);
    }
    qsort(d,n,sizeof(DIAK),nev_cmp); kiiras(n,d);
    qsort(d,n,sizeof(DIAK),jegy_cmp); kiiras(n,d);
    return 0;
}
```

```
int nev_cmp(const void *p1, const void *p2){
    DIAK *q1 = (DIAK*)p1;
    DIAK *q2 = (DIAK*)p2;
    return strcmp((*q1).nev, (*q2).nev);
}
```

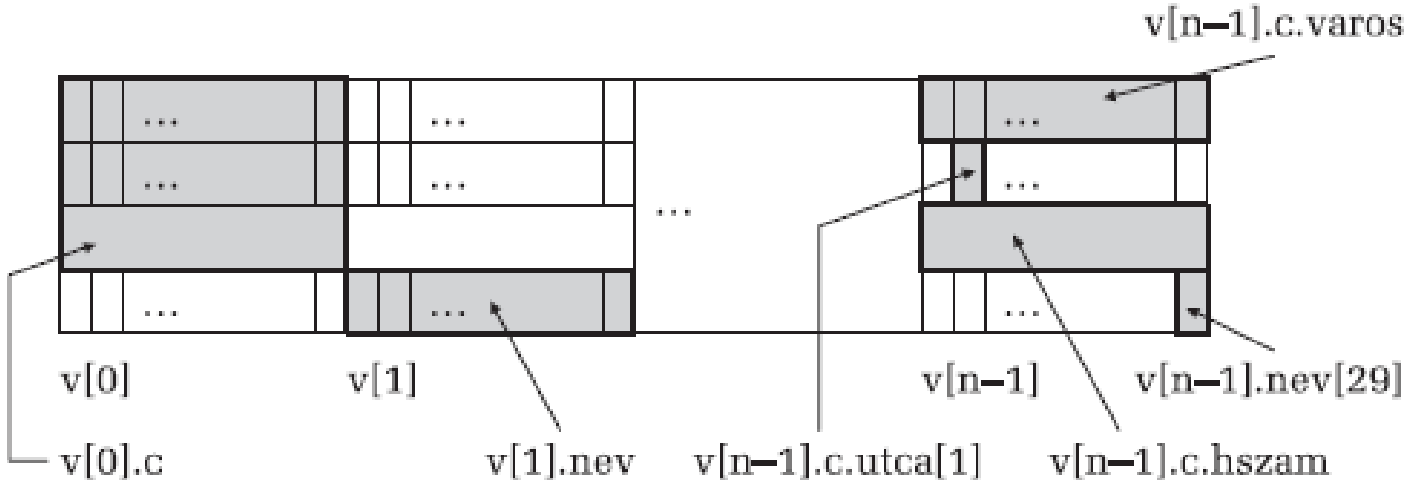
`struct` típus/címke/változó



Egymásba-ágyazott **`struct`**-ok

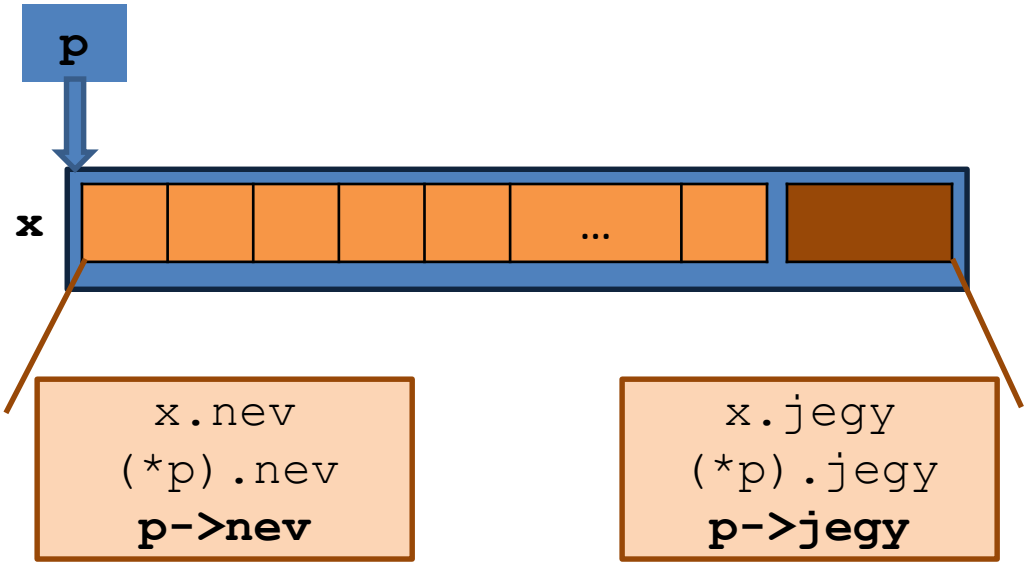
```
struct cim
{
    char varos[30], utca[25];
    int hszam;
};
typedef struct
{
    struct cim c;
    char nev[30];
} szemely;
szemely *v;
```

```
v = (szemely*)malloc(n * sizeof(szemely));
```



`struct` – pointerek

```
typedef struct{
    char nev[21];
    int jegy;
}DIAK;
...
DIAK x, *p = &x;
```



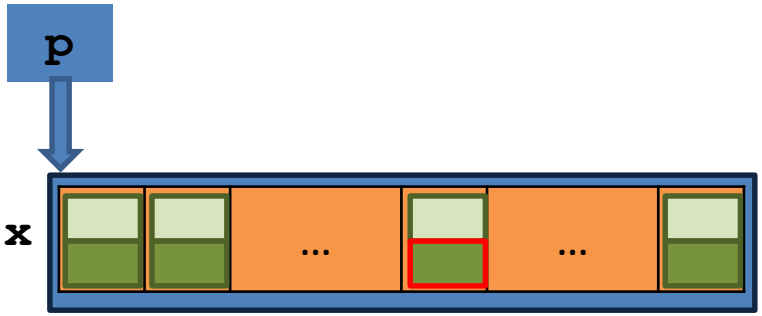
Tehát egy struktúra mezőire, a rá mutató pointer segítségével, az alábbi módokon hivatkozhatunk:

`<pointer> -> <mező>` vagy `(*<pointer>).<mező>`

```
int jegy_cmp(const void *p1, const void *p2){
    DIAK *q1 = (DIAK*)p1;
    DIAK *q2 = (DIAK*)p2;
    return q2->jegy - q1->jegy;
}
```

Összehasonlító függvény jegy-szerint csökkenő sorrendbe rendezéshez

Agytorna




```
typedef struct {int a,b;} tipus1;
typedef struct {tipus1 c[50];} tipus2;
tipus2 x, *p = &x;
```

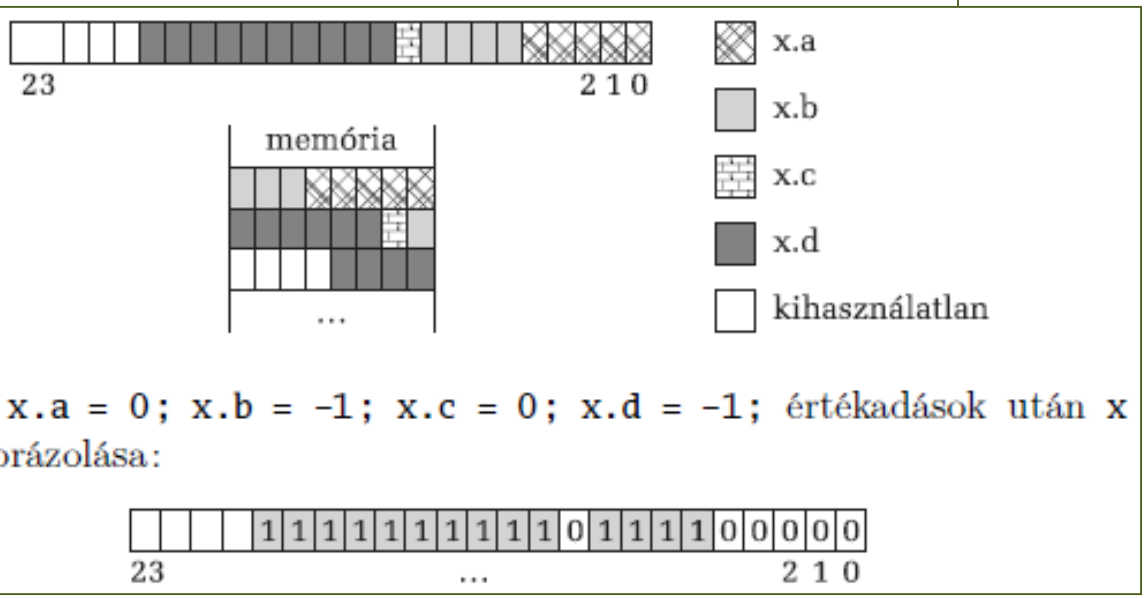
Hogyan hivatkozhatunk az x struktúra c mezőjének 25. eleme b mezőjére?

Megoldások:

1. x.c[25].b
2. (*p).c[25].b
3. p->c[25].b
4. (*((*p).c+25)).b
5. (p->c+25)->b

BIT– mezők

```
struct címke
{
  unsigned a:5;    /* 5 biten tárolt előjel nélküli egész */
  signed b:4;    /* 4 biten tárolt előjeles egész */
  unsigned c:1;    /* 1 biten tárolt előjel nélküli egész */
  int d:10;    /* 10 biten tárolt előjeles egész */
} x;
```



Az `x.a = 0; x.b = -1; x.c = 0; x.d = -1;` értékadások után `x` belső ábrázolása:

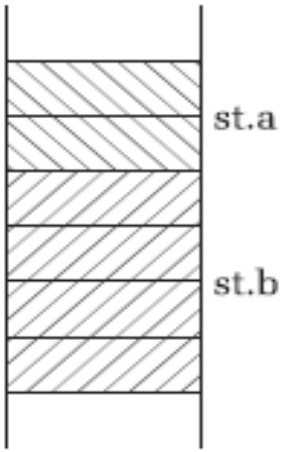
Megjegyzések:

- Egy `struct` változónak vegyesen lehetnek bitmezői és nem bitmezői.
- Nem lehet hivatkozni egy bitmező címére.
- A bitmezők nem rendezhetők tömbökbe.
- A géptől függ, hogy a bitmezők jobbról balra vagy balról jobbra sorrendűek.

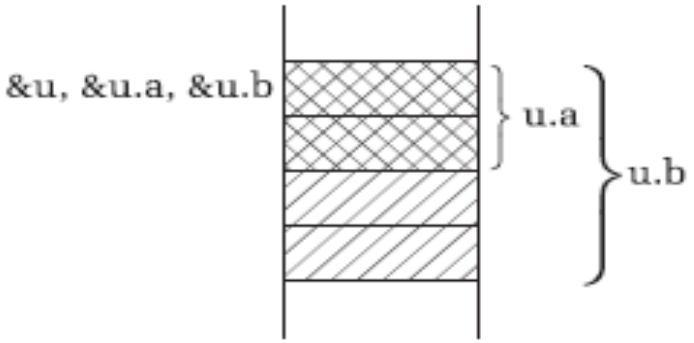
union típus

A **struct** típus esetén a mezők egymás után kerülnek eltárolásra. A **union** típust pontosan úgy definiáljuk, mint a **struct**-ot, csakhogy mezői egymásra tevődnek. Úgy is mondhatnánk, hogy a mezők együtt használják ugyanazt a memóriaterületet. Az alábbi ábrák jól érzékeltetik a különbséget:

```
struct
{
  short a;
  float b;
} st;
```




```
union
{
  short a;
  float b;
} u;
```



Közös lónak túros a háta

Egy **union** változó tárhelyigényét nyilván a legnagyobb méretű mezője határozza meg. Például az **u** változó mérete 4 byte lesz (`sizeof(float)`).

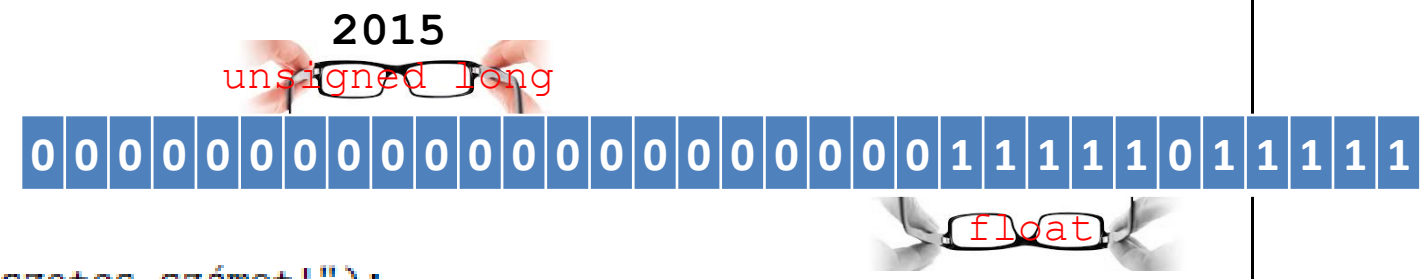
Közös lónak túros a háta

- Sokan értetlenül állnak e mondás előtt, hogyan is lehet túrós egy ló háta? Holott mindössze egy félreértésről van szó. Nem túrós, hanem túros. A túr egy szláv eredetű szó, jelentése: fekély, seb. Vagyis a közös lóra – mivel senki sem a kizárólagos gazdája – nem vigyáznak: igába fogják, és addig hajtják, amíg kisebesedik a háta.

`union` típus

8.4. feladat. Adott egy természetes szám. Melyik az a valós szám, amelynek `float` típusúkénti belső ábrázolása azonos az illető természetes szám `unsigned long`-kénti belső ábrázolásával?

```
union
{
    unsigned long x;
    float y;
} a;
puts("Írd be a természetes számot!");
scanf("%lu", &a.x);
printf("A megfelelő valós szám: %f", a.y);
```



enum típus

Mi határozza meg, milyen értékeket vehet fel egy változó? A típusa. Az eddig megismert egyszerű típusok esetében implicite adva volt az értéktartomány. Például egy **char** változó **-128** és **127** közötti egészeket kaphat értékként. Az **enum** típus segítségével a felhasználó sorolhatja fel, milyen értékeket vehet fel egy ilyen típusú változó. A felsorolt azonosítók, amint látni fogjuk, valójában egész konstans nevek. Tehát a **#define** direktíva és a **const** módosító jelző mellett az **enum** típus is lehetővé teszi szimbolikus egész konstansok definiálását. Mivel az **enum** típus értéktartományának minden értéke nevet kap, ezért alkalmazása áttekinthetőbb kódot eredményez.

```
enum napok
{
  hetfo, kedd, szerda, csutortok, pentek, szombat=10,
    vasarnap
} nap;
```

hetfo értéke 0, kedd-é 1, pentek-é 5, vasarnap-é

?

```
nap = pentek;
```

```
enum
{
  jan=1, feb, mar, apr, maj, jun, jul, aug, szept, okt,
    nov, dec
};
```

enum típus

8.7. feladat. Készítsünk egy kimutatást arról, hogy a hét melyik napján a legnagyobb egy adott üzlet bevétele. A billentyűzetről **n** nap sorszámát (hétfő: 1, . . . , vasárnap: 7) és az aznapi bevételt olvassuk be.

```
enum
{
    hetfo=1, kedd, szerda, csutortok, pentek,
        szombat, vasarnap
};
char * napok[8] = {"", "hétfő", "kedd", "szerda",
    "csütörtök", "péntek", "szombat", "vasárnap"};
unsigned long bevetel[8] = {0}, penz, maxpenz;
int nap, n, maxnap, i;
puts("Add meg a napok számát:");
scanf("%d", &n);
puts("Írd be a napokat és a pénzösszegeket:");
for(i = 1; i <= n; i++)
    {
        scanf("%d%lu", &nap, &penz);
        bevetel[nap]+=penz;
    }
maxpenz = bevetel[hetfo]; maxnap = hetfo;
for(i = kedd; i <= vasarnap; i++)
    if(bevetel[i] > maxpenz)
{maxpenz = bevetel[i]; maxnap = i;}
printf("A legnagyobb forgalmú nap: %s", napok[maxnap]);
```

ISMÉTLÉS

További részletek végett lásd a jegyzet 8. fejezetét

- **`typedef`** módosító jelző
- **`struct`** típus
  - hivatkozás adott mezőre **pont** operátorral
  - `struct` – pointerek
    - hivatkozás adott mezőre **nyíl** operátorral
  - BIT – mezők
- **`union`** típus
- **`enum`** típus