

NÉV (olvasható):

NEPTUN kód:

max. 10p, megoldási idő: 20 perc

Egy biológus egy hím kísérleti szarvasbogar mozgását szeretné elemezni. A bogár az origóból indul, pozícióját másodpercenként mérték a laboratórium asztalához rögzített descartes-i koordináta-rendszerben, centiméter egységben, az érvényes tartományon valamelyik koordináta nemnegatív. Ezt a rögzített adatsorozatot kapja majd a programunk a szabványos bemeneten szóközzel elválasztott valós számpárok formájában. Az utolsó érvényes adatot két negatív érték követi.

Készíts programot, amely egy `std::vector`-ban tárolja a mért adatokat. A koordinátákat adott `Vekt2D` típusú objektumokban kell tárolni, értelemszerűen az origóból az adott pontba mutató vektor. A `Vekt2D` típust nem kell létrehozunk; publikus valós tagváltozókkal bír (x , y), és rendelkezik aritmetikai, valamint kiíró és beolvasó operátorokkal. Az `std::vector` ilyen `Vekt2D`-ket tároljon! Szükség szerint létrehozhat szegítő függvényt vagy osztályt.

a) Olvasd be, és tárold el a nyomkövető által szolgáltatott koordinátákat!

b) Váltsd át méterbe az összes koordinátát, és az eredményt helyezd egy új tömbbe (=vektorba). Ha STL algoritmust használ, és ezt helyesen teszed, +1 bónuszpontot kapsz.

c) Hozzuk létre a bogár sebességvektorának tömbjét, majd a skalár sebességértékek tömbjét. (Természetesen ez utóbbi már valóság tömbje, amibe a vektorok abszolút értéke kerül.) Ha STL algoritmust használ, és ezt helyesen teszed, +1 bónuszpontot kapsz. (Ki is kell számolni a sebességeket!)

d) Az útja során egyszer állt csak meg a szarvasbogar (két egymást követő mért pont azonos), ahol igen kis koncentrációban nőstény feromont helyeztünk el. Hányadik másodpercnél történt ez? Mennyi volt a bogár csúcsebessége? Ha STL algoritmust használ, és ezt helyesen teszed, +1 bónuszpontot kapsz.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>

struct Vekt2D {
    double x, y;
    Vekt2D(double X = 0.0, double Y = 0.0) : x{ X }, y{ Y } {}
    Vekt2D operator-(const Vekt2D& v) const { return { x - v.x, y - v.y }; }
};

std::istream& operator>>(std::istream& is, Vekt2D& v) { is >> v.x >> v.y; return is; }
std::ostream& operator<<(std::ostream& os, const Vekt2D& v) { os << v.x << ' ' << v.y;
return os; }
//v/100 is jo
Vekt2D cm2m(const Vekt2D& v) { return Vekt2D(v.x / 100, v.y / 100); } // function
class hossz { //functor
public: double operator()(const Vekt2D& u) { return std::sqrt(u.x * u.x + u.y * u.y); }
};

int main() {
    Vekt2D a;
    std::vector<Vekt2D> s; // 1p
    std::cin >> a; // 1p
    while (a.x >= 0 || a.y >= 0) { // 1p
        s.push_back(a); // 1p
        std::cin >> a;
    }
    std::vector<Vekt2D> s_m(s.size()); // 1p
    std::transform(s.begin(), s.end(), s_m.begin(), cm2m); // +1p // 2p
    std::vector<Vekt2D> vv(s_m.size());
    std::adjacent_difference(s_m.begin(), s_m.end(), vv.begin()); // +1p // 2p
    std::vector<double> v(vv.size());
    std::transform(vv.begin(), vv.end(), v.begin(), hossz()); // 2p
    std::vector<double>::iterator megall = std::min_element(v.begin() + 1, v.end()),
        max_seb = std::max_element(v.begin(), v.end()); // +1p
    std::cout << "megallas " << megall - v.begin() << " masodperc\n"; // 1p
    std::cout << "leggyorsabb " << *max_seb; // 1p
}
```