

## Tartalomjegyzék

- 1 Mutatók (pointer-ek)
  - ◆ 1.1 Ismerkedés a mutatókkal és a debuggerrel
  - ◆ 1.2 Mutató paraméter? függvény
  - ◆ 1.3 Mutató paraméter? függvény 2
  - ◆ 1.4 Bet?raktár-kezel?

## Mutatók (pointer-ek)

A lenti feladatokat a CodeLite-ban oldhatjátok meg legegyszer?bben (ne felejtsetek el minden feladathoz új projektet nyitni!), de akinek a konzolos, gcc-s fordítás a szimpatikusabb úgy is csinálhatja.

### Ismerkedés a mutatókkal és a debuggerrel

Ha mutatókkal végzünk m?veleteket, azt könny? elrontani. Ezért miel?tt belemerülünk, ismerkedjünk meg egy hatékony hibakeresési módszerrel.

A CodeLite tartalmaz beépített debuggert (bug = rajtett hiba a kódban; debugging = ennek keresése/javítása; debugger = ezt segít? eszközkészlet/segédprogram).

Másoljuk be a következ? kódot egy új projekt *main.c*-jébe:

```
#include <stdio.h>

main() {
    int a=5; int b=6; int c = 7;
    int *bb;
    int *aa = &a;
    bb = &b;
    a = 3;
    aa = bb;
    *aa = 8;
    aa = &c;
    *aa = 10;
    *bb = *aa;
}
```

A hibakeresés egyik fontos eszköze hogy a program futását bárhol megállíthassuk és ránézessünk a változók pillanatnyi értékére. A program persze túl gyorsan fut ahhoz hogy futás közben megnyomjunk egy "http://wiki.math.bme.hupause"http://wiki.math.bme.hu gombot, ezért el?re meg kell adni azokat a helyeket (sorokat) ahol szeretnénk hogy megálljon a futás. Ezek az ún.

"http://wiki.math.bme.hutöréspontok"http://wiki.math.bme.hu (breakpoint). Állítsunk be töréspontokat a kód mellett balra a sorszámokat is tartalmazó csíkra (a sor-szám mellé kicsit jobbra) kattintva! Piros pöttyöknek kell megjelenniük a kattintás helyén. Ezek a pöttyök jelzik a töréspontokat. Egy következ? kattintás kitörli a töréspontot.

A programot ne úgy futtassuk mint eddig (Build menü -> Run), hanem a "http://wiki.math.bme.huDebug"http://wiki.math.bme.hu menüpont alatti "http://wiki.math.bme.huStart/Continue debugger"http://wiki.math.bme.hu -rel. A kerek kék "http://wiki.math.bme.huplay-gombbal"http://wiki.math.bme.hu ugorhatunk a következ? töréspontra, a kapcsolósárójeles-zöldnyilas gombokkal pedig sorról sorra haladhatunk akkor is ha nincs ott töréspont, illetve

bemehetünk egy függvényhívás belsejébe, vagy a visszatérése utáni állapotra ugorhatunk.

Futtassuk le lépésről lépésre (szinte minden sorban legyen töréspont) a programot és figyeljük meg a változók értékeinek változását ( a lenti "http://wiki.math.bme.huLocals" http://wiki.math.bme.hu fül alatt fognak látszani)!

Figyeljük meg azt is, hogy a töréspont az aktuális sorban található parancs lefutása előtt vagy után állítja-e meg a futást!

### Mutató paraméter? függvény

Írjunk függvényt, amely két mutatóval kijelölt double számot átlagol! Illesszük a függvényt az alábbi keretprogramhoz.

```
#include <stdio.h>
main() {
    double a, b, res;
    scanf("%lf%lf", &a, &b);
    res = average(&a, &b); /* ez a megírandó függvény */
    printf("%6.2lf%\n", res);
    return 0;
}
```

### Mutató paraméter? függvény 2

Módosítsuk az előző feladatot úgy, hogy a függvény a két double szám átlagát ne függvényértékként adja vissza, hanem egy mutatóval kijelölt harmadik változóba tegye! A keretprogramot is módosítsuk a feladatnak megfelelően.

### Betűraktár-kezelés

A "http://wiki.math.bme.hu/aktar" http://wiki.math.bme.hu nevű globális kétdimenziós tömbben karaktereket tárolunk 7 polcon, minden polcon 9 dobozban.

Írj, függvényeket a fenti programhoz:

- *void betesz(char a, int polc, int doboz):* A raktár megfelelő helyére beírja az *a* változóban kapott karaktert, ha létezik a raktárban az adott polc és doboz.
- *char mivanott(int polc, int doboz):* Visszaadja a raktár adott helyén tárolt karakter értékét, ha érvénytelen értékeket kap akkor írjon ki hibaüzenetet és '\0'-t adjon vissza.
- *void urit(int polc, int doboz):* A raktár megfelelő helyére beírja a '\0' karaktert (ha létezik a polc és doboz, egyébként kiírhat egy hibaüzenetet).
- *void polcoturit(int polc):* A raktár egy teljes sorát kiüríti (használd az előző függvényt!), ha van olyan polc.
- *void urit():* Kiüríti a teljes raktárat, vagyis minden elemnek a '\0' karaktert adja értékül.
- *char\* mutato(int polc, int rekesz) :* Visszaadja az adott elemre mutató pointert. Ha a *polc* vagy a *rekesz* nem megfelelő érték (kiindexelne a tömbből akár alul akár felül) akkor NULL-t adjon vissza.
- *char\* holvan(char s):* az első polctól és első doboztól kezdve végigkeresi a raktárat és visszaadja az első olyan karakter címét (mutatóját) aminek az értéke megegyezik a kapott *s* karakterrel. Ha nem találja a keresett elemet a raktárban, akkor NULL-t adjon vissza.

## Informatika2-2012/Gyakorlat04

- *char \* ures\_helyre\_pakol(char s, int polc)* : a kijelölt polcon belüli első üres ("http://wiki.math.bme.hu\0"http://wiki.math.bme.hu-t tartalmazó) helyre írja be az *s* értékét, és visszaad rá egy mutatót. Ha nem talált üres helyet akkor NULL-t adjon vissza.
- *int polcon\_darab(int polc)*: Összeszámolja a nem "http://wiki.math.bme.hu\0"http://wiki.math.bme.hu karaktereket a polcon és visszaadja a darabszámukat.
- *int leltar()*: Összeszámolja a nem "http://wiki.math.bme.hu\0"http://wiki.math.bme.hu karaktereket és visszaadja a darabszámukat az egész raktárban.

Használd a következő keretprogramot:

```
#include <stdio.h>

int polcok = 7;
int dobozok = 9;
char raktar[polcok][dobozok];

main() {
    urit();
    /* Ide írd még függvényhívásokat amik használják a raktárat! */
    return 0;
}
```