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, 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.

f.	max.	elért	jav.
1.	10		
2.	10		
3.	10		
4.	10		
Σ	**40**		
IM.	10		
Van megoldott IMSC:			

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.	I	**H**
Statikus tagfüggvénynek nincs this pointere.	**I**	H
A konstruktor előbb hajtja végre a programozott törzset, és csak ezután hívja az ősosztályok konstruktorát.	I	**H**
Az iterátor egy általánosított pointer.	**I**	H

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::list<int>::iterator it;

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

.           v.erase(it, v.end());
}
```

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

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)

Egy lehetséges megoldás:

```
template <typename Iter, typename Pred>
size_t count_if_not(Iter first, Iter last, Pred pred) {
    size_t cnt = 0;
    while (first != last)
        if (!pred(*first++)) cnt++;
    return cnt;
}
std::list<int> l(8, 8);
std::cout << count_if_not(l.begin(), l.end(), std::bind2nd(std::equal_to<int>(), 0));
```

500.AA Megoldó Kulcsár 500.A

2. Feladat **Σ 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, ami a fenti tárolóba a szabványos inputról a fájl végéig számokat olvas be! (1p)

d) Készítsen olyan **funktort (*kiirNagyobb*),** ami 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)

Egy lehetséges megoldás:

a)

```
template <typename T, typename C = std::vector<T> >
struct Traversable : public C {//   Delegáció nem jó, mert nem tudjuk, milyen fv-ei vannak a tárolónak
    Traversable(size_t n = 0, const T& value = T()) : C(n, value) {}

    template<typename Iter>//
    Traversable(Iter first, Iter last) : C(first, last) {}

    template<typename F> //t
    F traverse(F func) {
        typename C::iterator first = C::begin();
        typename C::iterator last = C::end();
        while (first != last)
            func(*first++);
    return func;
    }
};
```

b)

```
Traversable<long, std::list<lint> > tar;
```

c)

```
    int adat;
    while (cin >> adat)
        tar.push_back(adat);
```

d)

```
struct kiirNagyobb {
    int ettol;
    kiirNagyobb(int ettol) :ettol(ettol) {}
    void operator()(int adat) const {
        if (adat > ettol) std::cout << adat << ",";
    }
};

tar.traverse(kiirNagyobb(0));
```

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!

Egy lehetséges megoldás:

e)

```
template <typename O, typename F>
struct copyf {
    O outIter;
    F func;
    copyf(O outp, F func) :outIter(outIter), func(func) {}
    template <typename T>
    void operator()(T adat) {
        if (func(adat)) *outIter++ = adat;
    }
};

// Ez a sablon nem szükséges a megoldáshoz, de segítségével a sablonparamétereket a fordító le tudja vezetni,
// így nem kell azokat kiírni.
template <typename O, typename F>
copyf<O, F> copy(O o, F f) {
    return copyf<O, F>(o, f);
}
```

f)

```
Traversable<int> tar2(tar.size());
Traversable<int>::iterator it =
tar.traverse(copy(tar2.begin(), std::bind2nd(std::less<long>(), 10))).outIter;

tar2.erase(it, tar2.end());
tar2.traverse(func);
```

3. Feladat **Σ 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

Egy lehetséges megoldás:

```
struct MigratableMouse : public Mouse, public Migratable {}
    MigratableMouse(const std::string& n = "", unsigned a = 0) : Mouse(n, a) {}
    MigratableMouse(const Mouse& d) : Mouse(d) {}
    void migrateTo(std::ostream& os) const {
        os << "Mouse" << std::endl;
        os << getName() << std::endl;
        os << getAge() << std::endl;
    }
    void migrateFrom(std::istream& is) {
        std::string line;
        getline(is, line);
        if (line != "Mouse") throw std::domain_error("Mouse != " + line);
        getline(is, line);
        setName(line);
        unsigned int a;
        (is >> a).ignore(1);
        setAge(a);
    }
};
b)
        MigratableMouse mm1("MIckey"), mm2;
        std::stringstream ss;
        mm1.migrateTo(ss);
        mm2.migrateFrom(ss);
c)
     try {
        MigratableCat cat("figaro");
        cat.migrateTo(ss);
        mm2.migrateFrom(ss);
    } catch (std::exception& e) {
        cout << e.what();
    }
```

4. Feladat **Σ 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.

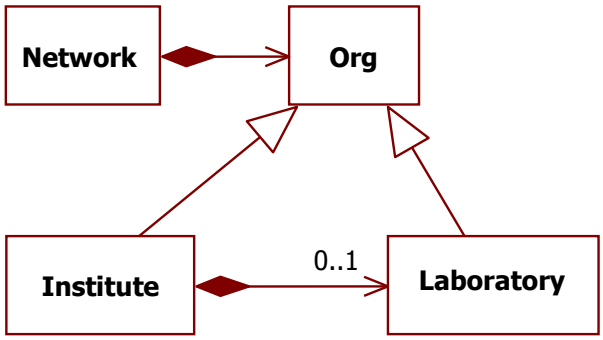
Tervezzen objektum-orientált megoldást a fenti leírás alapján a dőlt betűs megnevezések felhasználásával! Az alábbi kódrészlet mutatja az egyes metódusok és konstruktorok paraméterezését és használatát.

```
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)
  vector <Org*> v;
public:
  void add(Org* o);
  void remove(Org&);
  void listAll(int year = 0) const;
  ~Network() ;
};
```

```
class Org                  { // (2p)
  string name;
  int year;
public:
  Organization(string n, int y)
  virtual void print(ostream& os) const;
  virtual ~Organization();
  int getYear();
  string getName();
};
```

```
class Institute : public Org       { // (2p)
  string head;
  Org* lab;
public:
  Instutite(string n, int y, string h);
  void print(ostream& os) const;
  setLab(Laboratory* l);
  ~Institute();
};
```

```
class Laboratory : public Org  {// (1p)
int ev;
double value;
public:
  Laboratory(string n, int y, double v);
  void print(ostream& os) const;
};
```

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

```
void Network::listAll(int year) {
  for (vector<Org*>::iterator it = v.begin(); it != v.end(); it++) {
    if ((*it)->getYear() > year) {
      (*it)->print(std::out); std::out << std::endl;
    }
  }
```