500.AD Megoldó Aladar 500.D

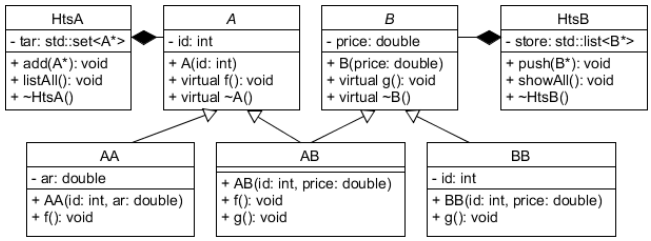
Programozás alapjai 2. (inf.) 2. PZH 2016.05.27. gy./l. hiány: 0/0	Csoport/csop
ALI123 **IB.028/97.**	kZh:0 0 nZh:0+n extra:0+0 Hf:#HIÁNYZIK J:0

Minden beadandó megoldását a feladatlapra, a feladat után írja! Készíthet piszkozatot, de csak a feladatlapra írt megoldásokat 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. mobil) nem használható. A feladatokat* ***figyelmesen olvassa*** *el, megoldásukhoz csak akkor használjon STL tárolót vagy algoritmust, ha a feladat ezt külön engedi/kéri!* ***Ne írjon felesleges*** *függvényeket ill. kódot! A feleletválasztós feladatoknál a hibás válaszért pontlevonás jár. A részfeladatokra kapott pontok a feladaton belül előjelesen összeadódnak. Negatív összeg esetén a feladatra kapott pontszám 0. Az első feladatrészben (beugró) minimum 5 pontot el kell érnie ahhoz, hogy a többit értékeljük.*

f.	max.	elért	jav.
1.	10		
2.	10		
3.	10		
4.	10		
Σ	**40**		

1. feladat: Beugró. A feladatok megoldásához **használjon STL elemeket**, ha lehet**!** **Σ 10 pont**

a) Deklarálja az alábbi osztálydiagramon szereplő osztályok közül az *B*, *BB*, és *HtsB* osztályokat. A konstruktorokat inline függvényként valósítsa is meg! A tagváltozók kezdeti értékét a konstruktorparaméterek adják. (3p)



Egy lehetséges megoldás:

```
class B {
    double price;
public:
    B(double price) :price(price) {}
    virtual void g() = 0;
    virtual ~B();
};

class BB : public B {
    int id;
public:
    BB(int id, double price) :B(price), id(id)
{}
    void g();
};

class HtsB {
    std::list<B*> store;
public:
    void push(B* b);
    void showAll();
    ~HtsB();
};
```

b) Tételezze fel, hogy az a) részfeladat összes osztálya létezik! Szintaktikailag helyes-e az alábbi kódrészlet? (1p)

```
HtsB().push(new AB(4, 7.8));
```

Helyes

c) Valósítsa meg az a) részfeladatban deklarált heterogén tároló (*HtsB*) *push* és *showAll* tagfüggvényét. A *showAll()* hívja meg a „tárolt” objektumok g*()* tagfüggvényét! (2p)

Egy lehetséges megoldás:

```
void HtsB::push(B* a) { store.push_back(a); }

void HtsB::showAll() {
    typedef std::list<B*>::iterator iter;
    for (iter i = store.begin();
                          i != store.end(); ++i)
        (*i)->g();
}
```

d) Mit ír ki az alábbi program a szabványos kimenetre? (2p)

```
using std::cout;
using std::endl;
struct B {
   B()          { cout << "K" << endl; }
   B(const B&)  { cout << "C" << endl; }
   ~B()         { cout << "D"; }
   B& operator=( const B&) {
                  cout << "=" << endl;
                  return *this;         }
};
int main() {
   B b;        // K_____________________
   B a = b;    // C_____________________
   b = a;      // =_____________________
   return 0;   // DD____________________
}
```

e) Jelölje, hogy igaz (I) vagy hamis (H)! (2p)

Konstruktor nem dobhat kivételt.	I	**H**
Absztrakt osztály nem példányosítható.	**I**	H
Az implicit másoló konstruktor nem hívja meg az alaposztály másoló konstruktorát.	I	**H**
A konstruktor nem lehet virtuális.	**I**	H

2. Feladat **Σ 10 pont**

a) Valósítsa meg az *std::generate* generikus algoritmust a *pzh* névtérben *generate* néven! Az *std::generate* az iterátorokkal megadott jobbról nyílt intervallumot feltölti egy generátor függvény által adott értékekkel. A generátor függvényt, vagy függvényobjektumot az algoritmus szintén paraméterként kapja. (2p)

A következő részfeladatok megoldásához használhat STL elemeket is!

b) Mutassa be az algoritmus használatát: Hozzon létre egy tömböt, ami 20 db egész szám tárolására alkalmas! Készítsen egy olyan funktort, ami mindig a következő egész számot adja! A funktor konstruktorában legyen megadható a kezdőérték! Az elkészített sablont és a funktort felhasználva töltse fel a tömböt egész számokkal 1-től 20-ig! (2p)

c) A **b)** részfeladatban feltöltött tömb elemeivel inicializálva hozzon létre egy *std::vector* típusú tárolót! Olvasson be ebbe a szabványos bemenetről fájl végéig érkező további számokat! (2p)

d) Másolja át előző feladatban létrehozott vektor elemeit egy *std::list* tárolóba! Törölje a listáról a húsznál nagyobb elemeket, majd rendezze a listát nagyság szerint, végül írja ki a lista tartalmát a szabványos kimenetre! (4p)

Egy lehetséges megoldás:

```
namespace pzh {
  template <class Iter, class Gen>
  void generate(Iter first, Iter last, Gen gen) {
      while (first != last) {
          *first = gen();
          ++first;
      }
  }
}

struct Seq {
   int next;
   Seq(int i = 1) : next(i) {}
   int operator()() { return next++; }
};

int tomb[20];
Seq seq(1);
pzh::generate(tomb, tomb+20, seq);

std::vector<int> v(tomb, tomb+20);
int i;
while (std::cin >> i)
  v.push_back(i);

std::list<int> l(v.begin(), v.end());
l.erase(std::remove_if(l.begin(), l.end(), std::bind2nd(std::greater<int>(), 20)), l.end());
       // vagy: l.remove_if(std::bind2nd(std::greater<int>(), 20));
l.sort();
std::ostream_iterator<int> out(std::cout, ", ");
std::copy(l.begin(), l.end(), out);
```

3. Feladat **Σ 10 pont**

A TV-t szeretnénk a telefonunkról ki-, ill. bekapcsolni. Ehhez rendelkezésünkre áll egy olyan osztály (*TV*), ami képes a TV-t távvezérelni. Van továbbá egy *Gomb* osztályunk is a telefonon, amihez tartozó grafikus elemet megérintve (megnyomva), az objektum aktivizálja (meghívja) az konstruktorban megadott *Callback* típusú objektum függvényhívás operátorát. Mivel a grafikus felülettel most nem akarunk foglalkozni, ezért ezt az osztályt leegyszerűsítjük úgy, hogy a gomb megnyomását a *megnyom()* tagfüggvénye szimulálja. Az osztályok fontosabb publikus tagfüggvényeit a következő táblázatban foglaljuk össze:

virtual void Callback::operator()()	függvényhívás operátor
Gomb::Gomb(Callback&)	konstruktor
void Gomb::megnyom()	gombnyomás szimulálása
void TV::on()	TV bekapcsolása
void TV::off()	TV kikapcsolása
bool TV::isOn()	igaz értékkel tér vissza, ha a TV be van kapcsolva

a) **Definiálja** (valósítsa meg) a *Callback* absztrakt osztályt! (2p)

b) **Definiálja** (valósítsa meg) a leegyszerűsített *Gomb* osztályt! (3p)

c) A TV, Gomb és Callback felhasználásával, azok módosítása nélkül **tervezzen és valósítson** meg egy olyan osztályt (*TVm*), ami átadható a *Gomb* osztály konstruktorának és a gomb megérintésekor bekapcsolja a TV-t, ha az ki volt kapcsolva, ill. kikapcsolja, ha az be volt kapcsolva! Az osztály legyen kompatibilis a *TV* osztállyal! (3p)

d) Egy programrészlettel **mutassa be** az elkészített *TVm* osztály használatát: Hozzon létre egy példányt, amit egy nyomógomb objektummal tud vezérelni, majd szimuláljon egy gombnyomást! (2p)

Egy lehetséges megoldás:

```
struct Callback {
    virtual void operator()() = 0;
    virtual ~Callback() {}
};

class Gomb {
    Callback& cb;
public:
    Gomb(Callback& cb) : cb(cb) {}
    void megnyom() { cb(); }
};

struct TVm : public TV, public Callback {
    void operator()() {
        if (isOn()) off();
        else on();
    }
};

    TVm ctr;
    Gomb g1(ctr);
    g1.megnyom();
```

---

4. Feladat **Σ 10 pont**

Banki rendszerben értékpapírszámlán (*Account*) különböző értékpapírokat (*Security*) tartanak nyilván. Értékpapírnak mindig van azonosítója (*id*, típus: *string*), ami lekérdezhető, de a létrehozását követően nem módosítható. Az értékpapírnak le lehet kérdezni az aktuális értékét is (*getValue*). Értékpapír lehet kötvény (*Bond*, jellemzője a névérték (*value*, típus: *double*), lejárati idő (*expiry*, típus: *date*)), részvény (*Shares*, jellemzője a mennyiség (*amount*, típus: *int*)). Később a rendszerben további értékpapírtípusokat is kezelni kell tudni. Az értékpapír összes adatát ki is lehet nyomtatni egy paraméterben megadott ostream-re (*print*). Ha a számla megszűnik, a rajta nyilvántartott értékpapírok is megsemmisülnek. Értékpapírt mindig hozzá lehet adni egy számlához (*add*), valamint a számla aktuális egyenlege is lekérdezhető (*getBalance*). Számla tartalma a paraméterben megadott ostreamre kilistázható (*list*). A *date* típus konstruktorai intelligens módon támogatják a dátumok inicializálását, és a *date* típusnak van stringgé kasztoló operátora. A modellben megvalósítandó műveleteket az alábbi kódrészlettel mutatjuk be:

```
Account acc;
acc.add(new Bond("JAMES007", 25000.0, date("2021-03-31")));
acc.add(new Shares("Praetor", 300));
acc.list(std::cout);
std::cout << acc.getBalance () << endl;
```

Tervezzen meg és **rajzoljon** fel egy olyan osztályhierarchiát, ami alkalmas a feladat megvalósítására, és könnyen bővíthető újabb értékpapírtípusokkal az *Account* osztály módosítása nélkül! Az attribútumok kivétel nélkül legyenek privát láthatóságúak. Az osztálydiagramban jelölje az adattagok és metódusok láthatóságát is, valamint a virtuális függvényeket is! A *string* és *date* osztályt nem kell lerajzolni, arra típusként hivatkozzon! Ügyeljen a helyes jelölésekre!

Deklarálja C++ nyelven az *Account*, *Bond* és *Security* osztályokat! (A konstruktoroknak és tagfüggvényeknek csak a deklarációját adja meg!). Használjon *std::list* tárolót!

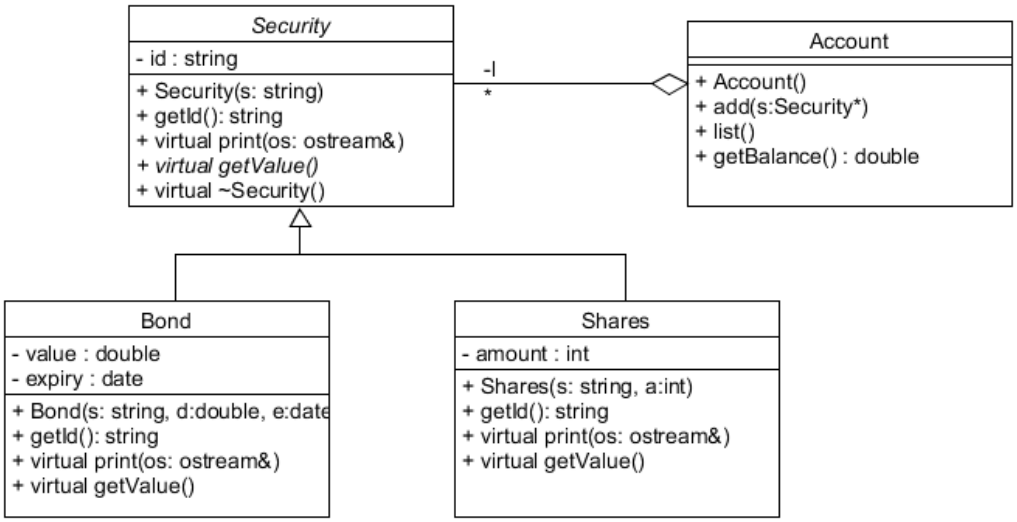
Valósítsa meg az *Account* osztály minden tagfüggvényét, valamint az *Security* és a *Bond* osztály konstruktorát!

Kívül definiált tagfüggvényekként **valósítsa meg továbbá** a megtervezett modell összes olyan függvényét, ami meghívódik a fenti példában a

```
acc.list(std::cout);
```

utasítás végrehajtása során!

Egy lehetséges megoldás:




```
class Security {
  const string id;
public:
  Security (string s);
  virtual ~Security();
  string getId();
  virtual void print(ostream& os);
  virtual double getValue() = 0;
};

class Account {
  list<Security*> l;
public:
  ~Account();
  void add(Security* s);
  double getBalance();
  void list(ostream& os);
};

class Bond : public Security {
  double value;
  date expiry;
public:
  Bond(string i, double v, date e);
  virtual void print(ostream& os);
  virtual double getValue();
};
```

```
typedef list<Security*>::iterator iter;
Account::~Account() {
  for (iter i = l.begin(); i != l.end(); ++i)
    delete *i;
}
void Account::add(Security* s) {
  l.push_back(s);
}
double Account::getBalance() {
  double bal = 0.0;
  for (iter i = l.begin(); i != l.end(); ++i)
    bal += (*i)->getValue();
  return bal;
}

void Account::list(ostream& os) {
  for (iter i = l.begin(); i != l.end(); ++i)
    (*i)->print(os); os << endl;
}
Security::Security(string s) : id(s) {}

Bond::Bond(string i, double v, date e)
         : Security(i), value(v), expiry(e) {}

void Security::print(ostream& os) {
  os << id;
}

void Bond::print(ostream& os) {
  Security::print(os);
  os << ", " << value;
  os << ", " << (string)expiry;
}
```