4. DD Megoldó Alfréd (A) DD 4.

Programozás alapjai 2. (inf.) zárthelyi 2012.05.10. gyak. hiányzás:	kZHpont:
ABCDEF **MEG/1.**	Hftest:

F.	Max.	Elért
1	3	
2	4	
3	4	
4	4	
5	5	
Σ	**20**	

Minden beadandó megoldását a feladatlapra, a feladat után írja! *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. kalkulátor, számítógép, notebook, mobiltelefon, Kindle) 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!*

A feladatra koncentráljon! Jó munkát! *Értékelés: 0-8:1, 9-11:2, 12-14:3, 15-17:4, 18-20:5*

1. Feladat 6*0.5=3 pont

Mit ír ki a szabványos kimenetre az alábbi program? Válaszához használja a négyzetrácsos területet!
Ha olyan sorba ír, ahova nem történik kiírás, azért pontot vonunk le!

```
#include <iostream>
using namespace std;
struct Alap {
  Alap(int i = 1)                    { cout << i << "k"; }
  virtual void f(int i)              { cout << "f"; }
  void operator=(const Alap& a)      { cout << "=" << "f"; }
  virtual ~Alap()                    { cout << "d"; }
};
struct Uj :public Alap {
  Uj(int a, const char *m) :Alap(a)  { cout << m << "K"; }
  Uj(const Uj& u) :Alap(u)           { cout << "C" << "f";; }
  void f(int i)                      { cout << "F" << i; }
  ~Uj()                              { cout << "D"; }
};
void f(Uj u) {
  u.f(2);
}

int main() {
  Uj u12(1, "cde");                  cout << endl;
  Alap *pa = &u12;                   cout << endl;
  Uj u2 = u12;                       cout << endl;
  u12 = u2;                          cout << endl;
  pa->f(1);                          cout << endl;
  f(u12);                            cout << endl;
  Alap a1 = *pa;                     cout << endl;
}
```

1	k	c	d	e	K						
C	f										
=	f										
F	1										
C	f	F	2	D	d						
d	D	d	D	d							

2. Feladat Σ 4 pont

Írjon függvénysablont (***talal_ha***), ami az STL *find_if* sablonjához hasonlóan kikeresi egy tárolóból az első olyan elemet, amelyik a harmadik paraméterben átadott predikátumnak eleget tesz! A függvény első két paramétere két iterátor, ami kijelöli a jobbról nyílt intervallum kezdetét és végét. A függvény visszatérési értéke egy olyan iterátor, ami a megtalált elemre vagy az intervallum végére mutat. Amennyiben helyesen oldja meg a feladatot, akkor az alábbi kódrészlet a következőt írja ki: **17**

```
    int v2[] = { 2, 2, 3, 17, 18, 6, 0};
    cout << *talal_ha(v2, v2+6, bind2nd(greater<int>(), 5)) << endl;
```

Készítsen egy olyan függvényobjektumot, ami alkalmazható lenne a fenti példában az első páratlan szám megtalálására!
Mutassa be, hogy az elkészített függvénysablonnal és függvényobjektummal hogyan tudná az első páratlan számot megtalálni egy egészeket tartalmazó std::vector-ban! Egy lehetséges megoldás:

```
template<typename Iter, class Pred>
Iter talal_ha(Iter first, Iter last, Pred pred) {
    while (first != last) {
        if (pred(*first))
            break;
        ++first;
    }
    return first;
}
```

A feladat második részéhez nem kellett template osztályt készíteni, de azt is elfogadtuk. Ezzel szemben csak függvény objektumot fogadtunk el, ahogy a feladat kérte, függvényt nem.

```
struct Odd {
    bool operator()(int i) {   // függvényhívás operátorát használjuk
        return (i & 1) == 0;
    }
};
```

Feladat harmadik része:

```
vector<int> bemutat;
vector<int>::iterator it = talal_ha(bemutat.begin(), bemutat.end(), Odd());
if (it == bemutat.end()) cout << "Nincs" << endl;
else cout << *it << endl;
```

---

3. Feladat Σ 4 pont

Egy korlátos méretű generikus adatsor *(Sor)* osztályt kell készíteni. A tároló maximális méretét az osztály egyparaméteres konstruktora kapja meg. A default konstruktor állítsa a méretet 160-ra! Az osztály rendelkezzen a következő műveletekkel:

- *front()* – a sor első elemének referenciáját adja; dobjon *std::out_of_range* kivételt, ha üres a sor;
- *back()* – a sor utolsó elemének referenciáját adja; dobjon *std::out_of_range* kivételt, ha üres a sor;
- *push()* – elemet tesz a sor végére; amennyiben a tárolt elemek száma meghaladná a tároló méretét, úgy a sor elejéről dobjon el egy adatot;
- *pop()* – eldob egy elemet a sor elejéről; dobjon *std::out_of_range* kivételt, ha üres a sor;
- *size()* – a tárolóban levő elemek számát adja;
- *maxsize()* – a tároló maximális méretét adja.

Az osztályból példányosított objektum:

- legyen átadható érték szerint függvényparaméterként;
- kezelje helyesen a többszörös értékadást (*q1=q2=q3*);
- a front és a back tagfüggvényeknek nem kell konstans objektumra működni!

Deklarálja a *Sor* osztályt! **Valósítsa** meg az osztály default konstruktorát, valamint a *front()* és a *push()* metódusokat!

Egy lehetséges megoldás:

```
template<class T>
class Sor {
    T *tar;              // pointer az adatra
    const int msiz;      //maximális méret
    int siz;             // aktuális méret
    int first;           //első adat helye
public:
    Sor(int msiz = 160);
    Sor(const Sor&);
    Sor& operator=(const Sor&);
    T& front();
    T& back();
    void push(const T&);
    void pop();
    int size() const;
    int maxsize() const;
    ~Sor();
};
// A tagfüggvények megvalósítása lehetne az osztály deklarációjával együtt, de külön írva a megoldás menetének jobban megfelel.
template<class T>
Sor<T>::Sor(int msiz = 160): msiz(msiz), siz(0), first(0) {
    tar = new T[msiz];   // tárterület lefoglalása
}
template<class T>
T& Sor<T>::front() {
    if (siz == 0) throw std::out_of_range("Sor::front: empty!");
    return tar[first];
}
template<class T>
void Sor<T>::push(const T& e) {
    if (siz == msiz) pop();
    tar[(first+siz) % msiz] = e;
    siz++;
}
```

// További csoportok megoldásai (back, pop, másoló)

```
template<class T>
T& Sor<T>::back() {
    if (siz == 0) throw std::out_of_range("Sor::back: empty!");
    return tar[(first+siz-1) % msiz];
}
template<class T>
void Sor<T>::pop() {
    if (siz == 0) throw std::out_of_range("Sor::pop: empty!");
    first = (first+1) % msiz;
    siz--;
}
template<class T>
Sor<T>::Sor(const Sor& s2) {
    t = NULL;
    *this = s2;
}
```

---

4. Feladat Σ 4 pont

Tételezze fel, hogy a **3. feladat** *Sor* osztálya elkészült, és hibátlanul működik! Ezen osztály **felhasználásával hozzon létre** egy *SzamSor* osztályt maximum 120 db int típusú elem tárolására! Az osztály **mindenben** működjön úgy, ahogy a *Sor* osztály, kivéve a *push* metódus, ami ne engedjen adatot veszni: ha megtelik a tároló, akkor dobjon *std::range_error* kivételt!

A *SzamSor* osztály **felhasználásával írjon** egy olyan **kódrészletet**, ami a standard inputról fájl végéig számokat olvas be, majd a beolvasás sorrendjében kiírja az átlagnál nagyobbakat! A programrészlet kezelje az esetleg keletkező kivételeket is!

A feladatot delegációval vagy örökléssel lehetett megoldani. Delegáció esetén többet kellet írni, mivel a Sor minden tagfüggvényét delegálni kellett. Egy lehetséges megoldás örökléssel:

```
class SzamSor : public Sor<int> {
public:
    SzamSor() :Sor<int>(120) {}
    void push(int i) {
        if (size() >= maxsize()) throw std::range_error("Szamsor::push");
        Sor<int>::push(i);
    }
};
```

Kódrészlet:

```
try {
    SzamSor szsor;
    int x;
    int sum = 0;
    while (cin >> x) {
        szsor.push(x);
        sum += x;
    }
    double atl = (double)sum / szsor.size();
    while (szsor.size()) {
        x = szsor.front();
        szsor.pop();
        if (x > atl)
            cout << x << endl;
    }
} catch (exceptions& e) {
    cerr << "Kivetel:" << e.what() << endl;
}
```

4. DD Megoldó Alfréd (A) DD 4.

5. Feladat Σ 5 pont

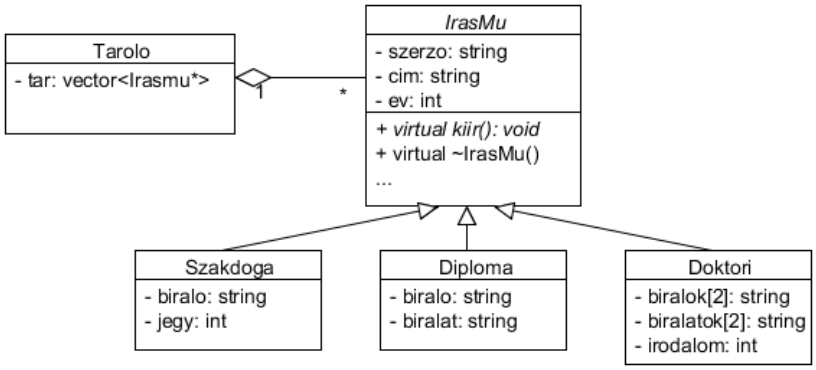
A BME irattárában szeretnénk a szakdolgozatok (*Szakdoga*), diplomatervek (*Diploma*), doktori disszertációk (*Doktori*) tárolását elektronikus rendszerrel segíteni. A rendszerben együtt akarják tárolni (*Tarolo*) ezeket az írásműveket. Minden ilyen írásműnek (*IrasMu*) van szerzője, beadási éve és címe. Ezen felül a szakdolgozatok mellett tárolni kell a bíráló nevét és a javasolt osztályzatot is. A diplomatervek esetében még a bírálat szövegét is tárolni kell. A doktori disszertációk esetében pedig két bíráló és két bírálati szöveg van, valamint tárolni kell, hogy hány db irodalmi hivatkozás van a dolgozatban.
A rendszerben a következő műveleteket szeretnénk megvalósítani:

- új írásmű felvétele;
- keresés adott cím szerint;
- a keresés eredményeként megtalált mű adatainak kiíratása;
- az összes mű címének kiíratása ábécé sorrendben;

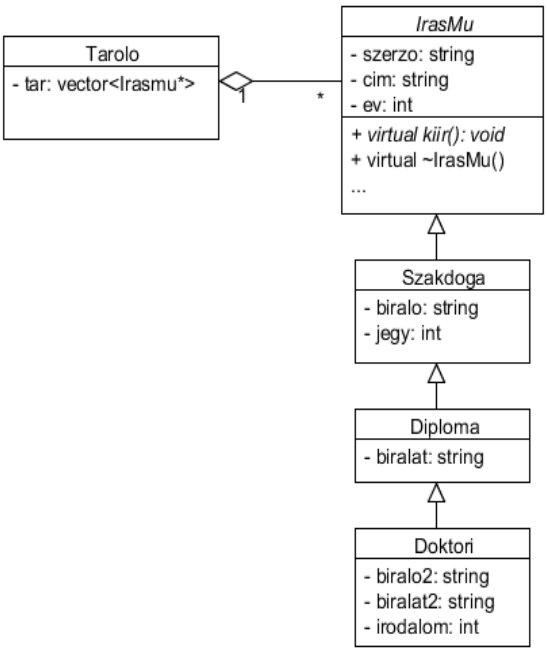
Feladatok:

- **STL** eszközök **felhasználásával tervezzen** olyan OO modellt, ami könnyen bővíthető további művekkel. Rajzolja fel a modell osztálydiagramját! Nem fontos tagfüggvényeket feltüntetni az UML ábrán. Használja a dőlt betűs neveket!
- **Legyen** a *Tarolo* osztálynak iterátora, amivel bejárhatók és elérhetők a tárolt írásművek!
- **Deklarálja** az osztályokat!
- **Implementálja**
  - *a Szakdoga* osztály konstruktorát;
  - a címek szerinti kereséséhez és ábécé sorrendben történő kiírásához szükséges metódusokat!
- **Írjon** egy olyan kódrészletet, ami *Tarolo* iterátorát felhasználva azonos című műveket keres, és kiírja azok adatait!

A heterogén kollekció elemeinek helyes használatát értékeltük. Az öröklési hierarchia több módon is kialakítható volt, (szétágazó, láncoló, többszörös) ezeket mind elfogadtuk. Heterogén kollekció esetén **súlyos hiba** a tárolóból örökölni. Ezt szigorúan büntetettük. Egy lehetséges megoldás:
Szétágazó:



Láncoló:



A láncoló hierarchia megvalósítása:

```
class IrasMu {
    string szerzo;
    string cim;
    int ev;
public:
    IrasMu(string szerz, string cim, int ev);
    string getSzerzo() const { return szerzo; }
    string getCim() const { return cim; }
    virtual void kiir() const = 0;        // virtuális kell
    virtual ~IrasMu();                    // virtuális kell
};
class Szakdoga: public IrasMu {
    string biralo;
    int jegy;
public:
    Szakdoga(string szerz, string cim, int ev, string bnev, int jegy)
            :IrasMu(szerz, cim, ev), biralo(bnev), jegy(jegy) {}
    void kiir() const;  // nincs getter, mert csak kiírjuk a bírálatot
};
class Diploma: public Szakdoga {
    string biralat;
public:
    Diploma(string szerz, string cim, int ev, string bnev, string bir, int jegy);
    void kiir();        // nincs getter, mert csak kiírjuk a bírálatot
};
class Doktori: public Diploma {
    string biralo2;
    string biralat2;
    int irodalom;
public:
    Doktori(string szerz, string cim, int ev, string bnev, string bir,
            string bnev2, string bir2);
    void kiir() const;  // nincs getter, mert csak kiírjuk a bírálatot
};
class Tarolo {
    vector<IrasMu*> tar;            // alaposztályra mutató pointer kell
public:
    typedef vector<IrasMu*>::iterator iterator; // vektor iterátorát delegáljuk
    void felvesz (IrasMu *ir) {
        tar.push_back(ir);          // vektor push műveletét delegáljuk
    }
    void kiir();
    iterator begin() { return tar.begin(); }    // vektor begin műveletét delegáljuk
    iterator end() { return tar.end(); }        // vektor end műveletét delegáljuk
    iterator keresCimre(string cim);
    void kiirCimeket();
    ~Tarolo(){}
};
Tarolo::iterator Tarolo::keresCimre(string cim) {
    iterator it=begin();
    while ( it!=end() && (*it)->getCim()!=cim )
        ++it;
    return it;
}
void Tarolo::kiirCimeket() {
    priority_queue<string, vector<string>, greater<string> > pq;
    for (iterator it=begin(); it!=end(); ++it)
        pq.push( (*it)->getCim() );
    while (!pq.empty()) {
        cout << pq.top() << endl;
        pq.pop();
    }
}
```

Azonos címűek keresése:

```
Tarolo tar;
for (Tarolo::iterator it1=tar.begin(); it1!=tar.end(); ++it1) {
    Tarolo::iterator it2=it1;
    while (++it2 != tar.end()) {
        if ( (*it1)->getCim() == (*it2)->getCim() ) {
            (*it1)->kiir();
            (*it2)->kiir();
        }
    }
}
```