Tartalomjegyzék

Ismétlés

- Függvények, pl:

```
double atlag(double a, int b){
    double x;
    x = (a + b) / 2;
    return x;
}
```

- Írtunk hatványozó függvényt, és egy olyan függvényt ami tömböt kapott bemenetként, így lehet tömböt átadni függvénynek (vagy pointerrel):

```
void fv(int t[], int v){...}
```

- Pointerek:
- Megtanultunk pointereket használni, melyekkel közvetlenül a memóriát kezelhetjük, így változók értékét már függvényekkel is tudjuk módosítani.

```
int *p;      // Létrehozunk egy int pointert p névvel
int a = 5;

p = &a;      // Az a változó pointerét eltároljuk p-ben
scanf("%d", p);   // Így például az a változóba olvasnánk be

printf("%d", *p); // Ezzel pedig az a értékét írnánk ki

*p = 15;     // Pointeren keresztül értéket is átírhatunk
printf("%d", a);  // Így ez mostmár 15-öt adna ki
```

- Azt is megtudtuk, hogy a tömbök pointerek, és nem tudjuk lekérni a méretüket, mert maga a program sem tárolja a méretüket, errõl nekünk kell gondoskodnunk.
- Átírtuk a hatványozó függvényünket, hogy pointereken keresztül kapja az értékeket, majd mégjobban átírtuk hogy egy pointer segítségével az eredményt egy változóba másolja.

Dinamikus memória foglalás

- Függvények:
  - **malloc** megadott byte-ot foglal le a memóriában,
  - **realloc** a már lefoglalt memória-tömb méretét változtatja meg,
  - **calloc** memóriafoglalás és a lefoglalt memória byte-jainek 0-ra állítása egyben,
  - **free** üríti a megadott memória-részt.
  - Ezek mind az stdlib.h-ban találhatók, így mostantól az stdio.h mellett ezt is be fogjuk tölteni.

- Példa malloc és free-re:

```
#include<stdlib.h>
...
int i;          // ciklusváltozónak
int m;          // ebbe olvassuk be a tömb méretét
scanf("%d",&m);
int *vec = (int *)malloc(m * sizeof(int)); // itt foglaljuk le a memóriát a tömbnek
...
for(i=0; i<M; i++){
  vec[i]=i*i;      // majd feltöltjük a tömböt az indexek négyzetével
}
...
```

- A memóriafoglalás rész kifejtve:
  - int *vec -el létrehozunk egy int pointert
  - le szeretnénk foglalni m darab int-nyi helyet
  - a sizeof(int) visszaadja az int méretét byte-okban
  - m * sizeof(int) megadja mennyi byte kell a tömbhöz
  - malloc(m * sizeof(int)) lefoglal ekkora részt a memóriában és visszaadja az erre a területre mutató pointert
  - de amit visszaad az egy void pointer! (hogy más típusú tömböknél is lehessen használni)
  - ezért az (int *)-al ezt a void * -ot átkonvertáljuk int * típusúvá

- Itt megjelent egy új dolog a típuskonverzió, másnéven castolás, pl:

```
...
int a = 2;
int b = 4;
double c = a / (double)b;
...
```

- Ez jól fog értéket adni c-nek, 0.5-öt, annak ellenére hogy **a** és **b** is int-ek.
- A b-t double-é castoljuk (ez még az osztás elõtt megtörténik), így egy egész lesz osztva egy double-el aminek már double az értéke.
- Amire még könnyen használható az egy lebegõpontos szám egészrészének a lekérése, hisz elég ha int-é castoljuk és máris az egészrészt kaptuk.

- Egy utolsó hasznos dolog a memóriafoglalással és a pointerekkel kapcsolatban a NULL pointer.
  - Ha a malloc függvény valami problémába ütközne (pl elfogyott a memória), akkor egy NULL pointert fog visszaadni.
  - Ez az aminek hangzik, egy memóriacím ami valójában nem tárol semmit, egy pointer ami nem mutat semmit.
  - Fõképp hibakezelésre használandó pl:

```
...
int *vec = (int *)malloc(m * sizeof(int));
if(vec == NULL){
  printf("Nem sikerult a memoriafoglalas.");
  return 1;
}
...
```

Feladatok 1

1. Átlaghoz közel

Írjatok programot, ami elõször bekér a felhasználótól egy int-et, nevezzük m-nek, ez határozza meg hogy hány darab további bemenetet fogunk adni. Majd bekér m darab lebegõpontos számot, kiszámolja az átlagukat, és meghatározza, hogy az m szám közül melyik van a legközelebb az átlagukhoz. Végül kiírja ezt a számot.

SPOILER (segítség):

- El kell tárolnotok az értékeket, direkt úgy van kitalálva a feladat, hogy ne lehessen enélkül megoldani.
- Tehát az elsõ lépés, az m beolvasása után, hogy dinamikusan létrehoztok egy ekkora double vagy float tömböt.
- Majd egy ciklussal bekértek m darab számot, ezeket sorban a tömbbe mentitek.
- Meghatározzátok az átlagot akár bekérés közben akár utána.
- Majd egy ciklusban minden elemet összehasonlítotok az átlaggal a minimum- és maximumkereséshez hasonlóan, töltsétek be a math.h-t és akkor használhatjátok az abs (abszolútérték) függvényt.

2. Bet?raktár-kezel?

A "http://wiki.math.bme.huraktar"http://wiki.math.bme.hu nev? 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 megfelel? 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 helyurit(int polc, int doboz)*: A raktár megfelel? 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 el?z? függvényt!), ha van olyan polc.
- *void urit()*: Kiüríti a teljes raktárat, vagyis minden elemnek a '\0' karaktert adja értékül.
- *char* mutato(int polc, int doboz)* : Visszaadja az adott elemre mutató pointert. Ha a *polc* vagy a *doboz* nem megfelel? érték? (kiindexelne a tömbb?l akár alul akár felül) akkor NULL-t adjon vissza.
- *char* holvan(char s)*: az els? polctól és els? doboztól kezdve végigkeresi a raktárat és visszaadja az els? 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.
- *char * ures_helyre_pakol(char s, int polc)* : a kijelölt polcon belüli els? üres ("http://wiki.math.bme.hu\0"http://wiki.math.bme.hu-t tartalmazó) helyre irja be az *s* érékét, 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.

Használjátok ezeket a mintákat (különbözõ nehézségi szintûek, ha valaki nagyobb kihívást szeretne):

- Normal
- Hard
- Ironman

- Megoldás