PROGRAMOZÁS ALAPJAI 2. (VIAUAA01)

2. LABOR IMSC FELADAT

POINTEREK ALKALMAZÁSAI

KOVÁCS LEVENTE (F5UHYT)

1. Pointer, mint ismeretlen törzsű függvény hívásának eszköze

- A *compare* függvényben elhelyezett töréspontnál láthatjuk, hogy a pointerek beállítása a *void **-ként kapott pointerek *double **-ra kasztolásával megfelelő értékadás történik.
- A függvénypointerek lehetővé teszik, hogy a sort függvény generikusan, bármilyen adattípusra működjön. Magyarul a sort függvénynek nem kell tudnia, milyen típusú adatokat rendez, amennyiben egy megfelelő compare függvényt (standard return értékű, adott adattípusokat összehasonlító) függvénypointeren keresztül átadunk neki.
- Függvénypointert úgy állíthatunk elő, hogy egyszerűen a függvénypointer nevét az annak megfelelő paraméterlistával leírjuk. Jelen esetben a sort függvények paraméterként átvesznek egy *int*-et visszaadó, két értéket *void **-ként átvevő (majd később megfelelő típusra kasztoló) függvényt, amit lokálisan *compar* néven tudnak meghívni.

```
int (*compar)(const void *, const void *);
```

- A *void ** típus a fentebb leírt generikus működés következménye. Ha az összehasonlító függvény nem tudja, milyen típusú adatokkal dolgozik, akkor nem is tudja az összehasonlítandó adatot típus szerint átadni az összehasonlító függvénynek.

3. Pointer, mint cím szerinti paraméterátadás eszköze

- A *scanf* függvény pointereket vár paraméterként, ugyanis a beolvasott értékeket közvetlenül a pointer által mutatott memóriaterületre írja be. Ezért vezethet programozói hiba könnyen segmentation fault hibához, ha az átadott pointer nem lefoglalt, vagy nem megfelelő méretű memóriaterületre mutat.
- A tömbelemek bekérését látszólag többféleképpen is meg lehet valósítani, de valójában mindegyik módszer ugyanazt jelenti:
    - Az első és legegyszerűbb a tömb megfelelő indexelésű elemének a címét átadni
    - A második esetben nem tömbként, hanem a *double* tömb egyik *double* típusú változójaként tekintünk az elemre, aminek vesszük a címét a & operátorral
    - A harmadik eset a pointeraritmetikai tulajdonságokat használja ki. Itt a „tömbre” csupán egy *double* pointerként tekintünk és a tömb i. elemét a tömb pointer i-vel való eltolásával kapjuk. Itt a fordító a tömb kezdő memóriacímét *i * sizeof(double)* mérettel tolja el.

4. Biztonságos szövegbevitel

A buffer overflow olyan szoftverhiba (lehetséges biztonsági rés), amikor egy fix hosszúságú lefoglalt memóriaterületre íráskor a program nem ellenőrzi a beírandó adat hosszát, így a beírt adat (amennyiben nagyobb, mint a megadott memóriaterület) olyan területre kerülhet, ami nem annak, vagy egyáltalán nem volt lefoglalva. Így szélsőséges esetben akár, ha kritikus (pl. operációs rendszer által fenntartott, vagy autentikációhoz szükséges adatnak fenntartott) területre próbálunk írni, a program le is állhat, vagy illetéktelen adatmódosítás, adathozzáférés is történhet. A *scanf* standard C függvény ez ellen nem véd, nem figyeli a beírni kívánt adat hosszát.

A Microsoft C runtime library részét képzik olyan biztonsági funkciókat tartalmazó függvények (pl. *scanf_s*), melyek az ilyen típusú buffer overflow sebezhetőségeket próbálják kiküszöbölni. A *scanf_s* függvény több ellenőrzést végez arra vonatkozóan, hogy a stdin-ről olvasott értékek megfelelő formátumúak, méretűek.

5. Saját rendezőfüggvény

Az egyszerűség kedvéért egy optimalizált bubble sort algoritmust valósítottam meg. A *myBubbleSort* függvény egy generikus buborékrendező algoritmust valósít meg. Bemeneti paraméterei a default *qsort*-tal megegyeznek:

- `void *base`: a tömb basepointere
- `size_t arr_size`: a tömb elemszáma
- `size_t elem_size`: egy elem memóriában foglalt mérete (bájtokban)
- `int (*compar)(const void *, const void *)`: a konvenciónak megfelelő visszatérési értékű összehasonlító függvény.

A függvény fejléce a következő:

```
void myBubbleSort(void *base, size_t arr_size, size_t elem_size, int (*compar)(const void *, const void *))
```

A függvényt például a következőképpen lehet meghívni:

```
myBubbleSort(array2, arr_size, sizeof(double), compare);
```

Az elemek cseréjéhez egy swap segédfüggvény is készült, mely egyszerűen megcseréli a két pointer által mutatott memóriaterületeken található adatokat.

```
 1. int swap(void *a, void *b, size_t elem_size)
 2. {
 3.     void *temp = malloc(elem_size);
 4.
 5.     memcpy(temp, a, elem_size);
 6.     memcpy(a, b, elem_size);
 7.     memcpy(b, temp, elem_size);
 8.
 9.     free(temp);
10. }
```

6. Megjegyzések

A könnyű tesztelhetőség érdekében a kódban implementáltam egy *fill_rand* függvényt, mely feltölti az adott tömböt 0-255-ig egész számokkal. A függvény a *rand* függvényt használja ehhez, melynek a seedjét a *time* függvény állítja be.

Érdekességképpen érdemesnek tartottam összehasonlítani a beépített quicksort és a saját improved bubble sort rendezőket futásidő tekintetében. Kis számokra a futásidő hasonló, de nagyobb tömbméretekre egyre nagyobb a két módszer közötti eltérés.