500.AA Megoldó Kulcsár 500.A

Programozás alapjai 2. (inf.) 2. ZH 2019.05.14. lab. hiányzás: +	/ HFt:
ABC123 IL.305./1	p: e:

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, 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.
A konstruktor előbb hajtja végre a programozott törzset, és csak ezután hívja az ősosztályok konstruktorát.
Az iterátor egy általánosított pointer.

b) Az alábbi függvénnyel egész számokat tartalmazó tárolóból szeretnénk eltávolítani elemeket. Teszteléskor észrevettük, hogy a függvény hívása után a tárolóban levő elemek száma nem változott. Mi lehet a baj? **Javítsa ki a függvényt** (2p)

```
void torol(std::list<int>& v, int elem) {


                std::remove(v.begin(), v.end(), elem);


}
```

c) Adott az alábbi deklaráció:

```
class Listam: public std::list<int> {
   int azon[10];
} t1, t2;
```

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)

`Listam t3 = t1;` utasítás helyes.
`t1 = t1 = t2;` utasítás hibás, mert az implicit operator= nem támogatja a többszörös értékadást.
`t1[4] = 56;` utasítás hibás, mert `t1` nem indexelhető.
`t1.push_front(45);` utasítás helyes.

d) Készítsen az *std::count_if* algoritmushoz hasonló **generikus algoritmust** (*count_if_not*), ami megszámolja, hogy az iterátorokkal megadott adatsorban hány olyan elem van, amire **NEM teljesül** a predikátum. Írjon rövid kódrészletet, melyben létrehoz egy sorozattárolót, mely egész elemeket tartalmaz, és a *count_if_not* sablon segítségével a standard kimenetre kiírja, hogy hány nullától különböző elem van a tárolóban! (4p)

500.AA Megoldó Kulcsár 500.A

2. Feladat **10 pont**

a) Készítsen adapter sablont (*Traversable*), ami minden szabványos sorozattárolóra alkalmazható, és van **traverse** tagfüggvénye. A *traverse()* függvény a tároló minden elemével hívja meg a paraméterként kapott egyoperandusú függvényt! Úgy alakítsa ki a sablont, hogy az alapértelmezett tárló az *std::vector* legyen! A sorozattároló minden publikus tagfüggvénye legyen elérhető! Ne felejtse el megvalósítani a sorozattárolókra jellemző konstruktorokat! (5p)

Példa a *Traversable* sablon használatra:

```
void func(long i) { std::cout << i << ","; } // kiírja a paraméterét és egy vesszőt
...
Traversable<long> tv(20, -3);                // létrehoz egy 20 elemű tárolót, -3-mal feltöltve
tv.traverse(func);                           // kiírja az elemeket
```

b) Hozzon létre az elkészített adapter és az *std::list* felhasználásával egy *int* típusú elemeket tartalmazó *üres* tárolót! (1p)

c) Írjon kódrészletet, amely a fenti tárolóba a szabványos inputról a fájl végéig számokat olvas be! (1p)

d) Készítsen olyan **funktort** (*kiirNagyobb*), amely alkalmazható a *traverse()* függvény paramétereként és segítségével kiírhatók a szabványos kimenetre a **c)** részfeladatban beolvasott adatok közül azok, amelyek egy referenciaértéknél nagyobbak! A számokat vesszővel válassza el egymástól. A referenciaértéket a funktor konstruktorában lehessen megadni! **Mutassa** be a funktor használatát egy rövid kódrészlettel, melyben vesszővel elválasztva írja ki a nullától nagyobb értékeket a tárolóból! (3p)

500.AA Megoldó Kulcsár 500.A

IMSC FELADAT

Az *std::copy_if(InputIterator first, InputIterator last, OoutputIterator result, UnaryPredicate pred)* algoritmus átmásolja az iterátorokkal megadott adatsor azon elemeit, amelyek eleget tesznek a predikátumnak.

e) Készítsen olyan **generikus** funktort, amelyet a *traverse()* függvény operandusaként használva a *copy_if* funkcionalitása kiváltható!

f) A **c)** részfeladatban feltöltött tárolóból a *traverse()* függvényt, használva **rövid kódrészlettel másolja** át a 100-nál kisebb elemeket egy tárolóba, a 100-nál nagyobbakat pedig egy másik tárolóba! Csak STL elemeket használjon, kivéve az **e)** részfeladatban készített funktort! Ügyeljen arra, hogy a céltárolóban legyen elegendő hely! A másolás után pedig ne maradjon szemét!

3. Feladat **10 pont**

Számítógép hálózat felett elosztott játékot készítünk. A játéknak vannak migrálható objektumai, melyek az egyik gépről a másikra át tudnak menni a *migrateTo(*)és a *migrateFrom()* metódusok segítségével. Ezeket a metódusokat a játékot működtető keretrendszer hívja meg a megfelelő szöveges stream paraméterekkel. A tagfüggvények feladata, hogy az objektum állapotát kiírják a stream-re (migrateTo) ill. beolvassák azt arról (migrateFrom). Az alábbiakban megadjuk a *Mouse* és a *Migratable* osztályok definícióját:

```
class Mouse {
  std::string name;                    // egér neve
  unsigned age;                        // egér kora
public:
  Mouse(const std::string& n = "", unsigned a = 0);
  std::string getName() const;
  unsigned getAge() const;
  void setName(const std::string&);
  void setAge(unsigned);
  virtual ~Mouse();
};
struct Migratable {
  virtual void migrateTo(std::ostream&) const = 0;
  virtual void migrateFrom(std::istream&) = 0;
  virtual ~Migratable() {}
};
```

a) A fenti osztályok felhasználásával, de azok módosítása nélkül **hozzon** létre a *Mouse* osztállyal kompatibilis, migralható *MigratableMouse* osztályt! Megoldásában vegye figyelembe, hogy a névben szóköz is lehet! A stream-re történő kiíráskor írjon ki egy azonosítót is (pl: 'M'), ami beolvasáskor alkalmas annak ellenőrzésre, hogy jó objektumból származik-e a beolvasott adat. Nem kell bombabiztos megoldás! Hiba esetén dobjon *std::domain_error* kivételt! 5p

b) Egy rövid kódrészletben hozzon létre egy *MigratableMouse* példányt „Mickey" névvel, majd egy *std::stringstream* típusú objektum segítségével migrálja azt át egy másik „gépre" (hozzon létre egy újabb egeret, és oda migrálja)! 3p

c) Tételezze fel, hogy létezik egy *MigratableCat* osztály is, ami képes kiírni saját állapotát egy adatfolyamra. Olvassa be ezt egy *MigratableMouse* példányba! A keletkező kivételt kezelje le! 2p

4. Feladat **10 pont**

Tudományos akadémiák kutatóhálózatát (*Network*) szeretnénk modellezni. A kutatóhálózatban különféle szervezetek (*Org*) vesznek részt. Mindnek van neve (*name, string*) és alapítási éve (*year, int*), ezek getter metódussal lekérhetők. A szervezetek közé tartoznak az Intézetek (*Institute*), a laborok (*Laboratory*) stb. Ezek újabb típusokkal bővülhetnek. Az intézeteknek van vezetője (*head, string*), a laboroknak van értéke (*value, double*). A *print* metódus meghívásakor a szervezetek kiírják adataikat a paraméterként megkapott *ostream*-re. Az intézeteknek lehet saját laborja is (a *setLab* metódussal rendelhető hozzá), ilyenkor a *print* rekurzívan a saját laboron is meghívódik.

A kutatóhálózatba felvehetők újabb szervezetek (*add*), de meglevő szervezeteket törölni is lehet a referenciájuk megadásával (*remove*). Ha a kutatóhálózat megszűnik, a benne levő szervezetek is megszűnnek. A *listAll* metódussal a kutatóhálózat minden szervezete kiíródik a szabványos kimenetre. Ha a metódusnak van egy *int* paramétere, akkor az adott évszám után alapított szervezetek listázódnak csak.

Tervezzen objektum-orientált megoldást a fenti leírás alapján a dőlt betűs megnevezések felhasználásával! Az alábbi kódrészlet mutatja az egyes metódusok és konstruktorok paraméterezését és használatát.

```
Network mta; // egyik kutatóhálózat
Laboratory* l1 = new Laboratory("Bölcsészlabor", 1876, 123456.789);
Laboratory* l2 = new Laboratory("Sugárlabor", 1978, 98765432.1);
Institute* i1 = new Institute("Innovatív intézet ", 2001, "Dr. Eher Kevin");
i1->setLab(l2);
mta.add(l1);
mta.add(i1);
mta.listAll(1920);
```

Definiálja az osztályokat! Az attribútumok kivétel nélkül legyenek privátok és konstruktorban állíthatók. Elegendő a fenti példa működéséhez szükséges getterek/setterek deklarációja. A megoldásban használja ki az STL adta lehetőségeket!

Egy lehetséges megoldás:

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

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

```
class Network                    { // (2p)
```

```
class Org                        { // (2p)
```

```
class Institute                  { // (2p)
```

```
class Laboratory                 {// (1p)
```

Implementálja a *Network* osztály évszám alapján listázó tagfüggvényét *iterátor* használatával. (2p)