500.AA Megoldo Arisztid 500.A

Programozás alapjai 2. (inf.) 2. ZH 2018.05.07. lab. hiányzás: 1+2	a/a/1
ABCD123 **a/1**	kZH: 4 E:3

Minden beadandó megoldását a feladatlapra, a feladat után írja! Készíthet piszkozatot, de csak a feladatlapra írt megoldásokat értékeljük! Jelölje a táblázatban, ha oldott meg IMSC feladatot. Ezt csak az alapfeladatok 75%-os teljesítése mellett értékeljük.

*A megoldások során feltételezheti, hogy minden szükséges input adat az előírt formátumban rendelkezésre áll. A feladatok megoldásához csak a letölthető C, C++ és STL összefoglaló használható. Elektronikus eszköz (pl. tablet, notebook, mobiltelefon) nem használható. A feladatokat **figyelmesen olvassa** el! **Ne írjon felesleges** függvényeket ill. kódot! Súlyos hibákért (pl. privát változó külső elérése, memóriakezelési hiba, stb.) pontot vonunk le.*

Az első feladatban minimum 5 pontot el kell érnie ahhoz, hogy a többi feladatot értékeljük..

1. feladat: Beugró. Használhat STL tárolót és algoritmust! Σ 10 pont

a) Jelölje (pl. karikázza be), hogy az állítás igaz (I), vagy hamis (H) a C++ nyelvre! Minden bejelölt válasz 0.5 pont, ha helyes, -0.5p pont, ha hibás! Az esetleges negatív eredmény is összeadódik a többi részfeladatra kapott ponttal. (2p)

Az implicit értékadó operátor nem hívja meg az ősosztály értékadó operátorát.
Statikus tagfüggvénynek nincs this pointere.
Az iterátor egy általánosított pointer.
A konstruktor előbb hajtja végre a programozott törzset, és csak ezután hívja az ősosztályok konstruktorát.

b) Az alábbi függvénnyel egész számokat tartalmazó vektorból szeretnénk eltávolítani az ismétlődő elemeket úgy, hogy azokból csak egy maradjon a vektorban! Tesztéleskor észrevettük, hogy a kódrészlet nem tökéletes. Javítsa ki a függvényt! (2p)

```
void tisztit(std::vector<int>& vec) {
      sort(vec.begin(), vec.end());

                                        unique(vec.begin(), vec.end());


}
```

c) Adott az alábbi deklaráció:

```
class Tarolo : public std::vector<double> {
   std::string duma;
} t1;
```

Jelölje (pl. karikázza be), hogy az állítás igaz (I), vagy hamis (H)! Minden bejelölt válasz 0.5 pont, ha helyes, -0.5p pont, ha hibás! Az esetleges negatív eredmény is összeadódik a többi részfeladatra kapott ponttal. (2p)

`Tarolo t2 = t1;` utasítás helyes.
`t1 = t1 = t2;` utasítás hibás, mert az implicit operator= nem támogatja a többszörös értékadást.
`t1[4] = 56.78;` utasítás hibás, mert a `Tarolo` nem indexelhető. Meg kellene valósítani az operator[] függvényt.
`t1.push_back(45);` utasítás helyes.

d) Készítsen az *std::for_each* algoritmushoz hasonló generikus algoritmust (*for_each2nd*), ami sorozattároló minden **második** elemével meghív egy paraméterként kapott függvényt! Írjon rövid kódrészletet, melyben létrehoz egy sorozattárolót, mely egész elemeket tartalmaz, és a *for_each2nd* sablon segítségével kiírja minden második elemét a szabványos kimenetre! (4p)

500.AA Megoldo Arisztid 500.A

2. Feladat **10 pont**

a) Készítsen adapter sablont (*PhyTomb*), ami minden olyan szabványos sorozattárolóra alkalmazható, ami indexelhető (*operator[]*). Egy N elemű *PhyTomb* elemei pozitív és negatív indexértékkel is elérhetők. Míg a pozitív indexértékek a szokásos elérést eredményezik, addig a negatív indexek a tömb végétől haladnak visszafelé. Azaz a -1 az utolsó elemet adja, a -N pedig az első elemet. Úgy alakítsa ki a sablont, hogy alapértelmezésként N = 3, a sorozattároló pedig az *std::vector* legyen! A sorozattároló minden tagfüggvénye legyen elérhető, kivéve az *at()*. Ügyeljen a sorozattárolókra jellemző konstruktorok megvalósítására is! (5p)
Példa a használatra:

```
PhyTomb<int> t3;                // 3 elemű int tömb, nullával feltöltve
t3[0] = 1;                      // első eleme
std::cout << t3[-3];            // ez is az első, ezért 1-et ír ki!
t3[2] = 3;                      // utolsó eleme
std::cout << t3[-1];            // ez is az utolsó, ezért 3-at ír ki!
```

b) Hozzon létre az elkészített adapter és az *std::deque* felhasználásával egy 40 elemű long elemeket tartalmazó tömböt! (1p)

c) Írjon C++ függvénysablont (*count_if*), ami iterátorokkal megadott adatsorozatban megszámolja azokat az elemeket, amire a paraméterként átadott predikátum igaz értéket ad! A függvény első két paramétere két iterátor, amivel a szokásos módon megadjuk a jobbról nyílt intervallumot. A függvény 3. paramétere pedig egy predikátum, ami egy egyparaméteres függvény vagy függvényobjektum. Ha jól oldja meg a feladatot, akkor az alábbi kódrészlet lefutása után az eredmény 2. (2p)

```
bool negativ(int a) { return a < 0; }
int sorozat[] = { 1, 4, 9, -16, 25, 3, -72, 100, 3};      // a sorozat
int eredmeny = count_if (sorozat, sorozat+9, negativ);
```

d) Készítsen olyan függvényobjektum sablont, ami a *count_if* sablonnal felhasználva alkalmas az olyan elemek megszámolására, amelyek kisebbek a függvényobjektum konstruktorában megadott értéknél! (2p)

3. Feladat **10 pont**

A *Diak* osztályban diákok adatait tároljuk.

```
class Diak {
  std::string nev;
public:
  Diak(const std::string& n = "");
  std::string getNev() const;
  void setNev(const std::string&);
  virtual ~Diak();
};
```

Adott továbbá a *Serializable* osztály.

```
struct Serializable {
  virtual void write(std::ostream&) const = 0;
  virtual void read(std::istream&) = 0;
  virtual ~Serializable() {}
};
```

a) A fenti osztályok felhasználásával, de azok módosítása nélkül **hozzon** létre a *Diak* osztállyal kompatibilis, perzisztens *PDiak* osztályt! Megoldásában vegye figyelembe, hogy a névben szóköz is lehet! Az elmentett állapot visszatöltésekor az osztály végezzen ellenőrzést, hogy jó adatokat kap-e, azonban nem kell bombabiztos megoldás! Hiba esetén dobjon *std::out_of_range* kivételt! Működjön helyesen az alábbi kódrészlet: 5p

```
Diak d("Gaz Edoemer"); PDiak pd = d;
```

b) Egy rövid kódrészletben hozzon létre egy *PDiak* példányt a saját nevével! Mentse ki az objektum adatait a szabványos kimenetre! Kommentben adja meg, hogy mit írt ki! Jelölje a nem látható karaktereket is! 2p

c) Készítsen olyan fűzhető *inserter* (operator<<) és *extraktor* (operator>>) operátorokat, melyeket *Serializable* osztályból származó objektumpéldányokra alkalmazva aktivizálódik az osztály megfelelő *read* ill. *write* metódusa! 3p

Működjön helyesen az alábbi kódrészlet:

```
PDiak d1("Nagy Edoemer"), d2("Kiss Zaszlo");
std::stringstream ss;
ss << d1 << d2;
PDiak nd1, nd2;
ss >> nd1 >> nd2;    // elvárjuk, hogy d1 == nd1 && d2 == nd2,  azaz az elmentett állapot
//álljon elő az nd1 és nd2 objektumokban!
```

IMSC FELADAT Tételezze fel, hogy a 2. feladat *PhyTomb* osztálya, valamint a 3. feladat **c)** részfeladata is elkészült és hibátlanul működik! A *PhyTomb* osztály felhasználásával készítsen egy perzisztens viselkedésű *PPhyTomb* generikus osztályt, ami kompatibilis a *PhyTomb* osztállyal! Feltételezheti, hogy az osztályt pointerek tárolására nem használják. Tudjuk továbbá, hogy amennyiben a tárolt típus nem elemi, úgy az a *Serializable* osztályból származik (3. feladat). Az alábbi kódrészlettől elvárjuk, hogy deserializáláskor kivételt dob (a tárolt adattípusok eltérnek). 10 p

```
std::stringstream ss;
PPhyTomb<int> pi;
ss << pi;
PPhyTomb<double> pd;
ss >> pd;
```

4. Feladat **10 pont**

Egy gázszerelő cégnyilvántartását (*Portfolio*) szeretnénk modellezni. Nyilván kell tartani a vétel (*Vetel*) dátumát (*datum*, előre definiált *date* osztály), a kifizetett összeget (*osszeg*, double), valamint a megvett cég minden jellemzőjét (*ceg*: Ceg). A nyilvántartásból kiíratható az összes cégvásárlás adata, ami szűrhető egy adott napra is. A Portfolió a céghez közvetlenül nem férhet hozzá! Miután számtalan cég (*Ceg*) lehet, melynek a jellemzői előre nem ismertek, olyan megoldást kell választani, ami könnyen bővíthető tetszőleges céggel. Heterogén kollekció alkalmazása mellett döntöttünk. Ami biztos: minden cégnek van *neve* (string) és *adószáma* (string). Jelenleg az alábbi konkrét cégtípusokat kell támogatni:
Zrt: részvények száma (*db*, int), részvények névértéke (*ertek*, double)
KKV: alakulás éve (*ev*, int).

Tervezzen objektum-orientált megoldást a fenti leírás alapján a dőlt betűs megnevezések felhasználásával! Az attribútumok kivétel nélkül legyenek privátok és konstruktorban állíthatók. A megoldásban használja ki az STL adta lehetőségeket!
Az alábbi kódrészlet mutatja az egyes metódusok és konstruktorok paraméterezését és használatát.

```
Portfolio port; // egyik nyilvántartás
// 2018-02-03-án 23Mrd Ft-ért: Zabhegyező Zrt, adó 123456-1-13, 20 részvény, 2300Ft/részvény
port.megvesz(date(2018, 2, 3), 23e9, new Zrt("Lencselaposító Zrt", "123456-1-13", 20, 23000
// 2016-05-01-jén 1000 Ft-ért: Éliás Kft, adószám 234567-4-11, alapítva: 2013-ban.
port.megvesz(date(2016, 5, 1), 1000, new KKV("Éliás Kft", "234567-4-11", 2013);
port.listaz(std::cout);  // kilistázza az összes cégvásárlást
port.listaz(std::cout, date(2018, 2, 3));  // kilistázza az összes cégvásárlást 2018-02-03-án
```

Rajzoljon UML osztálydiagramot, amin **csak az osztályok neve** szerepeljen! (1p)

Definiálja az alábbi osztályokat! A metódusoknak csak a deklarációját adja meg!

class Portfolio { // (2p)

class Vetel { // (2p)

class Ceg { // (2p)

class KKV { // (1p)

Implementálja a *Portfolio* osztály dátum alapján listázó tagfüggvényét. Feltételezheti, hogy a *date* osztály összehasonlítható. (2p)