# Tartalomjegyzék


- 1 Tömbök és függvények
  - 1.1 Tömb mint függvény paraméter
  - 1.2 Tömb visszaadása függvényb?l
- 2 Típusok definiálása
  - 2.1 const
    - 2.1.1 konstans változók
    - 2.1.2 konstans mutatók
  - 2.2 typedef
  - 2.3 enum, switch/case
    - 2.3.1 enum típusok megadása
    - 2.3.2 switch/case
  - 2.4 struktúrák


# Tömbök és függvények

Még az el?z? el?adás mutató/tömb témájával kapcsolatban ez kimaradt akkor.

## Tömb mint függvény paraméter

Mint mondtam, a tömb változóknak nem lehet változtatni az "http://wiki.math.bme.huértékét"http://wiki.math.bme.hu, csak a "http://wiki.math.bme.hutartalmát"http://wiki.math.bme.hu, tehát eleve nem futtathatnék ilyesmit, ezt nem fogadja el a fordító:

```
int v1[10];
int v2[10];
v1 = v2;
```

Tehát ha ilyesmit nem csinálhatunk, akkor a függvény paraméterként kapás, amit múlt órán láttunk, az hogy hogy m?ködik? Úgy, hogy ha függvény paraméternek tömböt írunk, akkor azt a C teljesen úgy kezeli, mintha mutatót írtunk volna. Tehát ez a 3 mind ekvivalens:

```
void f1(int param[]) { ... }
void f2(int param[10]) { ... }
void f3(int *param) { ... }
```

Tehát összefoglalóan, változó definiálásnál a szögletes zárójelnek kétféle jelentése lehet:

- Ha a változó egy függvény paraméter, akkor a szögletes zárójel az nem jelent mást mint a csillag, azt jelenti, hogy az adott változó mutató.

- Ha a változó nem függvény paraméter, hanem egy tetsz?leges blokkon belül definiált változó, akkor az egy új változó, ami vagy akkora amit a szögletes zárójelben megadtunk, vagy akkora amennyi értéket beletettünk kezdetben. Addig létezik ez a tömb, amig a blokk végéig érünk.

Ennek egyik következménye, hogy a tömbök "http://wiki.math.bme.humutable"http://wiki.math.bme.hu-ként viselkednek, amikor átadjuk ?ket függvény paraméterként, de ez csak azért van, mert ilyenkor gyakorlatilag mutatót adunk át.

## Tömb visszaadása függvényb?l

Eddig nem beszéltünk arról, hogy hogy nézne ki az, ha egy tömböt akarnánk visszaadni egy függvényb?l. A szintaxis eleve nem m?ködik, mert hogy nézne az ki? "http://wiki.math.bme.hu`int f[]() { ...`"http://wiki.math.bme.hu? vagy "http://wiki.math.bme.hu`int f()[] { ...`"http://wiki.math.bme.hu? Ez egyik sem m?ködik, és ez nem véletlen. Hiszen az így visszaadott tömböt nem is lehetne letárolni, mint az el?bb láttuk.

Ehelyett egyel?re azt a módszert alkalmazhatjuk, ha azt akarjuk, hogy egy függvény egy tömböt adjon vissza, hogy a függvénynek odaadunk egy tömböt, és abba beírja a megfelel? értékeket a függvény. Ilyenkor érdemes a tömb hosszát is odaadni a függvénynek, hogy ne menjen túl. És ilyenkor az a szokás, hogy az a visszatérési érték, hogy hány elemét sikerült a tömbnek ténylegesen feltölteni. Pl.

```
// Feltolti a tomb-ot az n-nel kisebb paros
// szamokkal
int parosok(int n, int tomb[], int tombhossz) {
  int i;
  int hely = 0;
  for(i = 0; i < n; i += 2) {
    tomb[hely] = i;
    ++hely;
    if(hely >= tombhossz){
      return hely;
    }
  }
  return hely;
}

#define TOMB_MERET 20

int main()
{
  int szamok[TOMB_MERET];
  int siker;
  siker = parosok(30, szamok, TOMB_MERET);

  // ...

  return 0;
}
```

Itt a "http://wiki.math.bme.husiker"http://wiki.math.bme.hu változó értéke 15 lesz, mert 15 elemet írt a függvény a tömbbe.

# Típusok definiálása

Az eddigi típusok amiket tanultunk, azok az egész és valós számok, és az ezekb?l álló tömbök, meg az ezekre mutató mutatók. A mai el?adás hátralev? részében végignézünk minden más típust ami még fontos a C-ben

és eddig nem említettük. Két dolgot még kihagyunk (a **union**t és a függvény mutatót), mert azokat akkor lenne hasznos elmondani ha b?ven több mint 6 el?adás lenne C-b?l, és azok már sokkal ritkábban kellenek.

Ezekkel a típusokkal amik ezen az el?adáson lesznek, a C kódok túlnyomó többségét már meg tudjátok érteni.

# const

## konstans változók

A **const** kulcsszó a C nyelvben egy típus módosító, amit a típus neve elé lehet tenni. Ha egy változó *const* típusú, az azt jelenti hogy nem változtathatjuk meg az értékét sohasem. Mivel nem változtathatjuk meg az értékét, ezért ha *const*-nak deklarálunk egy változót, akkor általában egyb?l abban a sorban értéket is adunk neki, mert kés?bb már nem lehet:

```
const int n = 10;
// Ezt nem tudnam csinalni:
// n = 20;
```

Ha egy függvényparaméterre mondjuk hogy *const*, akkor az persze a függvény hívásakor kap értéket, és a függvényen belül nem változtatható meg:

```
void kiir(const int k) {
  printf("%d\n", k);
  // Itt nem valtoztathatnam meg k-t, nem lehetne pl.
  // --k;
}

void main() {
  int n = 10;
  // Itt kap a "k" erteket:
  kiir(n);
}
```

## konstans mutatók

Mutató esetén kétféleképpen érthetem hogy egy változó legyen konstans:

- vagy azt akarom, hogy a mutatón keresztül elérhet? változó értékét ne módosíthassam,
- vagy azt, hogy ne módosíthassam azt, hogy melyik változóra mutat a mutató.

Attól függ?en hogy melyiket szeretném csinálni, máshova kell tennem a változó deklarálásában a *const*-ot:

```
int a = 5;
// Normalis mutato:
int *m1 = &a;
// Mutato, amin keresztul nem lehet valtoztatni
// a masik valtozo erteket:
const int *m2 = &a;
// Mutato, aminel nem lehet valtoztatni
// hogy hova mutat:
int * const m3 = &a;
// Mutato aminel semmit sem lehet valtoztatni:
const int * const m4 = &a;
```

Úgy lehet megjegyezni, hogy melyiknek mi a hatása, hogy amikor az egész számot nem lehet változtatni (amire mutat), akkor az *int* mellett van a *const*, amikor a mutatót magát, tehát azt hogy hova mutat, akkor a csillag mellett van a *const*.

Karakterláncoknál sokszor nem szabad változtatni a karakterlánc tartalmát. Pl. ha egy karakterláncot nem egy saját tömbben tárolunk, csak leírjuk idéz?jelek közé (és pl. így adjuk át függvényparaméternek), akkor az olyan karakterlánc, aminek a tartalmát nem szabad változtatni. Ezért az ilyen karakterláncokat általában "http://wiki.math.bme.hu`const char *`"http://wiki.math.bme.hu típusként szokták a függvények paraméterként kapni.

```
char elsobetu(const char* szoveg) {
  return szoveg[0];
}

char c = elsobetu("Mondat!");
```

A beépített könyvtárak függvényei is, amik nem módosítják a karakterláncokat, így mindig "http://wiki.math.bme.hu`const char *`"http://wiki.math.bme.hu típusú paramétereket várnak. Az ilyeneknek odaadható olyan karakterlánc is amit tömbben tárolunk, és olyan is amit csak odaírunk. ("http://wiki.math.bme.huchar*"http://wiki.math.bme.hu vagy "http://wiki.math.bme.huchar[]"http://wiki.math.bme.hu típusú változót elfogad a C olyan helyre ahol "http://wiki.math.bme.huconst char*"http://wiki.math.bme.hu-ot vár.)

# typedef

Egy kulcsszó a C nyelvben, amit eddig nem tanultunk, a **typedef**. Ennek az a célja, hogy ha egyfajta típust sokszor használunk, rövidebb nevet is adhatunk neki. Úgy néz ki, formátumilag, hogy leírjuk hogy "http://wiki.math.bme.hu`typedef`"http://wiki.math.bme.hu, majd a sor hátralev? része pontosan ugyanolyan mintha egy változót deklarálnánk, de amit névnek megadunk, az egy új típusnak a neve. Pl.:

```
typedef unsigned long long int ull_int;

int main()
{
  ull_int i;
  for(pont = 0; i < 50000000000ll; ++i) {
    printf("%d ", i % 100);
  }
}
```

Ezt nem csak az alap típusokhoz, hanem a különböz? módosított típusokhoz is lehet használni:

```
typedef int vektor[3];
typedef int *p_int;

int main()
{
  vektor pont;
  pont[0] = 4;
  pont[1] = 5;
  pont[2] = 6;
  p_int mutato = &pont[0];
}
```

Ekkor a 3 elem? tömböt elneveztem egy új néven, *vektor*-nak, és innent?l kezdve a *vektor* ugyanazt jelenti, mint ha azt írnám hogy 3 elem? tömb legyen. De attól még hogy így elneveztem, nem tud semmivel többet mint a 3 elem? tömb tudna: nem lehet egyben megváltoztatni az értékét, nem m?ködnek rá az összehasonlító

operátorok, ha függvényparaméterként átadom, továbbra is csak mutatóként adódik át, nincsen egyszer?bb mód a kiiratására, semmi ilyesmi.

```
vektor pont1, pont2;
// Ezek mind nem mukodnek
pont1 = pont2;
if(pont1 < pont2) {
  printf("%v", pont1);
}
```

Annak ellenére hogy ez ilyen korlátozottan hat a dolgokra a *typedef*, azért hasznos lehet

- mert rövidebb név lehet, amit könnyebb leírni
- ha több különböz? változónak mindenképp ugyanaz a típusa, de a típus lehet hogy megváltozik majd a program fejl?dése folyamán, így elég lehet csak a typedef-et átírni.

# enum, switch/case

## enum típusok megadása

Az **enum** nem csak egy típus a C-ben, hanem egy típus család. Az *enum*-al olyan típusokat lehet csinálni, amiknek azt szeretnénk, hogy az értéke bizonyos fix lehet?ségek közül legyen kiválasztva. Nézzünk egy példát:

```
// Ebben a sorban a tipust definialjuk, hasonloan mint a typedef:
enum izek {CSOKIS, VANILIAS, EPRES, MALNAS};
// Ezek ilyen tipusu valtozok:
enum izek fagyi = CSOKIS;
enum izek szekrenyben[] = {CSOKIS, EPRES, EPRES, EPRES, VANILIAS};

if(fagyi == EPRES) {
  printf("Hmm de finom!\n");
}
```

Tehát ebben az esetben a változók típusa "http://wiki.math.bme.hu`enum izek`"http://wiki.math.bme.hu. Az els? sor mondja meg hogy az "http://wiki.math.bme.hu`enum izek`"http://wiki.math.bme.hu nev? *enum* típus milyen értékeket vehet fel, majd a második sorban definiálok egy "http://wiki.math.bme.hu`enum izek`"http://wiki.math.bme.hu típusú változót, a harmadik sorban pedig egy ilyen típusú változókat tartalmaz? tömböt.

Az *enum* változókat a C egész számokként tárolja, és egész számokként kezeli, ezért ezt megengedi a fordító:

```
fagyi = 2;
```

De ha már egy változó *enum* típusú, akkor én azt ajánlom, mindig csak a megnevezett értékeket tegyük bele, és azokkal hasonlítsunk össze.

## switch/case

A **switch** egy feltételt megadó parancs, ami kicsit az *if*-re hasonlít, de bizonyos konkrét esetben használható. Arra használható, ha egy változónak több különféle konkrét értéke lehet, amik esetében mást kell csinálni. Tehát ha ilyen *if* sorozatunk van:

```
if(a == 0) {
  // valami
```

```
} else if(a == 1) {
  // valami mas
} else if(a == 2) {
  // harmadik fele dolog
} else {
  // kulonben ez legyen
}
```

Akkor ezt lehet lecserélni a direkt erre kitalált struktúrával:

```
switch(a) {
case 0:
  // valami
  break;
case 1:
  // valami mas
  break;
case 2:
  // harmadik fele dolog
  break;
default:
  // kulonben ez legyen
  break;
}
```

Ezt *enum*okra f?leg szokták használni:

```
switch(fagyi) {
case CSOKIS:
  printf("Az edessegek kiralya!\n");
  break;
case VANILIAS:
  printf("Jo, de kicsit uncsi.\n");
  break;
case MALNAS:
  printf("Sok kis kemeny mag, bleh.\n");
  break;
}
```

Azért lehet ezt *enum*ra különösen jól használni, mert

- Ennél úgyis csak bizonyos véges sok érték lehet, és általában egyenl?séget ellen?rzünk vele, nem olyant hogy nagyobb/kisebb valaminél, amit *switch*-el nem lehet.
- Mivel tudhatjuk a konkrét értékeket amiket felvehet a változó, tud szólni a fordító ha valamelyiket elfelejtjük kezelni. Pl. a fenti kódnál szólna hogy az *EPRES* esetet kifelejtettük (csak *warning*gal, tehát azért lefordulna és futna a program).

Még szeretném külön felhívni a figyelmet a *break*-ekre mindegyik *case* végén, azt könny? elfelejteni amikor *switch* kódot írunk, de ha kihagyjuk akkor a következ? *case* is egyben végrehajtódik.

# struktúrák

A struktúrák C-ben a megfelel?i a python objektumoknak, de (ahogy megszokhattuk), jelent?sen egyszer?bbek. Lényegében csak arra jó, hogy egy adag változót összecsoportosíthatunk egy blokkba, egy változóba. Az *enum*-hoz hasonlóan kell definiálni, tehát el?ször megmondjuk hogy az adott fajta struktúra mib?l is áll, majd után csinálhatunk ilyen változókat. A kulcssó a **struct**, itt egy egyszer? példa:

```
// Az adott struct altipus definialasa:
```

```
struct ZHeredmeny {
  int pontszam, jegy;
  char neptun[7];
};

// Hasznalat:
struct ZHeredmeny eredmeny1, eredmeny2;
eredmeny1.pontszam = 23;
strcpy(eredmeny1.neptun, "KOD123");
eredmeny2 = eredmeny1;
eredmeny2.neptun[0] = 'D';
```

Tehát mit tud pontosan egy *struct*?

- A tagváltozók a "http://wiki.math.bme.hu."http://wiki.math.bme.hu (pont) operátorral érhet?ek el, mint python-ban. Ezeket C-nél inkább a struktúra **mez?**inek szokták hívni, nem tagváltozóknak.
- Kezdeti érték adásnál meg lehet adni a mez?k tartalmát kapcsos-zárójelben (mint egy tömböt). Ugyanazt a sorrendet kell használni mint az adott struktúra típus megadásánál:

```
struct ZHeredmeny eredmeny3 = {20, 4, "LZRKTD"};
```

- Lehet lemásolni egyik struktúra értékét egy másikba. Ilyenkor minden mez? lemásolódik, beleértve a struktúrán belül tárolt tömböket.

Miket nem tud, amit esetleg elvárhatnánk?

- Az összehasonlító operátorok nem m?ködnek rá, még az egyenl?séget ellen?rz? "http://wiki.math.bme.hu=="http://wiki.math.bme.hu sem. Ha ilyesmi kell, akkor külön függvényt kell írni rá, pl:

```
int ZHeredmeny_egyenlo(struct ZHeredmeny a, struct ZHeredmeny b) {
  if(a.pontszam == b.pontszam && a.jegy == b.jegy &&
     strcmp(a.neptun, b.neptun) == 0) {
    return 1;
  } else {
    return 0;
  }
}
```

- Nincsen külön szintaxis a struktúrához tartozó függvényekre, nincsenek "http://wiki.math.bme.humetódusok"http://wiki.math.bme.hu. A struktúrához köthet? függvényeknek a nevéb?l derülhet ki hogy oda tartoznak, mint az el?z? példa.
- Ehhez köt?d?en, nincsen semmi egyszer? módszer a struktúra kiírásának kezelésére, nem lehet egy új fajta százalékos jelöl?t hozzárakni a *printf()*-hez, vagy ilyesmi. Az egyetlen megoldás hogy csinálunk egy hozzátartozó függvényt, ami kiírja.

Annak ellenére hogy ilyen limitált képességei vannak, azért az a képesség hogy az adatokat így összecsoportosítsa, már elég hasznossá teszi a struktúrákat.