500.J Megoldó Béla Bendegúz 500.J

Programozás alapjai 2. (inf.) 1. pZH 2017.09.09. gyak./lab. hiányzás:	/
ABC123 **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.

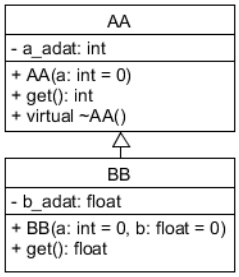
f.	max.	Elért	jav.
1.	10		
2.	10		
3.	10		
4.	10		
Σ	**40**		
Van IMSC megoldásom:			
IM.	10		

1. feladat: Beugró **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 pontokkal. (3p)

A privát adattagok a scope (::) operátorral mindig elérhetők.	I	**H**
Referencia típusú tagváltozót konstruktor törzsében lehet inicializálni.	I	**H**
A logikai típus egész értéket is kaphat értékül.	**I**	H
B osztály kompatibilis A-val, ha B mindenütt használható, ahol A használható.	**I**	H
A destruktor mindig meghívja a tartalmazott objektumok destruktorait.	**I**	H
A konstruktor előbb hajtja végre a programozott törzset, és csak ezután hívja a tartalmazott objektumok konstruktorait.	I	**H**

b) Valósítsa meg C++ nyelven az alábbi osztálydiagramon szereplő osztályokat! A tagváltozók kezdeti értékét a konstruktorparaméterek adják. (2p)




```
class AA {
    int a_adat;
public:
    AA(int i = 0) :a_adat(i) {}
    int get() { return a_adat; }
    virtual ~AA() {}
};
class BB : public AA {
    int b_adat;
public:
    BB(int i = 0, float f = 0) : AA(i), b_adat(f) {}
    float get() { return b_adat; }
};
```

c) A **b)** feladatban készített ***AA*** osztályhoz készítsen olyan *inserter* operátort, amivel a tárolt adata egy *std::ostream* típusú adatfolyamra kiírható! Az operátor legyen fűzhető! (Ügyeljen az adattag láthatóságára!) Egy kódrészlettel mutassa be az operátor használatát! Megjegyzésben adja meg azt is, hogy mit ír ki a kódrészlet az adatfolyamra (2p)

```
// Egy lehetséges megoldás: Jegyezze meg: az inserter nem tagfüggvénye az osztálynak
std::ostream& operator<<(std::ostream& os, const AA& aa) {
  return os << aa.get();
}
std::cout << A(1); // 1
```

d) Írjon programrészletet, melyben a dinamikus memóriaterületen létrehoz egy olyan területet, amelyen a b) feladat osztályaiból 100 példányt tud tárolni vegyesen (heterogén!)! Ezt követően olvasson be a szabványos bemenetről egy egész számot (*n*), és dinamikusan hozzon létre *n* darab *AA*, és 100-*n* darab *BB* típusú objektumot, amit a létrehozott tárolóban eltárol. Feltételezheti, hogy 0 <= *n* <= 100! Végül szabadítson fel minden memóriaterületet! (3p)

```
// Egy lehetséges megoldás:
    AA **p = new AA*[100];
    int i = 0, n; std::cin >> n;
    while (i < n) p[i++] = new AA;
    while (i < 100)  p[i++] = new BB;
    for (i = 0; i < 100; i++) delete p[i];
    delete[] p;
```

2. Feladat **Σ 10 pont**

Olvassa végig a feladatot, mielőtt megoldja! Tételezze fel, hogy egy olyan csapatban dolgozik, melynek egy olyan osztályt kell létrehoznia, amely $a_nx^n+...a_1x+a_0$ alakú nem negatív együtthatójú polinomok együtthatóinak (a_n, ..., a_1, a_0) dinamikus tárolására alkalmas. A csapatnak meg kell valósítani az alábbi műveleteket:

operator+	Két polinom összeadása (az azonos indexű együtthatók páronkénti összeadása). Az eredmény egy új polinom, melynek a fokszáma a nagyobb fokszámmal egyezik meg!
operator+=	két polinom összeadása, az **eredmény a bal oldali op.**
operator<<	Polinom eltolása balra. Az eredmény egy új polinom, melyben minden együttható indexe a megadott *n* értékkel növekszik. PL: $3x+4$ (3,4 együtthatókkal) 2-vel eltolva $3x^3+4x^2$ lesz (3,4,0,0).
at	A polinom adott indexű együtthatójának elérése (indexelés). hibás indexelés esetén `std::out_of_range` kivételt dob.
degree	Polinom fokszámát adja. Az ilyen indexű együttható csak akkor lehet 0, ha a fokszám 0.
c_array	a szokásos C stílusú tömbben adja vissza az együtthatókat
calc	Visszaadja a polinom helyettesítési értékét a paraméterben megadott érték esetén. *Pl. 2x+3 esetén p.calc(4) pont 11-et ad.*
erase	törli a polinom elemeit, eredmény egy 0 fokszámú, konstans 0 értékű polinom.
is_null	igaz értéket ad, ha a polinom minden együtthatója 0

Az osztály megvalósításán többen dolgoznak egyszerre, akikkel megállapodtak az osztály belső adatszerkezetében, és a tagfüggvények funkcióiban. Úgy döntöttek, hogy a polinom fokszámának a legmagasabb indexű, nullától különböző értékű együttható indexét tekintik (pl. a $3x^2+4x+6$ polinom fokszáma *2*). A megállapodást az alábbi kommentezett deklaráció rögzíti, amint ki lehet egészíteni a pontozott részen, de **nem lehet változtatni**. Követelmény, hogy az osztály legyen átadható érték szerint függvényparaméterként, és működjön helyesen a többszörös értékadás is. Feltételezhető, hogy az osztályból további osztályokat származtatunk.

Egészítse ki az osztály deklarációját úgy, hogy az a követelményeknek megfelelően minden szükséges tagfüggvény deklarációját tartalmazza (csak deklaráció)! Vegye figyelembe, hogy az *at()* tagfüggvénynek konstans objektumra is működnie kell!

```
class Polinom {
protected:
  size_t deg;     // A polinom fokszáma.
  double *arr;    // Pointer a dinamikus adatterületre. Ezen a területen tároljuk az együtthatókat.
public:
  Polinom ();     // Létrehoz egy konstans nulla polinomot (0*x^0)
  Polinom& operator=(const Polinom&);// értékadó (polinom = polinom)
  Polinom operator+(const Polinom&) const;  // polinom + polinom
  Polinom& operator+=(const Polinom&);// értékadó (polinom += polinom)
  Polinom operator<<(int) const;      //  Polinom eltolása balra
  double& at(size_t);                 // adott indexű együttható elérése Hibát dob, ha az index rossz
  size_t degree() const;              // visszaadja a polinom valós fokszámát;
  const double* c_array() const;      // C stílusú tömböt ad vissza
  double calc(double) const;          // Visszaadja a polinom helyettesítési értékét
  void erase();                       // törli a polinom együtthatóit
  bool is_null();                     // visszaadja, hogy a polinom konstans 0-e

  virtual ~Polinom();
  Polinom(const Polinom&);
  double at(size_t) const;
IMSC: ..........................................................
   operator const double*() const;
   Polinom& operator--()
   Polinom operator--(int)

};
```

Valósítsa meg az alábbi tagfüggvényeket osztályon kívül definiált függvényekkel!

Paraméter nélküli konstruktor, destruktor, egyszerű értékadás (=)

Polinom operator+(const Polinom&), Polinom& operator+=(const Polinom&)

Vegye figyelembe, hogy a mások által írt függvények belső megoldásait nem ismeri, azaz nem használhat ki olyan működést, ami a fenti kódrészletből vagy annak megjegyzéseiből nem olvasható ki.

```
// Egy lehetséges megoldás:
Polinom::Polinom() :deg(0), arr(new double[deg+1]) { arr[0] = 0; }

Polinom::~Polinom()
  delete[] arr;
}

Polinom& Polinom::operator=(const Polinom& rhs) {
  if (this != &rhs) {
    delete[] arr;
    deg = rhs.deg;
    arr = new double[deg+1];
    for (size_t i = 0; i <= deg; i++)
        arr[i] = rhs.arr[i];
  }
  return *this;
}

Polinom Polinom::operator+(const Polinom& rhs) const {
  Polinom tmp;
  tmp.deg = std::max(deg, rhs.deg);
  delete[] tmp.arr;
  tmp.arr = new double[tmp.deg+1];
  for (size_t i = 0; i <= tmp.deg; ++i) {
    double e1 = 0, e2 = 0;
    if (i <= deg) e1 = arr[i];
    if (i <= rhs.deg) e2 = rhs.arr[i];
    tmp.arr[i] = e1 + e2;
  }
  return tmp;
}

Polinom& Polinom::operator+=(const Polinom& rhs) {
  return *this = *this + rhs;
}
```

IMSC feladat:

Valósítsa meg a **Polinom operator<<(int)** tagfüggvényt, valamint egészítse ki a Polinom osztályt úgy, hogy az paraméterként átadható legyen a következő függvénynek:

```
void unknownAlgorithmWithPolynomials(const double *p, size_t n);
```

Így szeretnénk használni:

```
Polinom poli; ...
unknownAlgorithmWithPolynomials(poli, poli.degree()+1);
```

Legyen továbbá a Polinom osztálynak olyan -- **operátora**, ami deriválja a polinomot!
Mindkét autodekremens operátort készítse el! Működjenek az elvárásoknak megfelelően!

```
// IMSC
// Egy lehetséges megoldás:
Polinom Polinom::operator<<(int n) {
  Polinom tmp;
  delete[] tmp.arr;                  // fontos törölni
  tmp.arr = new double[deg+n+1];
  for (int_t i = 0; i <= deg; i++)
       tmp.arr[i] = arr[i];
  tmp.deg = deg+n;
  for (int i = 0; i < n; ++i) tmp.arr[i] = 0;
  return tmp;
}

Polinom::operator const double*() const { return arr; }

Polinom& Polinom::operator--() {
    if (deg) {
        double *tmp = new double[deg];
        for (size_t i = 0; i < deg; ++i)
            tmp[i] = (i+1) * arr[i+1];
        delete[] arr;
        arr = tmp;
        deg--;
    } else {
        arr[0] = 0;
    }
    return *this;
}

Polinom Polinom::operator--(int) {
    Polinom tmp = *this;
    operator--();
    return tmp;
}
```

3. Feladat Σ 10 pont

Tételezze fel, hogy a második feladat *Polinom* osztálya a specifikációnak megfelelően elkészült! **Hozzon létre** egy olyan osztályt (*EnPolinomom*), ami kompatibilis a *Polinom* osztállyal, és van ***operator*=(double)*** és ***operator/=(double)*** tagfüggvénye is! Ezek a polinom összes együtthatóját külön-külön felszorozzák vagy leosztják a paraméterként adott értékkel. Ha az osztásnál 0-t adunk meg, akkor **const int** kivételt dob! A műveletek az értékadó operátoroknak megfelelően az aktuális objektumot (*this) módosítják. Legyen az osztálynak olyan művelete, ami Polinom osztályból *EnPolinomom* osztályt hoz létre!
Készítsen olyan fűzhető inserter operátort (<<) is, ami kilistázza az osztályban tárolt együtthatókat csökkenő indexek sorrendjében egy *std::ostream* típusú objektumra! Az adatokat **pontosvessző** válassza el egymástól! Az utolsó adat után soremelés legyen, ne legyen pontosvessző!
Mutassa be egy rövid programrészleten az **inserter** és a fenti tagfüggvények használatát (konverziót is)! A kódrészletben készítsen egy olyan polinomot is, ami legalább 3-ad fokú! Kapja el az esetlegesen keletkező kivételt!

```
// Egy lehetséges megoldás:
struct EnPolinomom : Polinom {
    EnPolinomom() {}
    EnPolinomom(const Polinom& p) : Polinom(p) {}
    EnPolinomom& operator*=(double d) {
        for (size_t i = 0; i <= deg; i++)
            arr[i] *= d;
        return *this;
    }
    EnPolinomom& operator/=(double d) {
        if (d == 0) throw 12;
        for (size_t i = 0; i <= deg; i++)
            arr[i] /= d;
        return *this;
    }
};

std::ostream& operator<<(std::ostream &os, const EnPolinomom& poli) {
    for (size_t i = poli.degree(); i > 0; --i)
        os << poli.at(i) << ';';
    os << poli.at(0) << std::endl;
    return os;
}

try {
    Polinom p;
    EnPolinomom mp(p);
    mp = mp << 3;
    mp.at(3) = 1;
    mp.at(2) = 2;
    mp.at(1) = 3;
    mp.at(0) = 4;
    std::cout << mp;
    mp *=3;
    std::cout << mp;
    mp /=0;
    std::cout << mp;
} catch (const int) {
    std::cerr << "baj van";
}
```

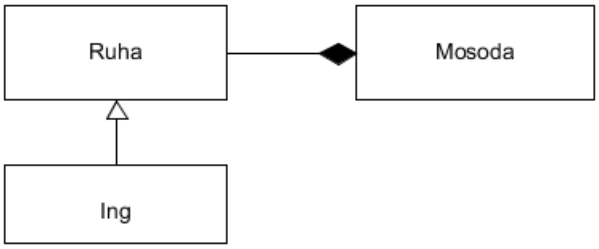
4. Feladat Σ 10 pont

Egy mosoda (*Mosoda*) működését szeretnénk modellezni. Az érkező koszos ruhákat (*Ruha*) a mosoda azonnal kimossa és eltárolja. A ruháknak van tulajdonosuk (*string*). Mosáskor (*mos*) a ruhák koszos állapotból tiszta állapotba kerülnek. Az általános ruhafajtán kívül van a színes ing (*Ing*), ami minden alkalommal 3%-ot veszít a színéből (*double*). Ezen kívül a jövőben további speciális ruhafajtákat tervezünk bevezetni (pl. kétrészes ruha). Ha a mosoda megsemmisül (pl. leég), az összes még ki nem adott ruha is megsemmisül. A rendszerben a következő műveleteket kell megvalósítani:

- új ruha beadása a mosodába (*bead*);
- adott tulajdonos tiszta ruhájának kiadása a *Mosoda*-ból (*kiad*); ha nincs ilyen ruha, dobjon egy standard kivételt;
- mosoda megsemmisülése;
- ruha adatainak kiírása (*kiir*); ing esetén a szín erősségét is írja ki;
- ruha tisztítása (*mos*).

Feladatok:

- **Tervezzen** olyan OO modellt, ami könnyen bővíthető további ruhafajtákkal. Feltételezheti, hogy a mosodában egyszerre maximum 100 ruha lehet. Rajzolja fel a modell osztálydiagramját! Ne tüntesse fel a tagfüggvényeket és adattagokat az UML ábrán! Használja a dőlt betűs neveket!



- **Adja meg** a *Ruha* osztály összes virtuális metódusának deklarációját!

```
virtual void Ruha::mos();
virtual void Ruha::kiir();
virtual Ruha::~Ruha();
```

- **Deklarálja** a *Mosoda* osztályt. A tagfüggvényeknek is csak a deklarációját adja meg.

```
class Mosoda {
    Ruha* ruhak[100];
    int db;
public:
    Mosoda()
    ~Mosoda()
    void bead(Ruha*);
    Ruha* kiad(std::string);
};
```

- **Adja meg** az *Ing* osztály *kiir* metódusának külső definícióját (valósítsa meg), feltételezve, hogy a Ruha osztálynak minden attribútuma privát!

```
void Ing::kiir() {
   Ruha::kiir();
   std::cout << szin << std::endl;
}
```

- **Írjon** egy olyan kódrészletet, ami különböző ruhákat dinamikusan létrehoz, bead a mosodába, és kiadat egy kimosott ruhát, amelynek az adatait kiírja! A kivett ruhát ezután szüntesse meg!

```
Mosoda mosoMasa;
mosoMasa.bead(new Ruha("Jozsi"));
mosoMasa.bead(new Ing("Feri", 23.2));
Ruha *tisztaRuha mosoMasa.kiad("Feri");
if (tisztaRuha) {
      tisztaRuha->kiir();
      delete tisztaRuha;
}
```