3. labor kis ZH, 2019. ÁPRILIS 4., csütörtök 10-12

NÉV (olvasható): NEPTUN kód:

Készíts *double* értékek tárolására alkalmas *verem* osztályt! Az osztály dinamikus tömbben tárolja az adatokat. Készíts

- alapértelmezett
- és másoló konstruktorokat,
- destruktort,
- láncolható = operátort,
- *pop* függvényt, amely kiveszi és visszaadja a *verem*be utoljára betett értéket. Részletesebben: a dinamikus tömb utolsó elemét kell visszaadni, de ezzel együtt ezt az elemet el is kell távolítani a veremből, azaz eggyel kisebb dinamikus tömbbe áttenni a többi elemet. Ha üres veremre hívják meg a *pop* függvényt, akkor dobjon *std::underflow_error* hibát!
- + operátort, amely double+verem módon működik (verem+double nem jó), és a visszaadott verembe beleteszi akapott *double* értéket. Tételezd fel, hogy a *verem* osztálynak van *push* nevű tagfüggvénye, amely egy *double* értéket kap paraméterként, és beleteszi a *verem*be. A *push* függvényt nem kell megvalósítanod, de felhasználhatod.
- *main* függvényt, amelyben kipróbálod a *verem* összes funkcióját, és elkapod az esetleg dobott hibát is.

max. 10p, megoldási idő: 20 perc

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

class Verem { // Ha nem jó, -2p
    double *t;
    unsigned n;
public:
    // tömb átméretezés: 2p (kell a másoló ctor-ban, =-ben és a pop-ban, de csak egyszer jár érte
a pont)
    Verem() :t{ nullptr }, n{ 0 } {} // 0,5p
    Verem(const Verem & masik) :t{ nullptr }, n{ 0 } { *this = masik; } // deklaráció: 1p
(go/nogo), működés: 1p
    ~Verem() { delete[]t; } // 0,5p
    const Verem & operator=(const Verem & masik) { // deklaráció: 1p (go/nogo), működés 1p
        if (this != &masik) { // Ha nincs, -1p
            if (n != masik.n) {
                n = masik.n;
                delete[]t;
                t = new double[n];
            }
            for (unsigned i = 0; i < n; i++)
                t[i] = masik.t[i];
        }
        return *this;
    }
    double pop() {
        if (n == 0)
            throw std::underflow_error("Verem::pop: empty"); // Hibakezelés: 1p (go/nogo)
        n--;
        double ret = t[n]; // visszaadott érték helyes kezelése: 1p
        double * temp = new double[n]; // működés: 1p
        for (unsigned i = 0; i < n; i++)
            temp[i] = t[i];
        delete[]t;
        t = temp;
```

```
        return ret;
    }
    void push(double uj) { // nem kellett
        double * temp = new double[n + 1];
        for (unsigned i = 0; i < n; i++)
            temp[i] = t[i];
        temp[n] = uj;
        n++;
        delete[]t;
        t = temp;
    }
    friend std::ostream & operator<<(std::ostream & os, const Verem & v) { // nem kellett
        for (unsigned i = 0; i < v.n; i++)
            os << v.t[i] << (i == v.n - 1 ? '\n' : ' ');
        return os;
    }
};

Verem operator+(double uj, Verem v) { // deklaráció: 1p (go/nogo)
    v.push(uj); // megvalósítás: 1 bónuszpont
    return v;
}

int main() {
    try { // hibakezelés: 1p (go/nogo)
        Verem a;
        a.push(3.14);
        a.push(2.71);
        std::cout << "a = " << a;
        Verem b(a);
        std::cout << "b = " << b;
        std::cout << a.pop() << std::endl; // 0,5p
        std::cout << "a = " << a;
        b = 9.81 + a; // 0,5p
        std::cout << "b = " << b;
    }
    catch (const std::underflow_error & hiba) {
        std::cerr << "Hiba:" << hiba.what() << std::endl;
    }
}
```