Tartalomjegyzék

Google Docs

docs

Ismétlés

- Dinamikus memória kezelés::

```
#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
}
...
```

- Típus konverzió:

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

- NULL és lezáró nulla '\0'

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

- Valamint a fõ mondanivalója az elõzõ gyakorlatnak a 2. feladatban rejlett, mégpedig:
  - Összetett kódot érdemes el?re megtervezni.
  - A kódot részenként kell megírni, ezeket a részeket folyamatosan teszteni.
  - Ha már egyszer megírtunk valamit függvényként, akkor felesleges ugyanezt megírni mégegyszer.
  - Tehát új függvények írásánál próbáljuk használni korábbi függvényeinket.

File I/O, argumentumok

- El?adásjegyzet: File I/O
- fopen-el nyitunk meg file-t, fclose-al zárjuk be, FILE* segítségével dolgozunk rajta.
- Ugyanúgy írhatunk file-ba mintha printf-el tennénk csak fprintf-el kell és meg kell adni a file pointerét.
- Ugyanúgy olvashatunk file-ból mintha a terminálból, a felhasználótól olvasnánk be, csak fscanf-el és meg kell adni a file pointerét.
- Karakter tömbökbe olvashatunk a %s-el, ekkor az fscanf az elsõ whitespace karakterig olvas (space, újsor, tab...)
- Egy további példa:

```
#include<stdio.h>

int main(void){
  int i;
  int z;
  char s[100];
  FILE* fp;            // Letrehozzuk a file pointerunket
  fp = fopen("test.txt", "w");  // Megnyitjuk a test.txt-t irasra

  for(i = 0; i < 10; i++){
    fprintf(fp, "%d\n", i * i);  // Negyzetszamokat irunk a file-ba
  }
  fprintf(fp, "Most irunk a file-ba.\n");
  // Csak ugy irtunk valami szoveget a file-ba
  fclose(fp);    // Bezarjuk a file-t

  fp = fopen("test.txt", "r");    // Ujra megnyitjuk, de olvasasra

  for(i = 0; i < 10; i++){
    fscanf(fp, "%d", &z);         // Kiolvasunk a file-bol egy int-et
    printf("%d, ", z);            // Kiirjuk a kepernyore amit kiolvastunk
  }
  fscanf(fp, "%s", s);  // Kiolvasunk egy szot a file-bol
  printf("%s", s);      // Majd ezt ki is irjuk
  fclose(fp);     // Bezarjuk a file-t

  return 0;
}
```

- Argumentumokról: el?adásjegyzet

Feladatok

Átlaghoz közel file-al

- Írjátok meg az elõzõ gyakorlat 1. feladatát úgy, hogy file-ból olvassa be a bemeneteket, és egy másik file-ba mentse a kimenetet.
- A program argumentumként kapja meg:
  - A beolvasandó file nevét
  - a kiírandó file nevét,
  - azt, hogy hány darab szám található a file-ban
- Az elõzõ gyakorlat 1. feladatának egy megoldása itt található:
  - Megoldás
  - Megoldás kommentezve

Typedef és Struktúrák

- Elõadás részlet
- Rövid példák:

Typedef

```
typedef int* int_mutato;  /* ezzel elneveztük "int_mutato"-nak az "int*" típust */
    int szam1 = 42;
    int* szam1_ptr = &szam1;
    int_mutato szam2_ptr = szam1_ptr;
    *szam2_ptr = 23;    // szam1 értéke 23 lesz
```

Struct

```
struct Pont {
   int x;
   int y;
};
```

- Ezzel létrehoztunk egy 2 dimenziós pontot, vagy tekinthetünk rá akár vektorként is.

```
struct Pont p1;
p1.x = 6;
scanf("%d", &p1.y);
```

- Létrehozni hasonlóan tudjuk mint a többi változónkat, használata is egyszerû az adattagokat a . (pont) operátorral érhetjük el, ezek után úgy mûködnek mintha a megfelelõ típusú sima változók lennének

```
typedef struct {
  int x;
  int y;
} Pont;
```

- Kombinálva a typedef-el, egy sokkal esztétikusabb dolgot kaphatunk.

```
Pont p1;
p1.x = 6;
scanf("%d", &p1.y);
```

Feladatok

Egyszer? struktúra kezelés

Hozzatok létre egy struktúrát amiben egy személyrõl adatokat tárolunk (életkor, súly, magasság). Majd hozzatok létre egy ilyen adatszerkezetet, adjatok az adattagoknak értéket, akár scanf-el, akár közvetlenül a programban. Majd írjátok ki a kimenetre a személy születési évét (vehetitek, hogy január 1-én született), és a magassága és tömege hányadosát.

Vektor mûveletek

Használjátok a következõ struktúrát a feladat megoldásához:

```
typedef struct {
  double x;
  double y;
} Vektor;
```

A feladat hasonlít az elõzõ gyakorlat 2. feladatához. Sok egyszerû függvényt kell írni ehhez a struktúrához. Ha megírtatok egy függvényt azt teszteljétek is le a main-ben, utána csináljátok csak a következõt. A függvények:

- *double hossz(Vektor v)*: Számoljátok ki a vektor 2-es normáját, használható a math.h-ban található sqrt függvény a gyökvonáshoz.
- *int egysegvektor_e(Vektor v)*: 1-et ad vissza, ha az adott vektor egységvektor, 0-t ha nem, használjátok hozzá a hossz függvényt. (Érdemes az összehasonlítást nem ==-vel végezni, hanem hogy a különbségük abszolút értéke kisebb-e mint egy kicsi epszilon szám, erre ugye a számítási hibák és a lebegõpontos számok pontatlansága miatt van szükség.)
- *void kiir(Vektor v1)*: kiírja a képerny?re az adott vektor koordinátáit.
- *Vektor egysegvektor_x()*: visszaadja az (1, 0) vektort
- *Vektor egysegvektor_y()*: visszaadja a (0, 1) vektort
- *Vektor vektor(double x, double y)*: visszaadja az (x, y) vektort. (Ezzel könnyebben meg lehetett volna oldani az elõzõ 2-t.)
- *Vektor normalt(Vektor v)*: normál egy vektort, tehát visszaadja a vele megegyezõ irányú egységvektort (Tipp: ismét használható a hossz függvény)
- *Vektor osszeg(Vektor v1, Vektor v2)*: visszaadja a két vektor összegét
- *Vektor kulonbseg(Vektor v1, Vektor v2)*: visszaadja a v1 - v2 vektort
- *double skalarisSzorzat(Vektor v1, Vektor v2)*: visszaadja a két vektor skaláris szorzatát
- *Vektor meroleges(Vektor v)*: visszaad az adott vektorra egy merõleges vektort
- *Vektor meroleges_egyseg(Vektor v)*: visszaad az adott vektorra egy merõleges egységvektort (2 elõzõ függvényt kell használni)
- *Vektor skalarralSzorzat(Vektor v, double c)*: visszaadja a v * c vektort, ahol c skalár
- *int fuggetlenek_e(Vektor v1, Vektor v2)*: 1-et ad vissza ha az adott két vektor lineárisan független, 0-t ha nem

Bónusz

Elõzõ feladathoz:

- *Vektor tombOsszeg(Vektor *vt, int n)*: visszaadja a vt tömbben tárolt n darab vektor összegét
- *Vektor linearisKombinacio(Vektor *vt, double *st, int n)*: vt-ben vektorok st-ben skalárok vannak n darab mindkettõben, a függvény visszaadja a $\sum_{i=0}^{n-1} vt[i]st[i]$ vektort.