3. gyakorlat

Beolvasás, tömbök, beolvasás fájlból

Azt fogjuk rövid példákon kipróbálni, hogy hogyan lehet standard inputról, vagy fájlból beolvasni adatokat, illetve tömböket létrehozni. Majd nézünk egy hosszabb példát arra, hogy ezeket mind ötvözzük, és fájlból olvasunk be vektort (adatsort).

Néhány gyakorlati tanács, amit menetközben, vagy akár előre kipróbálhatunk:

- egy függvény nevének beírása közben a codeblocks felajánlja a már beírt szórészlettel kezdődő függvényneveket, egyúttal mutatja a visszatérési érték típusát (nem minden függvényre működik, de saját függvényeket is kiad)
- a függvény neve utáni zárójelet beírva a függvény szintaxisa jelenik meg (sajnos ez sem mindig működik)
- bővebb leírást könnyen találunk a neten (pl. google: c scanf), a c függvényekről pl. a tutorialspoint.com ad jó leírást egyszerű példákkal
- linux terminálban is lekérdezhetjük a c függvényeket, pl.: man scanf
- globális változókat nem tanácsos használni, ezért a beadandó feladatokban elvárjuk, hogy ne legyenek ilyenek (kivételes esetek vannak, amikor nehéz másként megoldani, pl. ha a quick sort algoritmus esetében az összehasonlítások számát akarjuk meghatározni)
- ha el akarjuk kerülni a program és outputjának megjelenítési problémáit más gépeken (pl. kooplex) állítsuk át UTF-8-ra a karakterkódolást a codeblocksban a Settings - Editor - Encoding settings menűpontban

Input

In [ ]:

```
// a terminálból, vagy esetleg '|' ill. '<' jelek segítségével átadott
// adatokból (azaz a standard inputról)
// így olvashatunk be egész típusú változóba:

    int n;
    printf("n ");
    scanf("%d",&n);    // format string kell és a változó címe
    printf("%d\n",n);  // ellenőrzésül mindjárt ki is írjuk

// Próbáljuk ezt ki egy új projektben:
// pl. "vektorok" néven hozzuk létre, mert tömböket is fogunk létrehozni!

// Miért a változó címe kell? C-ben a függvények argumentumként fv(n) módon
// megadott változóknak csak az értékét kaphatják meg. Nem kapják meg a
// címét, azt a pozíciót, ahol a memóriában tárolódik.
// (Írhatunk oda számot, kifejezést is, amikor nem lenne értelme az
// argumentum címének. A c egyszerűségével, következetességével összhangban
// változó átadásakor sem kaphatja meg annak a címét.) Így viszont egy
// függvény fv(n) módon hívva nem tud a változóba írni. Ezért a scanf-nek a
// változó címét kell átadnunk, és azt a scanf is címként kezeli. Az & jel
// szolgál arra, hogy az n változóról leolvassa a címét, ahol tárolódik.
```

In [ ]:
```c
// Lebegőpontos számok beolvasása

// Mivel a változó címét kell megadnunk, annak az az érdekes következménye,
// hogy a scanf nem tudja, milyen típusú változó az. Így valós szám esetén
// azt is meg kell adnunk, hogy float v. double típusú-e. Nem elég a %f
// típus jelzés, mint a printf-nél, hanem float típus esetén %f kell, double
// esetén pedig %lf használandó.
// Vigyázzunk, ez a printf-nél nincs így, ott mindkét esetben %f kell!

#include <stdio.h>
#include <stdlib.h>

int main()
{
    float s;
    double x;

    printf("s ");
    scanf("%f",&s);
    printf("%f\n",s);  // Beolvasunk egy float változóba, és kiírjuk.

    printf("x ");
    scanf("%lf",&x);
    printf("%f\n",x);  // Beolvasunk egy double változóba, és kiírjuk.

    printf("%15.10f\n",s);
    printf("%15.10f\n",x);

    printf("%f\n",(x-s)*1000);  // Így látjuk is, hogy a float pontatlanabb.

    return 0;
}
```

Vektorok, tömbök létrehozása és használata

In [ ]:
```c
// Kétféleképp hozhatunk létre egyindexes tömböt:


// A veremben:

    double xx[4];

  // Ekkor a rendszer gondoskodik a helyfoglalásról.
  // Értéket adni és kiolvasni egyszerű:
    xx[0]=1.1;             // index: 0...3
    printf("%f\n",xx[1]);


// A halom területen pedig:

    int n=100;
    double *v;  // pointer változó létrehozása, ami memóriacímet tárolhat
    v = (double*)malloc(n*sizeof(double));  // Külön kell gondoskodnunk a
          // helyfoglalásról. Zárójelben a lefoglalandó byteok száma.
          // A malloc visszaadja a lefoglalt terület kezdőpontjának a címét.

  // Ezt felhasználva ugyanúgy tudunk adatokat tárolni a lefoglalt terület
  // egyes celláiban, mint a veremben tárolt esetben:

    v[2]=2.2;              // index: 0...99
    printf("%f\n",v[2]);
    free(v);               // ha már nem használjuk a tömb területét,
```

```
                        // meg kell szüntetni a foglalást

// Mi a különbség a kettő között? Ennek megértéséhez ajánlott az elméleti
// anyagokat átolvasni. Tömören megfogalmazhatjuk az alábbi módon.
// A programok által használható memóriaterület két részre bomlik:

// |- verem -|------------ halom ------------|

// Az egyszerű változók a verem területen foglalódnak. Az xx[4] módon itt
// foglalni is egyszerűbb, mint a halom területen foglalni.
// Viszont ha a tömb mérete nagy, vagy csak a program futásakor derül ki,
// hogy mekkora (pl. ha a program egyik paramétere az), akkor
// a halom területet kell használnunk.

// Az viszont közös a két esetben, hogy a változó neve (xx ill. v)
// a lefoglalt memóriaterület kezdetére mutató pointer lesz.

// Még egy fontos dolog tömbökkel kapcsolatban:
// A fenti program futásakor egyes fordítók figyelmeztetést adnak, hogy
// xx[1] értékét előzetes értékadás nélkül akarjuk használni. Igazuk van!
// Hiszen itt egy elírás van, az xx[0]-nak adtunk csak értéket, tehát
// csak azt olvashatjuk ki. Javítsuk is ki!
// Ez viszont felhívja rá a figyelmet, hogy az indexekkel vigyázni kell,
// mert ha indexként változó értékét írjuk be, vagy egy kifejezést, akkor
// már nem veszi észre a fordító, hogyha helytelen index értékkel akarjuk a
// tömböt címezni. Még akkor sem mindig figyelmeztet, ha a megengedett
// tartományból kicímzünk.
```

In [ ]:

```
// Beolvasás vektorba:

    printf("v[3] ");
    scanf("%lf",&v[3]);  // A korábbihoz hasonlóan a v[3] címét adjuk át.
    printf("%f\n",v[3]);
```

Beolvasás fájlból

In [ ]:

```
// Eddigi programunkat /* és */ jelekkel körítve kommentté tudjuk
// alakítani, így az megmarad mintának, de hatástalan lesz.
/* Tehát e módon lehet könnyen többsoros kommentet csinálni, ahogy e hét
sor is mutatja.
Alatta pedig kipróbálhatjuk az újabb példaprogramot. Ahhoz persze létre
kell hoznunk egy adatok.dat nevű fájlt abban a direktoriban, ahol a c
program van, és írni bele egy számot (a későbbiek kedvéért inkább ötöt).

Beolvasás fájlból így lehetséges:*/

/*    float s;             // az eddigi sorokat kikommenteltük
    ...
    printf("%f\n",v[3]);
*/

    double x;
    FILE* f;                        // fájlpointer változó létrehozása
    f = fopen("adatok.dat", "r");  // az értékadással adjuk meg, hogy
                // melyik fájlt akarjuk megnyitni,
                // és hogy olvasni ("r"), vagy írni ("w") akarunk e bele
    fscanf(f,"%lf", &x);    // az fscanf a scanf-hez hasonlóan használható
    fclose(f);              // ha már nem használjuk, így zárjuk be a fájl!

    printf("%f\n",x);
```

In [ ]:

```
/* Kombináljuk az utóbb tanult két dolgot: vektort olvassunk be fájlból!
Megint kommenteljük ki az előző programrészt!

Praktikus, ha függvénybe szervezzük a beolvasást. Így könyebben
tudjuk felhasználni egy másik programban, ill. egy programban két vagy több
tömböt beolvasni. Ehhez szebb megoldás, ha a fájlnevet változóként adjuk
át, akkor tudjuk a függvényt különböző fájlok beolvasására is használni.
Ehhez meg kell tanulni, hogyan lehet stringet lérehozni:
(legközelebb bővebben lesz szó stringekről)

Építsük fel lépésenként a programot!

Elvileg megoldható a feladat úgy is, hogy a függvényen belül foglalunk
helyet a vektornak, és visszatérési értékként megadjuk a vektor pointerét.
Ilyesmi történt pythonban is, amikor a függvény egy vektort adott visssza.
Azonban a program szervezése szempontjából előnyösebb, ha a függvényen kívül
foglalunk neki helyet, és mi adjuk át a pointert a függvénynek. Ekkor a free
utasítást beírhatjuk ugyanabba a függvénybe, ahol a helyfoglalás történik,
rögtön a helyfoglalás beírásakor is.
*/

void readvec(char *fn, double *v, int n) {
    FILE *f;
    f = fopen(fn, "r");

    for (int i=0; i<n; i++) {
        fscanf(f,"%lf", &v[i]);
    }
    fclose(f);
    return;
}

int main() {
    int n=4;
    double *v;
    v = (double*)malloc(n*sizeof(double));

    readvec("adatok.txt",v,n);

    for (int i=0; i<n; i++) {
        printf("%f\n",v[i]);
    }
    free(v);
    return 0;
}
```