7. LABOR

OPERÁTOROK TÚLTERHELÉSE

Általános információk

1 IMSC pont jár a 3. és 4. feladatok helyes megoldására

Kötelező feladatok

1. Malacpersely osztályt kell készíteni az operátorok gyakorlására. Az osztály a mellékelt kódnak megfelelően int típusban tárolja az eddig bedobott pénzek összegét (penztartalom). Az első kódváltozat még nem használ operátorokat, ezt szeretnénk intuitívebbé tenni. A MalacperselyTest osztályban sorra haladj végig a programkódok kikommentelésével, hogy teszteld sorra az elkészített operátorokat.
   - A **`persely1 = persely1 + 100`** utasítás hatására kerüljön a visszaadott perselybe plussz 100 Ft. Valósítsd meg globális függvénnyel, vagy tagfüggvénnyel.
   - A cout-ra való kiírásnál azt szeretnénk, hogy az egyes perselyek a forintos értéket írják ki, ld. a kommentet (//Ezt akarjuk latni: "A persely1 erteke 100 Ft." - A Ft. -ot nem mi irtuk moge!)
   - A **`persely2 += 200`** hatására a persely2 értéke növekedjen 200 Ft-tal. Mi legyen a visszatérési típus, hogy így is működjön: **`cout << "A persely2 erteke " << (persely2+=200) << endl;`**
   - A **`persely1 += persely2`** hatására az összes persely2-beli pénz kerüljön át a persely1-be (ld. az **`atont`** függvény)
   - Az utolsó kiíratásnál a láncolást látod (**`persely1 + persely2 + persely3 + 100`**). Egy új operátorra is szükséged lesz, valamint mindenképp végig kell gondolnod esetleg a korábbi operátorok visszatérési típusát!

2. Dinamikus tömb (*Vector*) osztály: közös értelmezés

Egy egész számokat tároló dinamikusan nyújtozkodó tömböt kell kiegészítened. A megértéshez bemelegítésként közösen értelmezzétek a megadott *Vector* solution állományait!

1. Hogyan lehetne leegyszerűsíteni a másoló konstruktor definiálását?
2. Miért (konstans) referenciatípus a paraméter és a visszatérési érték az egyes függvényeknél?
3. Mi a helyzet a *v=v* kifejezés esetén?
4. Mivel térjen vissza a [] operátor?
5. Milyen változtatásokra lenne szükség, ha az int típusú változók helyett saját osztályt példányosító objektumokat tárolna a tömb?
6. Miért vannak külön névtérben az *ascend* és a *descend* predikátum függvények?
7. Miért nem statikus függvény a *sort*?
8. Miért *friend* a kiíratő operátor?
9. Szükség van az *at* függvényekre?
10. Értelmezzétek a *sort* függvény pointer paraméterét!

3. Dinamikus tömb (*Vector*) osztály: tagfüggvények definiálása – első rész

Az alábbi függvényeket mindenkinek kötelező definiálnia:

- `ostream& operator<<(ostream& os, const Vector& v)` – A példakódban levő kommentnél látod, mi az elvárás az elemek kiírásánál.
- `Vector& Vector::operator=(const Vector& theOther)` – Dinamikus adattagot tartalmazó osztály, a bitről bitre másolás nem lenne jó.
- `Vector::Vector(const Vector& theOther)` – Ezen a ponton mindössze 2 sorral meg lehet ezt írni.
- `int& Vector::at(unsigned position)`
- `const int& Vector::at(unsigned position) const`
- `int& Vector::operator[](unsigned position)`
- `const int& Vector::operator[](unsigned position) const`

Javaslatok:

- A másoló konstruktor definiálásakor érdemes az operator=()-t felhasználni
- Az operator[]-nél pedig az at() függvényt célszerű használni

Opcionális (iMSC) feladat

4. Dinamikus tömb (*Vector*) osztály: tagfüggvények definiálása – második rész

Az alábbi függvények definiálása opcionális (de ajánlott):

- `void Vector::operator*=(unsigned right)`
- `void Vector::operator+=(unsigned right)`
- `istream& operator>>(istream& is, Vector& v)`
- `bool Sorters::ascend(const int& a, const int& b)`
- `bool Sorters::descend(const int& a, const int& b)`
- `void Vector::sort(bool(*predicate)(const int& a, const int& b))`

  Javaslat: sort()-nál egyszerű és elegendően hatékony a minimum kiválasztásos rendezés, de az insertion sort is alkalmas választás