0. LABOR

AZ ÚJ KÖRNYEZET ÉS ISMÉTLÉS

Ennek a kurzusnak a laboralkalma megelőzi az előadást. Ezért az első héten egy extra ismétlős feladatokból álló labort végzünk el, hogy belerázódjunk újra a programozás világába. Psszt! El ne áruljátok a többieknek! 😊

Általános információk

1. iMSC pontok

Az első laboron még nem szerezhető iMSc pont. A többi laboron szerezhető iMSc-s feladatok megoldásainak ***AUT portál***ra *.zip fájlba csomagoltan való feltöltési határideje az aktuális labortól számított három nap. Ha egy feladatban kérdések szerepelnek, a pontok csak akkor fogadhatók el, ha mellékletben egy **igényes** jegyzőkönyv is szerepel a kérdésekre vonatkozó válaszokkal. iMSc pont szerzésére bármely hallgató jogosult, aki az előtte lévő feladatokkal már végzett (laborvezető ellenőrzi a haladást).

2. Házi feladat

Házi feladat pontozási irányelvekről bővebben a „HF_IRANYELVEK.pdf” szól.

Kötelező feladatok

1. Emlékeztető, ismerkedés az új környezettel

Visual Studioban készíts egy új *Empty project* projektet *GettingStarted* néven, amely valósítsa meg az alábbiakat:

- *Olvasson be* egy egész számot (N), majd egy legfeljebb 10 karakterből álló szöveget (S)
- N-t és S-t adja át egy *függvény*nek (F)
- F-ben S-t a standard kimenetre *írja ki* N-szer

2. Forráskód dekompozíció

Az előző feladat F függvényét valósítsd meg kétféleképpen:

1. Külön *printing.cpp* fájlban definiáld, majd abban a fájlban, amiben használod (*main.cpp*), *extern* kulcsszóval jelezd a fordítónak, hogy egy másik *.cpp-fájlban keresse a definíciót
2. Töröld az *extern*-t, helyette hozz létre egy *printing.h*-t
    a. csak a deklaráció szerepeljen benne, a definíció maradjon a *printing.cpp*-ben
    b. oldd meg, hogy többszöri include-olás esetén ne legyen többszörös deklaráció (Emlékeztető: https://en.cppreference.com/w/cpp/preprocessor/include)

3. Adatszerkezet létrehozása és dinamikus tárolása

Ebben és a következő feladatban átismételjük a C nyelvi emlékeinket.

Készíts egy programot, amely **diákok** adatait tárolja dinamikusan! Az adatok egy Diak nevű **struct**-ban tárolódjanak, amely a következő mezőkből áll:

- char nev[50]: a diák neve
- int eletkor: a diák életkora

- float atlag: a diák tanulmányi átlaga

A program:

- Bekéri a diákok számát.
- Dinamikusan foglal memóriát Diak* típusú tömb számára.
- Bekéri az összes diák adatát, és eltárolja őket.
- Kiírja az összes diák adatait.
- A végén felszabadítja a foglalt memóriát.

Példa futás:

```
Hány diákot szeretnél megadni? 2
Add meg az 1. diák nevét: Kiss Péter
Add meg az életkorát: 19
Add meg az átlagát: 4.5
Add meg a 2. diák nevét: Nagy Anna
Add meg az életkorát: 20
Add meg az átlagát: 3.8

Diákok listája:
1. Kiss Péter, 19 éves, átlag: 4.50
2. Nagy Anna, 20 éves, átlag: 3.80
```

Ld. még a feladatok utáni ötleteket a megvalósításhoz!

4. Egy diák adatainak módosítása

Bővítsd az előző programot egy függvénnyel, amely módosítja egy adott diák életkorát és átlagát!

- A függvény neve modosit_diak legyen.
- A módosítandó diák **pointerként** érkezik a függvényhez.
- A program kérje be, hogy melyik diák adatait kell módosítani (index alapján)!
- A módosítás után listázza ki az összes diákot.

Példa futás:

```
Melyik diák adatait szeretnéd módosítani? (1-től kezdődik) 2
Add meg az új életkort: 21
Add meg az új átlagot: 4.2

Diákok listája:
1. Kiss Péter, 19 éves, átlag: 4.50
2. Nagy Anna, 21 éves, átlag: 4.20
```

Opcionális feladat

5. Fájlból olvasás

Készíts egy függvényt, amely a diákok adatait fájlból olvassa be. Minden diák külön sorban szerepel, az adatokat pontosvessző határolja. Feltételezhetjük, hogy a fájlban nincs a struktúrában definiáltnál hosszabb vagy más módon hibás adat.

Gondold végig:

- A függvénynek a beolvasott darabszámot is vissza kell adnia. Miért?
- Mi a leghatékonyabb módja, hogy az ismeretlen darabszámú elemet egy tömbbe olvassuk? Hányszor kell ehhez a fájlon végigmenni?

Ötletek a feladatok megvalósításához

1. scanf használata is lehetséges fgets helyett

A scanf("%[^\n]", str); formátum használható arra, hogy egy teljes sort beolvassunk, egészen az újsor (\n) karakterig. Ez a módszer azonban veszélyes, mert nem korlátozza a beolvasott karakterek számát, így túlcsordulás történhet. Ehelyett biztonságosabb, ha scanf("%49[^\n]", str); módon megadunk egy maximális hosszt, amely a str tömb méreténél eggyel kisebb kell legyen.

---

2. scanf_s szabványossága

A scanf_s a Microsoft által bevezetett, biztonságosabb változata a scanf-nek, amely a puffer méretét is elvárja, például scanf_s("%49s", str, sizeof(str));. Ez a függvény csak a **C11 szabvány opcionális kiegészítéseként** érhető el, és főként Windows/MSVC környezetben támogatott. Más fordítóprogramok, például GCC vagy Clang, alapértelmezetten nem implementálják, így hordozhatósági problémát okozhat.

---

3. strtok működése és paraméterei

A strtok egy karakterlánc darabolására szolgál, ahol az első paraméter a darabolandó string, a második pedig az elválasztó karakter(ek) listája (";"). Az első híváskor a teljes stringet kell megadni, utána pedig NULL-t, hogy az előző állapotot folytassa. Példa:

```
char str[] = "alma;korte;banan";
char* token = strtok(str, ";");
while (token != NULL) {
    printf("%s\n", token);
    token = strtok(NULL, ";");
}
```

Ez a kód az alábbi kimenetet adja:

```
alma
korte
banan
```

---

4. strtok_s változat

A strtok_s a strtok biztonságosabb változata, amely egy extra **context** változót használ az állapot követésére. Ennek köszönhetően **több szálban is biztonságosan használható**. Példa:

```
char str[] = "alma;korte;banan";
char* context = NULL;
char* token = strtok_s(str, ";", &context);
while (token != NULL) {
    printf("%s\n", token);
    token = strtok_s(NULL, ";", &context);
}
```

A működése megegyezik a strtok-éval, de biztonságosabb, mert az állapot nem egy globális változóban, hanem a context paraméterben tárolódik. Azonban ez sem szabványos kiterjesztés.