# 191. BB Megoldó Alfréd BB 191.

| Programozás alapjai 2. (inf.) pótzárthelyi 2011.05.19. gyak. hiányzás: 0 | kZHpont: 0 |
|---|---|
| **N123OO** **IB.028/1.** | Hftest:**0,NZH:n** |

***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. Számítógép, notebook, menedzser kalkulátor, organiser, mobiltelefon nem használható.*

*A feladatokat **figyelmesen olvassa** el, megoldásukhoz **ne használjon fel STL** tárolót, kivéve, ha a feladat ezt külön engedi! **Ne írjon felesleges** függvényeket ill. kódot, a feladatra koncentráljon! Jó munkát!* *Elégséges szint: 8p*

| F. | Max. | Elért |
|---|---|---|
| 1 | 3 | |
| 2 | 5 | |
| 3 | 3 | |
| 4 | 3 | |
| 5 | 6 | |
| **Σ** | **20** | |

1. Feladat 6*0.5=3 pont

**Mit ír ki** a szabványos kimenetre az alábbi program? Válaszához használja a négyzetrácsos területet! A soremelések segítségül szerepelnek a programban, de nem szükséges a soremelésekkel foglalkoznia. Abba a sorba írjon, amelyik sorban a kiírás keletkezik, ez lehet egy zárójelet tartalmazó sor is! A többit hagyja üresen!

```
#include <iostream>
#include <string>
using namespace std;

class Hiba {
  int h;
public:
  Hiba(int h) :h(h) { cout << 'H' << endl << endl;}
  int hibaKod() { return h; };
};

struct A0 {
  A0() { cout << '1'; }
  ~A0() { cout << 'd'; }
};

class Alap : public A0 {
  int al;
  string n;
public:
  Alap(int i, const char *p = "N123OO") : n(p) {
      if (i == 0) throw Hiba(1);
      al = i;
      cout << i << 'k';
  }
  Alap(const Alap& a) :al(a.al)     { cout << al << 'C' << endl << endl;}
  Alap& operator=(const Alap& a) { al = a.al; cout << a.n; return *this; }
  ~Alap()   { cout << 'D'; }
};

int main() {
  Alap a0(52);                    cout << endl;
  try {                           cout << endl;
    Alap a1(1, "23o");            cout << endl;
    a0 = a1;                      cout << endl;
    Alap a2(0,"n1");              cout << endl;
    Alap a3 = a1;                 cout << endl;
  } catch (Hiba& h) {             cout << endl;
      cout << h.hibaKod();        cout << endl;
  }                               cout << endl;
  return(0);
}
```

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 2 | k | | | | | | | | | | |
| | | | | | | | | | | | | | |
| 1 | 1 | k | | | | | | | | | | | |
| 2 | 3 | o | | | | | | | | | | | |
| 1 | H | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| d | D | d | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| D | d | | | | | | | | | | | | |

Az a2 a kivétel miatt nejött létre, ezért a destruktora sem fog lefutni. (ld. 10. előadás 12-13. slide)!

2. Feladat Σ 5 pont

**Készítsen** C++ nyelven egy generikus vektor (*Vektor*) osztályt, amely sablonparaméterként kapja a vektor méretét, és egy osztályt is, amit indexelési hiba esetén kivételként eldob! Ez utóbbinak legyen egy standard hibaosztály az alapértelmezése! Az osztálynak legyen alapértelmezett és egyparaméteres konstruktora is. Alapértelmezett esetben a vektor minden eleme a generikus típus alapértelmezett értéke (alaptípus esetén 0) legyen, egyparaméteres esetben pedig minden eleme a paraméterként megadott érték legyen! Valósítsa meg a következő operátorokat:

- két vektor különbsége a - operátorral
- két vektor összege (+)
- balértékként is használható index ([ ]) operátor

Az osztály legyen átadható érték szerint, és működjön helyesen a többszörös értékadás is!
**Sorolja fel**, hogy a generikus adatra vonatkozóan mely tagfüggvények meglétét tételezi fel a megvalósítás!
**Hozzon** létre egy olyan vektort, ami 2 db char típusú elemet tartalmaz! Specializálja az osztály operator- műveletét úgy, hogy moduló 10 aritmetika (az eredmény 10-es maradékát kell venni) szerint működjön!

Egy lehetséges megoldás: (A feladat sokban hasonlít a 8. hét 2. laborfeladatához, és a minta házi feladathoz is, de kisZH-n is volt hasonló feladat)!

```
template <class T, int size, class E = std::range_error>
class Vector {
    T adat[size];
public:
    Vector(T val = T()) { // T() – a generikus típus alapértelmezett értéke. Alaptípus estén 0
        for (int i = 0; i < size; i++)
            adat[i] = val;
    }
    Vector operator+(const Vector& v) const {
        Vector osszeg;
        for (int i = 0; i < size; i++)
            osszeg.adat[i] = adat[i] + v.adat[i];
        return osszeg;
    }
    Vector operator-(const Vector& v) const {
       Vector kulonbseg;
        for (int i = 0; i < size; i++)
            kulonbseg.adat[i] = adat[i] + v.adat[i];
        return kulonbseg;
    }
    T& operator[](unsigned int i) {
        if (i >= size) throw E("Vector::Index");
            return adat[i];
    }
};
```

Generikus adat tagfüggvényei: `Konstruktor/destruktor, op+, op-, op=`

2 db char elemet tartalmazó vektor: `Vector<char, 2> v3;`

Specializáció:

```
template <>
Vector<char,2> Vector<char, 2>::operator-(const Vector& v) const {
    Vector kulonbseg;
    for (int i = 0; i < 2; i++)
        kulonbseg.adat[i] = (adat[i] - v.adat[i]) % 10;
    return kulonbseg;
}
```

További tagfüggvények (ebben a csoportban nem kellett megírni):

```
template <class T, int size, class E>
T  Vector<T, size, E>::operator*(const Vector& v) const {
    T tmp = T();
    for (int i = 0; i < size; i++)
        tmp = adat[i] * v.adat[i];
    return tmp;
}
```

3. Feladat Σ 3 pont

Tételezze fel, hogy a második feladatban elkészített sablon jól működik! **Ennek felhasználásával** készítsen egyszerűsített modellt részecskék ütközésének vizsgálatához! **Definiáljon** egy *Reszecske* osztályt, ami tárolja a részecske tömegét (*double*), töltését (int) és impulzusát (*3D double vektor*). Legyen az osztálynak minden adata lekérdezhető és beállítható (*getter/setter*), valamint legyen egy olyan metódusa, ami az ütközést (*utkozik*) szimulálja. Az ütközés során keletkező új részecske tömege, töltése, és impulzusa a két eredeti részecske megfelelő adatainak (tömeg, töltés, impulzus) összege. **Definiáljon** az osztályhoz olyan << operátort, mellyel egy std::ostream típusú objektumra kiírható a *Reszecske* osztály minden adata! A fentieket felhasználva modellünkben két részecske ütközése így valósítható meg:

```
Reszecske *bozon1 = new Reszecske;
Reszecske *bozon2 = new Reszecske;
Reszecske *bozon3 = bozon1->utkozik(bozon2);
delete bozon1; delete bozon2;
cout << *bozon3;
```

**Valósítsa** meg a következőket: << operátor, *utkozik* tagfüggvény!

Egy lehetséges megoldás:

```
class Reszecske {
    double tomeg;
    int    toltes;
    typedef Vector<double, 3> imp_t;
    imp_t impulzus;
public:
    double getTomeg() const;
    int getToltes() const;
    imp_t getImpulzus() const;
    void getTomeg(double);
    void getToltes(int);
    void getImpulzus(imp_t);
    Reszecske* utkozik(const Reszecske*);
};

Reszecske* Reszecske::utkozik(const Reszecske *r2) {
    Reszecske* ret = new Reszecske;
    ret->tomeg = tomeg + r2->tomeg;
    ret->toltes = toltes + r2->toltes;
    ret->impulzus = impulzus + r2->impulzus;
    return ret;
}

ostream& operator<<(ostream& os, Reszecske& r) {
    os << "Tomeg:" << r.getTomeg();
    os << " Toltes:" << r.getToltes();
    os << " (" << r.getImpulzus()[0];
    os << "," << r.getImpulzus()[1];
    os << "," << r.getImpulzus()[2] << ")";
    return os;
}
```

4. Feladat Σ 3 pont

Tételezze fel, hogy a 2. és 3. feladatban elkészített sablon és osztály a rendelkezésére áll. **Ezek felhasználásával deklaráljon** egy *Reaktor* osztályt, melyben maximum 150 részecskéket tud tárolni és ütközéseket szimulálni! Az osztály konstruktora hozzon létre 150 részecskét, majd *a react* tagfüggvénye ütköztesse azokat, azaz menjen végig a tárolón, és ütköztesse a tárolóban levő részecskéket! Az ütközés során keletkező új részecskét a két megszűnő közül az egyik helyére tegye a tárolóba! Az ütközésekben részvevő részecskéket bármilyen sorrendben kiválaszthatja. A folyamatot akkor állítsa le, amikor már csak 1 részecske marad! **Ügyeljen** arra, hogy ne lépjen fel memóriaszivárgás! **Írjon** programrészletet, ami bemutatja a Reaktor osztály használatát! A szabványos kimenetre írja ki a folyamat végén megmaradt részecske adatait!

Egy lehetséges megoldás:

```
class Reaktor {
    static const int db = 150;
    Vector<Reszecske*, db> tar;
    Reaktor(const Reaktor &);
    Reaktor& operator=(const Reaktor &);
public:
    Reaktor() {
        for (int i = 0; i < db; i++) tar[i] = new Reszecske;
    }
    void react() {
        for (int i = 0; i < db-1; i++) {
            Reszecske *tmp = tar[i]->utkozik(tar[i+1]);
            delete tar[i];
            tar[i] = 0;
            delete tar[i+1];
            tar[i+1] = tmp;
        }
    }
    void kiir() {
        cout << *tar[db-1];
    }
    ~Reaktor() {
        for (int i = 0; i < db; i++) delete tar[i];
    }
};

Reaktor rr;
rr.react();
rr.kiir();
```

5. Feladat Σ 6 pont

Egy bevásárlóközpontot (*Plaza*) szeretnénk modellezni. A plázát különféle **vevők** (*Vevo*) látogatják, akik eltérő módszerekkel végigjárják a plázában található különféle **boltokat** (*Bolt*). A boltok eltérő módon szolgálják ki a vevőket: például a **mozi** (*Mozi*) minden jegyet 1000 forintért ad, a **lottózó** (*Lottozo*) pedig minden vásárlótól beszed 200 forintot, de minden 10. vásárlónak 400 forintot ad. A vevők eltérő mennyiségű pénzzel indulnak vásárolni, és különböző módszerekkel látogatják a boltokat. Az **alapos** (*Alapos*) vevő minden boltot végigjár, míg a **sietős** (*Sietos*) csak a legelső boltba megy be. A plázában a boltok sorban épülnek. A boltok bejárását egy információs rendszer segíti, ami azonban csak 3 dolgot tud: 1. tudja, hogy hol van az első bolt; 2. tudja, hogy melyik a következő bolt; 3. tudja, hogy hol lesz a következő, még fel nem épült bolt. A rendszerben az alábbi műveleteket (tagfüggvényeket) kell megvalósítani:

**Pláza**:

1. Új bolt felvétele (*ujBolt*)
2. Első boltra mutató előre haladó iterátor létrehozása (*begin*)
3. Az utolsó bolt utáni, még fel nem épült boltra mutató iterátor létrehozása (*end*)
4. Előre haladó iterátor osztály szokásos műveletei.

**Bolt**:

5. Vevő kiszolgálása (*kiszolgal*)

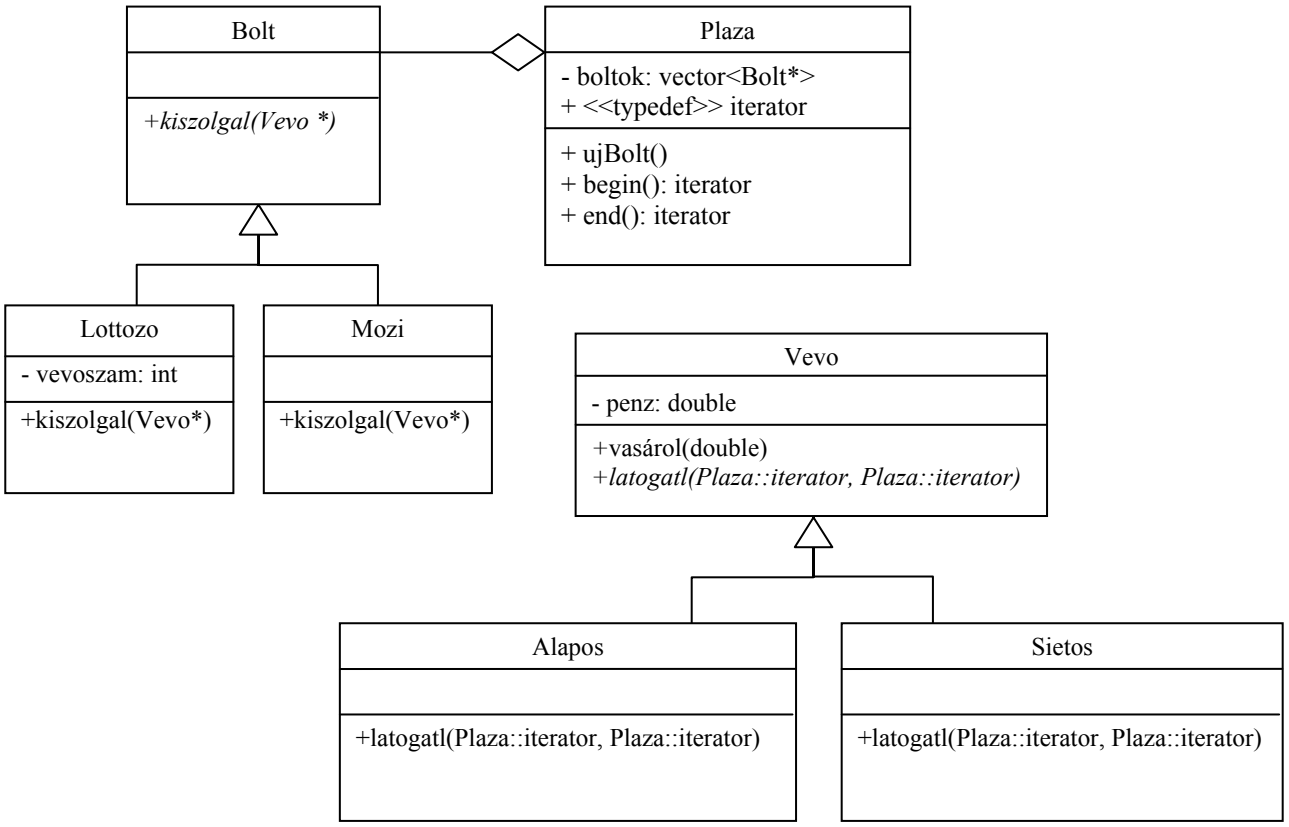
**Vevő**:

6. Vásárlás (*vasarol*). Paraméterként kapja kifizetendő/jóváírandó összeget (negatív: fizet). Ha a vásárló pénztárcája kiürül, vagy negatívba fordul, akkor dobjon std::runtime_error kivételt!
7. Boltok bejárása (*latogat*). Két iterátorral megadott boltintervallumot kap paraméterként (pl: begin, end)

**Feladatok:**

- **Tervezzen** egy OO modellt a problémának megfelelően, amelyben van *Plaza*, *Mozi, Lottozo, Alapos* és *Sietos* vevő, és könnyen bővíthető újabb típusú boltokkal és vevőkkel. Rajzolja fel a modell osztálydiagramját! Használja a dőlt betűs neveket!
- **Deklarálja** az objektumokat C++ nyelven!
- **Implementálja** (valósítsa meg) a *Plaza, Lottozo, és az Alapos* objektumok összes tagfügvényét!

**Használhat** STL tárolókat, algoritmusokat is!



```
class Bolt;
class Vevo;

class Plaza {
    vector<Bolt*> boltok;
public:
    typedef vector<Bolt*>::iterator iterator;
    void ujBolt(Bolt *bp) { boltok.push_back(bp); }
    iterator begin() {return boltok.begin();}
    iterator end() {return boltok.end();}
};

class Bolt {
public:
    virtual void kiszolgal(Vevo *vp) = 0;
    virtual ~Bolt() {}
};

class Vevo {
protected:
    int penz;
public:
    Vevo(int p):penz(p) {}
    void vasarol(int ennyiert);{
    virtual void latogat(Plaza::iterator eleje, Plaza::iterator vege) = 0;
    virtual ~Vevo() {}
};

class Mozi :public Bolt {
public:
    void kiszolgal(Vevo *vp);
};

class Lottozo :public Bolt {
    int vevoszam;
public:
    Lottozo():vevoszam(0) {}
    void kiszolgal(Vevo *vp) {
        vevoszam++;
        vp->vasarol(-200);
        if (vevoszam % 10 == 0) vp->vasarol(+400);
    }
};

class Alapos :public Vevo {
public:
    Alapos(int p) :Vevo(p) {}
    void latogat(Plaza::iterator eleje, Plaza::iterator vege) {
        while (eleje != vege)
           (*eleje++)->kiszolgal(this);
    }
};

class SietosVevo :public Vevo {
public:
    SietosVevo(int p) :Vevo(p) {}
    void latogat(Plaza::iterator eleje, Plaza::iterator vege);
};
```