500.AJ Megoldó Magyar Magdolna 500.J

Programozás alapjai 2. (inf.) 2. pZH 2017.05.09. gy./l. hiány:	/
ABCDEF **IB.028/1.**	Z: / /

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

*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 nem használható. A feladatokat **figyelmesen olvassa** el, megoldásukhoz **ne használjon fel STL** tárolót, kivéve, ha a feladat ezt külön engedi/kéri! **Ne írjon felesleges** függvényeket ill. kódot!*

Az első feladatrészben 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**		
Van IMSC megoldásom:			
IM	10		

1. feladat: Beugró. A feladatok megoldásához **használhat STL tárolót és algoritmust!** **Σ 10 pont**

a) Mit ír ki az alábbi program a szabványos kimenetre? Válaszát soronként a vonalazott részre írja! (3p)

```
#include <iostream>
using std::cout;
using std::endl;

inline void e() { cout << endl; }

struct J {
   int i;
   J()          { cout << 'k'; }
   J(const J&)  { cout << 'c'; }
   ~J()         { cout << 'd'; }
   void f(J& )  { cout << 'f'; }
   J& operator=(J a){
        i = a.i; cout << '=';
                 return *this; }
};

int main() {
   J *a= new J; e();//  k  _________
   J b = *a;    e();//  c  _________
   *a = b;      e();//  c=d  _______
   a->f(b);     e();//  f  _________
   delete a;    e();//  d  _________
}                  //  d  _________
```

Soronként 0.5 p.

b) Készítsen generikus függvényt (mySwap), ami az std::swap függvényhez hasonlóan felcseréli a paraméterként kapott két generikus adatot! (2p)

```
//Egy lehetséges megoldás:
template <typename T>
void myswap(T& a, T& b) {
     T tmp = a;
     a = b;
     b = tmp
}
```

c) Írjon **programrészletet**, ami a szabványos bemenetről fájl végéig egész számokat olvas be, majd fordított sorrendben kiírja azokat a standard kimenetre, végül rendezve is írja ki a beolvasott számokat! A megoldáshoz használjon STL eszközöket! (3p)

```
//Egy lehetséges megoldás:

std::vector<int> tar;
int a;
while (std::cin >> a)
   tar.push_back(a);

for (size_t i =tar.size(); i > 0;)
   std::cout << tar[--i] << " ";

std::sort(tar.begin(), tar.end());
for (size_t i = 0; i < tar.size(); ++i)
   std::cout << tar[i] << " ";
```

d) 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 iterátor egy általánosított pointer	**I**	H
A kivételosztályokat kötelező az *std::exception* osztályból származtatni.	I	**H**
Alaposztály destruktorából hívott virtuális tagfüggvény a leszármazottban fut le.	I	**H**
Függvénysablon példányosítása futási időben történik	I	**H**

2. Feladat **Σ 10 pont**

a) Készítsen adapter sablont (*PyTomb*), ami minden olyan szabványos sorozattárolóra alkalmazható, melynek van *at()* tagfüggvénye. Egy N elemű *PyTomb* elemei pozitív és negatív indexértékkel is elérhetők. Míg a pozitív indexértékek a szokásos elérést eredményezik, addig a negatív indexek a tömb végétől haladnak visszafelé. A -1 az utolsó elemet adja, a -N pedig az első elemet. Úgy alakítsa ki a sablont, hogy alapértelmezésként a tároló az *std::vector* legyen! Ügyeljen arra, hogy a sorozattárolókra jellemző konstruktorok elérhetőek legyenek! A sorozattároló minden tagfüggvénye legyen elérhető! Példa a használatra: (4p)

```
PyTomb<int> t(3);              // int tömb 3 elemmel
t[0] = 1;                      // első elem (=1)
std::cout << t[-3];            // ez is az első (=1)
t[2] = 3;                      // utolsó eleme (=3)
std::cout << t[-1];            // utolsó (3.) elem (=3)
t.push_back(10);               // most már 4 elemű!
std::cout << t[-1];            // utolsó (4.) elem (=10)
```

```
// Egy lehetséges megoldás:
template <typename T, class C = std::vector<T> >
class PyTomb : public C {
public:
    PyTomb(size_t n = 0, const T& value = T()) : C(n, value) {}
    template<typename Iter>
    PyTomb(Iter first, Iter last) : C(first, last) {}p
    T& operator[](int ix) { if (ix < 0) ix += C::size(); return C::at(ix); }
    T operator[](int ix) const { if (ix < 0) ix += C::size(); return C::at(ix); }
};
```

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

```
PyTomb<long, std::deque<long> > t40(40);
```

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);
```

```
// Egy lehetséges megoldás:
template<typename I, typename P>
int count_if(I first, I last, P pred) {
    int cnt = 0;
    while (first != last)
       if (pred(*first++)) cnt++;
    return first;
}
```

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

```
// Egy lehetséges megoldás:
template <class T>
class KisebbMint {
    T ref;
public:
    KisebbMint(const T& a) : ref(a) {}
    bool operator()(const T& a) const { return a < ref; }
};
```

e) A részfeladatok eredményeit felhasználva írjon kódrészletet, ami a **b)** részfeladatban létrehozott tömbben megszámolja a negatív elemeket! (1p)

```
std::cout << count_if(t40.begin(), t40.end(), KisebbMint<long>(0));
```

IMSC FELADAT:

Egészítse ki a PyTomb sablont unique iterátorral! Az iterátort *ubegin( )* és *uend( )* metódusokkal lehessen lekérni. A unique iterátor ugyanúgy működik, mint a hagyományos iterátor, de a szomszédos ismétlődő elemeket átugorja. Pl ha a tároló tartalma (2,3,4,4,5,5,5,2), akkor a unique iterátor a 2,3,4,5,2 értékeket adja sorban vissza.

```
// Egy lehetséges megoldás:
// PyTomb sablon kiegészítése: (a sablonba ez kerül):

    typedef typename C::iterator iterator;
    class unique_iterator;
    unique_iterator ubegin() { return unique_iterator(C::begin(), C::end()); }
    unique_iterator uend() { return unique_iterator(C::end(), C::end()); }

    class unique_iterator : public C::iterator {
        iterator e;
    public:
        unique_iterator() {}
        unique_iterator(const iterator& i, const iterator& e)
            : C::iterator(i), e(e) {}

        unique_iterator& operator++() {
            iterator first = *this;
            iterator tmp = *this;
            while (++tmp != e && *tmp == *first);
            *this = unique_iterator(tmp, e);
            return *this;
        }

        unique_iterator operator++(int) {
            unique_iterator tmp = *this;
            ++(*this);
            return tmp;
        }
    }; // iterator class vége
}; // PyTomb template vége
```

---

3. Feladat **Σ 10 pont**

A *Film* osztályban házimozinkhoz tárolunk adatokat.

```
class Film {
    std::string cim;
    int polc;
public:
    Film(const std::string& n = "", int p = 0);
    int getPolc() const;
    std::string getCim() const;
    void setPolc(int);
    void setCim(const std::string&);
    virtual ~Film();
};
```

Adott továbbá a *Serializable* osztály.

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

a) A fenti osztályok felhasználásával, de azok módosítása nélkül hozzon létre a *Film* osztállyal kompatibilis, perzisztens *PFilm* osztályt! Megoldásában vegye figyelembe, hogy a címben 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! 5p

```
// Egy lehetséges megoldás:
struct PFilm : public Film, public Serializable {
    PFilm(const std::string& n = "", int p = 0) : Film(n, p) {}
    void write(std::ostream& os) const {
        os << "PFilm" << endl;
        os << getCim() << endl;
        os << getPolc() << endl;
    }
    void read(std::istream& is) {
        std::string line;
        getline(is, line);
        if (line != "PFilm")
              throw std::out_of_range("Film szakadas");
        getline(is, line);
        setCim(line);
        int a;
        (is >> a).ignore(1);
        setPolc(a);
    }
```

};**b**) Egy rövid kódrészletben hozzon létre egy *PFilm* példányt a kedvenc filmcímével. Mentse ki az objektum adatait a szabványos kimenetre! Kommentben adja meg, hogy mit írt ki! Jelölje a nem látható karaktereket is! 2p

```
PDiak d1("Nagy Elemer", 5.0);
d1.write(std::cout); // PFilm\nNagy Elemer\n5.0\n
```

c) Tételezze fel, hogy rendelkezésére áll a gyakorlaton elkészített *PKomplex* osztály is, ami szintén a Serializable osztály segítségével valósítja meg a perzisztens viselkedést! Egészítse ki megoldását, hogy az alábbi kódrészlet az elvárásoknak megfelelően működjön! 3p

```
        PFilm f1("Vak asszony visszanez", 1), f2("A destruktor bosszuja", 2);
        PKomplex k1(2, 4);
        stringstream ss;
        f1.write(ss);
        k1.write(ss);
        f2.write(ss);
        PFilm nf1, nf2;
        PKomplex nk1;
        ss >> nf1 >> nk1 >> nf2;      // elvárjuk, hogy f1 == nf1 && f2 == nf2 && k1 == nk1 legyen
                                      // azaz az elmentett állapot álljon elő.

std::istream& operator>>(std::istream& is, Serializable& s) {
        s.read(is);
        return is;
}
```

4. Feladat **Σ 10 pont**

Az Állampolgárokat Védő Hivatal (AVH) nyilvántartást (*Nyilvantartas*) készít a különböző szervezetekről (*Organ*). Minden szervezetnek van neve (*name*, string), és lekérhető, hogy mennyi pénzt kapott a gyíkemberektől (*getAmount()*). Több fajta szervezetünk lehet, és ezek száma később nőni fog. A *Friend* típusú szervezet például mindig 0-t ad vissza, ha gyíkember-pénzről kérdezzük, de van kedvenc focicsapata (*team*, string). A *Civil* típusú szervezet egy külön attribútumban tartja nyilván, hogy melyik évben ismerkedett meg a fő gyíkemberrel (*soros*, int). Vannak gyűjtőszervezetek, amelyek több másik szervezetből állnak (*Liga*), ezeknél a *getAmount()* mindig a bennük levő szervezetek *getAmount()*-jának összege. Egy Ligába a *join* metódussal lehet új szervezet felvenni. Minden szervezet ki tudja írni a megadott ostreamre az összes adatát (*report*). A Liga típusúak rekurzívan a tagszervezetekét is.

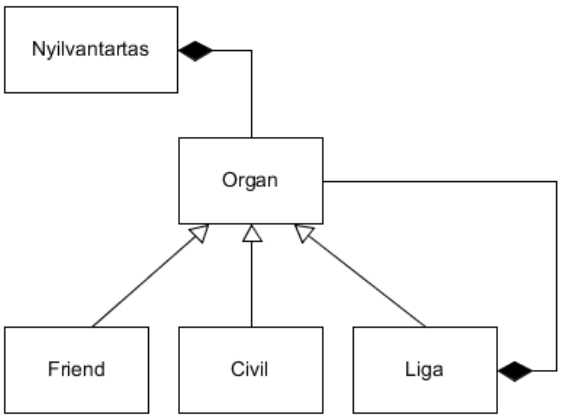
A nyilvántartásba fel tudunk venni új szervezetet (*add*), és ki is tudjuk listázni a meglevőket (*list*), aminek a során az egyes szervezetek egyedi kiíratása fut le. Ha egy nyilvántartás megsemmisül, a tárolt adatok is elvesznek. Ugyanez igaz a Liga típusú szervezetekre is.

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.

```
Nyilvantartas ny3per1;
ny3per1.add(new Civil("Lofarok", 1989));
ny3per1.add(new Friend("Szarnyalas", "Fertoapati"));
Liga* fuveszek = new Liga();
fuveszek->join(new Civil("Spangli420", 2008));
fuveszek->join(new Civil("BobMarley", 1981));
ny3per1.add(fuveszek);
ny3per1.list(std::cout);
```

Egy lehetséges megoldás:

Rajzoljon UML osztálydiagramot, amin ne szerepeljenek az attribútumok és a metódusok sem.



Definiálja az alábbi osztályokat, de a metódusoknak csak a deklarációját adja meg!

```
class Nyilvantartas {
    vector<Organ*> orgs;
public:
    void add(Organ*);
    void list(ostream&) const;
    ~Nyilvantartas();
};
```

```
class Organ {
    string name;
    double amount;
public:
    Organ(string);
    virtual double getAmount() const;
    virtual void report(ostream&) const;
    virtual ~Organ();
};
```

```
class Liga : public Organ {
    vector<Organ*> orgs;
public:
    Liga(string n = "Noname");
    double getAmount() const;
    void report(ostream&) const;
    void join(Organ*);
    ~Liga();
};
```

```
class Civil : public Organ {
    int soros;
public:
    Civil(string, int);
    double getAmount() const;
    void report(ostream&) const;
};
```

Implementálja külső definícióval a *Liga* összes metódusát és konstruktorát, ami a példakód futtatásához szükséges!

```
Liga::Liga(string n) : Organ(n) {}

void Liga::join(Organ *o) {
    orgs.push_back(o);
}

void Liga::report(ostream& os) const {
    Organ::report(os);
    os << endl;
    for (size_t i = 0; i <orgs.size(); i++) {
        orgs[i]->report(os);
        os << endl;
    }
}

Liga::~Liga() {
    for (size_t i =0; i <orgs.size(); i++)
        delete orgs[i];
}
```