3. labor kis ZH, 2018. április 12., csütörtök 10-12

NÉV (olvasható): NEPTUN kód:

Adott a következő struktúra: `struct adat { long ido; double ertek; };`, amely egy mérés időpontját és értékét tárolja.
Készíts dinamikus tömb osztályt ilyen struktúrák tárolására! Az osztály két adattagja a dinamikusan foglalt tömbre mutató pointer, valamit a tömb mérete.
- Készíts egyparaméteres konstruktort, amely a létrehozandó tömb méretét kapja paraméterként, melynek legyen alapértelmezett értéke 100! Ha a megadott méret 1-nél kisebb, a konstruktor dobjon std::range_error típusú hibát! A konstruktor hozza létre a megadott méretű tömböt, állítsa be az osztály tagváltozóit!
- Készíts destruktort, másoló konstruktort, értékadás operátort, index operátort, méretet lekérdező getter függvényt!
- Készíts push_back függvényt, amely egy időpontot és egy értéket kap paraméterként, megnöveli a tömb méretét eggyel (lefoglal egy eggyel nagyobb tömböt, átmásolja a régi tömb tartalmát, törli a régi tömböt és az újra állítja a pointert), és a megnövelt tömb végére beteszi a paraméterként átvett adatot.
- Készíts kiíró operátort (<<), amely az ostream-re írja a tömböt: először a méretét, majd az idő-érték párokat szóközzel elválasztva.
- Bónusz feladat +2 pontért: készíts index operátort, amely egy időpontot kap paraméterként, és az időponthoz tartozó érték címét adja vissza. Ha nincs ilyen időpont, NULL pointert adjon! (Természetesen ez a függvény ütközik a másik [] operátorral, de ez most nem baj.)

max. 10+2p, megoldási idő: 20 perc

```
#include <iostream>
#include <stdexcept>

struct adat { long ido; double ertek; };

class Dintomb {
    adat *t;
    int n;
public:
    Dintomb(int n = 100) :n(n) {
        if (n < 1) throw std::range_error("Dintomb::Dintomb: n<1");
        t = new adat[n];
    }
    ~Dintomb() { delete[]t; }
    Dintomb(const Dintomb &m) :t(NULL) {
        *this = m;
    }
    const Dintomb &operator = (const Dintomb &m) {
        if (this == &m) return *this;
        delete[]t;
        n = m.n;
        t = new adat[n];
        for (int i = 0; i < n; i++) t[i] = m.t[i];
        return *this;
    }
    adat &operator[](int i) {
        return t[i];
    }
    int getSize()const { return n; }
    void push_back(long ido, double ertek) {
        adat *uj = new adat[n + 1];
        for (int i = 0; i < n; i++)
            uj[i] = t[i];
        uj[n].ido = ido;
        uj[n].ertek = ertek;
```

```
        n++;
        delete[]t;
        t = uj;
    }
    double *operator[](long ido) {
        for (int i = 0; i < n; i++)
            if (t[i].ido == ido)
                return &t[i].ertek;
        return NULL;
    }
};

std::ostream & operator<<(std::ostream &os, Dintomb & dt) {
    os << dt.getSize();
    for (int i = 0; i < dt.getSize(); i++)
        os << ' ' << dt[i].ido << ' ' << dt[i].ertek;
    return os;
}
```