5. labor kis ZH, 2016. november 29., szerda 10-12

NÉV (olvasható): NEPTUN kód:

- Definiálj felsorolt típust, amely negatív, semleges és pozitív töltés megkülönböztetésére alkalmas. (1p)
- Definiálj struktúrát, amely egy részecske két adatát tudja tárolni: tömeg és töltés típusa. (1p)
- Írj függvényt, amely egy részecskéket tartalmazó tömböt vesz át, és a következő módon sorba rendezi ezeket: először az összes negatív töltésű, majd az összes semleges töltésű végül az összes pozitív töltésű részecske legyen a tömbben. Adott töltéstípuson belül a részecskék tömeg szerint növekvő sorrendben legyenek! Az eredeti tömbben végezd a rendezést, segédtömböt nem használhatsz! A rendezést megvalósíthatod kézzel írt algoritmussal vagy a könyvtári qsorttal is. Ha a részecskék töltését nem veszed figyelembe, és egyszerűen a tömegük szerint rendezed őket sorba, akkor 1 pontot veszítesz. (4p)
- Írj függvényt, amely megszámolja, hogy hány különböző típusú részecske van a paraméterként átvett, rendezett tömbben. Két szomszédos részecske akkor számít különböző típusúnak, ha vagy eltérő a töltésük típusa, vagy a második tömege több mint 1%-kal nagyobb, mint az őt megelőző részecske tömege. (Feltételezheted, hogy a bemenő tömb helyesen rendezett.) (2p)
- Írj main függvényt, melyben létrehozol egy 5 elemű részecsketömböt, melyet kezdőértékekkel inicializálsz! Rendezd a tömböt, majd számold meg, és írd ki, hogy hány különböző típusú részecske van a tömbben! (2p)

max. 10p, megoldási idő: 20 perc

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

typedef enum{negativ, semleges, pozitiv} ttipus; // 1p

typedef struct {
    double m;
    ttipus tip;
} reszecske; // 1p

void sajatrendez(reszecske *t, int n) { // 4p
    int i, j, minindex;
    for (i = 0; i < n - 1; i++) {
        minindex = i;
        for (j = i + 1; j < n; j++)
            if (t[j].tip < t[minindex].tip // ha csak tömeg szerint rendez, -1p
                  || (t[j].tip == t[minindex].tip && t[j].m < t[minindex].m))
                minindex = j;
        if (minindex != i) {
            reszecske temp = t[i];
            t[i] = t[minindex];
            t[minindex] = temp;
        }
    }
}

int hasonlit(const void *p, const void *q) { // jó fejléc: 1p
    const reszecske *a = (const reszecske*)p;
    const reszecske *b = (const reszecske*)q; // típuskonverzió: 1p
    if (a->tip < b->tip) // típust és tömeget is néz: 1p, egyébként 0
        return -1;
    if (a->tip > b->tip)
        return 1;
    if (a->m < b->m)
```

```
        return -1;
    if (a->m > b->m)
        return 1;
    return 0;
}

void rendezqsort(reszecske *t, int n) {
    qsort(t, n, sizeof(reszecske), hasonlit); // 1p
}

int szamol(const reszecske *t, int n) { // 2p
    int db = 1, i;
    for (i = 1; i < n; i++)
        if (t[i].tip != t[i - 1].tip || t[i].m>t[i - 1].m*1.01)
            db++;
    return db;
}

int main(){
    reszecske t[5] = { {3, semleges}, {2, pozitiv}, {2, negativ}, // 1p
                       {2, pozitiv}, {2,semleges} };
    // sajatrendez(t, 5);
    rendezqsort(t, 5); // A két függvényhívás, ha mindkettő jó, 1p
    printf("%d", szamol(t, 5));
    return 0;
}
```