**Pointerek használata**

**\*\*\***

**--- gyakorló programozási feladatok   
\*\*\***

**Néhány megfontolás, megjegyzés**

A lenti programokban gyakran *tömböt* adunk át egy függvénynek.   
Az alapvető különbség a „sima” tömbök és a nullával lezárt sztringek között az, hogy az utóbbiaknál a tömb tartalma (karakterek és lezáró nulla) alapján tudjuk azt, hogy hol van vége annak. Ebben a speciális esetben nem kell átadni a tömb méretét a függvényeknek, máskor viszont mindig igen! Érdemes ilyen szemmel (is) összehasonlítani a programokat.

Másrészről: a paraméterként kapott tömböt a függvény tudja módosítani, hiszen cím szerint kapja meg azt. Az eredeti memóriát látja, nem a tömb egy másolatát! Ezt is kihasznál(hat)juk a függvényeinkben.

1. **mintafeladat: Cím szerinti paraméterátadás**

**Írjunk egy függvényt, amelyik kiszámítja és visszaadja két szám összegét és szorzatát  
  
Megjegyzések a megoldáshoz:**

1. **Mivel két adatot is vissza kell adnia, cím szerinti átadást kell használnunk. A visszatérési érték helyett átvesszük cím szerint a két változót, amelybe írni kell.**

void szamol(int a, int b, int \*osszeg, int \*szorzat) {

\*osszeg = a + b;

\*szorzat = a \* b;

}

int osszeg, szorzat;

szamol(9, 11, &osszeg, &szorzat);

1. **Természetesen ilyenkor is érték szerinti átadás történik, csak ez az érték most a memóriacím. A függvény belsejében az**osszeg**típusa**int**-re mutató pointer, a**szorzat**változóé ugyanaz.**
2. **Másik lehetőség, hogy a visszaadandó (számított) értékeket egy megfelelő összetett adatszerkezet, pl. tömb vagy struktúra formájában a „szamol()” függvény visszatérési értékeként határozzuk meg – pl. így:**

typedef struct Operatorok {

int elso\_operandus;

int masodik\_operandus;

int osszeg;

int szorzat;

} Muveletek

Muveletek\* szamol(int a, int b) {

Muveletek\* szamitasok = (Muveletek\*) malloc(sizeof(Muveletek));

szamitasok->elso\_operandus = a;

szamitasok->masodik\_operandus = b;

szamitasok->osszeg = a + b;

szamitasok->szorzat = a \* b;

return szamitasok;

}

Muveletek \*szamitasok = szamol(9, 11);

printf("Osszeg: %d\nSzorzat: %d", szamitasok->osszeg, szamitasok->szorzat);

1. **mintafeladat: Tömb átadása függvénynek**

**Írjunk függvényt, amely paraméterként átvesz egy egész számokból álló tömböt, és szétválogatja azokat egy második és egy harmadik tömbbe, aszerint hogy párosak vagy páratlanok!**

**Írjunk függvényt, amely kiírja a paraméterként kapott, egész számokból álló tömb elemeit!**

**Írjunk ezekhez főprogramot, amelyikkel szétválogatunk egy számsort, és kiír(at)juk külön a párosakat, külön a páratlanokat!**

#include <stdio.h>

void paritas\_valogat(int \*be, int be\_db, int \*ki\_paros, int \*ki\_ptln, int \*ki\_paros\_db) {

int i\_paros = 0, i\_ptln = 0;

for (int i = 0; i != be\_db; ++i) {

if (be[i] % 2 == 0) {

ki\_paros[i\_paros++] = be[i];

} else {

ki\_ptln[i\_ptln++] = be[i];

}

}

\*ki\_paros\_db = i\_paros;

}

void egesz\_tomb\_kiir(int \*be, int be\_db) {

for (int i = 0; i != be\_db; ++i)

printf("%d, ", be[i]);

printf("\n");

}

int main(void) {

int szamok[10] = { 1, 7, 2, 6, 9, 3, 4, 5, 9, 1 };

int paros[10], ptln[10];

int paros\_db, ptln\_db;

paritas\_valogat(szamok, 10, paros, ptln, &paros\_db);

ptln\_db = 10 - paros\_db;

printf("Parosak:\n");

egesz\_tomb\_kiir(paros, paros\_db);

printf("Paratlanok:\n");

egesz\_tomb\_kiir(ptln, ptln\_db);

return 0;

}

**Megjegyzések a megoldáshoz:**

1. **Három tömbbel dolgozunk, és mivel akárhány páros és páratlan szám lehet össze-vissza, három indexre van szükségünk:**

**--- (1) folytatása ---**

* 1. **A bemeneti tömböt indexeljük. Ez az index mindig nő.**
  2. **Az előálló páros számok tömbjét indexeljük. Akkor nő, ha páros számot találtunk.**
  3. **Ugyanez a páratlanokra.**

1. **Észre kell vennünk, hogy a függvény a tömb átadásakor csak egy pointert fog kapni a tömb elejére, ezért további paraméterként át kell adnunk neki a tömb elemszámát is. És aztán észre kell vennünk azt is, hogy vissza is kell adnunk a hívó számára egy darabszámot. Legalább az egyik új tömb elemszámát (párosak + páratlanok = összes), különben a kiírásnál gondban leszünk.**
2. **Figyeljük meg: a kiírásnál a***kiir()***függvény nem a**paros[]**és**ptln[]**tömbök méretét kapja, hanem az értékes elemek számát! Mind a két tömböt ugyanakkorára kellett méretezni, mint az eredeti tömb (lehetséges, hogy az összes szám páros vagy az összes páratlan), de valószínűleg nincsenek tele a tömbök a válogatás után. A végükön memóriaszemét van.**
3. **mintafeladat: Szöveg /string/ megfordítása**

**A feladatunk megfordítani egy sztringet.**

**Írjuk meg ezt két külön változatban, azaz két külön függvényben! Az egyik függvény két tömböt kapjon: a bemenetet és a kimenetet. A másik függvény pedig végezze el a sztring megfordítását helyben: módosítsa a paraméterként kapott sztringet!**

**Megjegyzések a megoldáshoz:**

1. **A két változat, amellett hogy nagyon eltérő módon kell használni őket, belül is másképp működik. Az új tömbbe másolós verzió érdemi része szinte csak egy másolásból áll, figyelve arra, hogy az egyik sztringben hátrafelé, a másikban előrefelé kell haladni, mert így alakul ki a fordított sorrend. A helyben megfordítós változat a sztring két végén lévő karakterek cserélgetésén alapul – persze csak a közepéig kell eljutni a cserékkel, különben visszafordul a sztring.**
2. **Mindkét esetben előbb meg kell határozni a sztring végét, a lezáró nulla megkeresésével. A függvények nem kapják meg a tömbméretet paraméterként, semmi értelme nem lenne.**

#include <stdio.h>

#include <string.h>

void megfordit\_uj\_tombbe(char \*be, char \*ki) {

/\* elmegyünk a legutolsó karakterig olyan módon,

\* hogy megkeressük a lezáró nullát, és visszalépünk 1-et \*/

int i = strlen(be) - 1;

/\* aztán visszajövünk, és közben a másik tömbbe írjuk a betűket \*/

int j = 0;

while (i >= 0) {

ki[j] = be[i];

++j;

--i;

}

/\* végére kell lezáró nulla! \*/

ki[j] = '\0';

}

void megfordit\_helyben(char \*str) {

/\* meghatározzuk a méretet \*/

int meret = strlen(str);

/\* utána elmegyünk a közepéig, folyamatosan elvégezve a cseréket \*/

for (int i = 0; i != meret/2; ++i) {

/\* str[i] <-> str[meret-1-i] \*/

char temp = str[i];

str[i] = str[meret-1-i];

str[meret-1-i] = temp;

}

/\* a lezáró nulla maradt a helyén. \*/

}

int main(void) {

char elso[] = "hello vilag!";

char masodik[20];

megfordit\_uj\_tombbe(elso, masodik);

printf("[%s] [%s]\n", elso, masodik);

printf("[%s]\n", masodik);

megfordit\_helyben(masodik);

printf("[%s]\n", masodik);

return 0;

}

1. **mintafeladat: Szóköztelenítés – „Mindentegybevéve”**

**Írjunk függvényt, amelyik paraméterként kap egy sztringet. Változtassuk meg olyan módon, hogy a szóközöket kiszedjük belőle! Dolgozzunk a megadott sztringen (tehát a keletkező sztring ne egy másik memóriaterületen legyen, hanem az eredetit változtassuk meg)!**

**Megjegyzések a megoldáshoz:**

1. **Gondoljuk végig:**

* Bele fog férni? Igen, mert ettől csak rövidülhet.
* Belül kell segédtömb? Nem, mert menet közben is csak rövidül.

1. **Le kell írni egymás alá egy példa bemenetet és kimenetet, abból jól látszik a megoldás: két változónk van: honnan másolunk (hányadik karaktert) és hova másolunk (hányadik helyre a tömbben).**
2. #include <stdio.h>
4. void spacetelenit(char \*t) {
5. /\* végig megyünk az összes karakteren \*/
6. /\* "honnan" mindig nő, "hova" csak néha \*/
7. int hova = 0;
8. for (int honnan = 0; t[honnan] != '\0'; ++honnan) {
9. /\* és ami nem space, másoljuk \*/
10. if (t[honnan] != ' ') { /\* NEM 32, hanem ' '! \*/
11. t[hova] = t[honnan];
12. hova++;
13. }
14. }
15. t[hova] = '\0'; /\* lezáró nulla! \*/
16. }
18. int main(void) {
19. char hello[] = "H e l l o, vilag!";
21. spacetelenit(hello);
22. printf("Mindentegybeirok: [%s]\n", hello);
24. return 0;
25. }
26. **mintafeladat: Palindrom (I.)**

**Írjunk programot, amelyik egy sztringről eldönti, hogy palindrom-e: azaz oda-vissza olvasva ugyanaz-e, mint önmaga. Közismert magyar nyelvű palindrom mondat az „Indul a görög aludni.” Hogyan egyszerű megírni, ha módosíthatjuk a sztringet (I.), vagy akkor, ha nem (II.)?   
Mi köze ennek az előző két feladathoz?**

**Megjegyzések a megoldáshoz:**

1. **Az egyszerűség kedvéért először tekintsünk csak kisbetűs, írásjel nélküli mondatokat!**

**„indul a görög aludni”**

**megfordítva**

**„indula görög a ludni”.**

**A szóközök máshol vannak. De ha a szóközöket előbb kiszedjük, akkor már a megfordított sztring, palindrom esetén, teljesen megegyezik az eredetivel.**

1. **A palindrom sztringeket vizsgáló program két részfeladatának megoldását láttuk az előbb – a szóközökön túl az írásjeleket is szűrni kell most.**

**Megoldás nincs, illetve a megoldást nem teszem közzé.**

1. **mintafeladat: Palindrom (II.)**

**Írjunk függvényt, amelyik megmondja egy szövegről, hogy palindrom-e! Ez azt jelenti, hogy visszafelé olvasva ugyanazt adja. Segédtömböt nem használhatunk, hanem az eredeti tömböt vizsgálva kell választ adni.**

**Tehát mint az 5. mintafeladat csak a kapott szöveg (palindrom ?) eredeti (tár)területét kell használni!**

1. **\*\*\*\*\* mintafeladat: „Pointer-lépkedés”**

**Írjuk meg a gyakorlat összes függvényét úgy, hogy sehol nem használunk tömb indexelést, sem \*(ptr + int) alakú kifejezéseket! Helyette mindenhol pointerekkel lépkedünk a tömbökben.**

**Példaként álljon itt a szóköztelenítő:**

#include <stdio.h>

void spacetelenit(char \*t) {

char \*hova = t;

for (char \*honnan = t; \*honnan != '\0'; ++honnan) {

if (\*honnan != ' ')

\*hova++ = \*honnan;

}

\*hova = '\0';

}

int main(void) {

char hello[] = "H e l l o, vilag!";

spacetelenit(hello);

printf("Mindentegybeirok: [%s]\n", hello);

return 0;

}