500.AA Megoldó Aladár 500.A

Programozás alapjai 2. (inf.) 1. ZH 2017.03.27. gyak./lab. hiányzás:	/
ABC123 **/1.**	kZH:

Minden beadandó megoldását a feladatlapra, a feladat után írja! Készíthet piszkozatot, de csak a feladatlapra írt megoldásokat értékeljük! Az IMSC feladat külön lapra is írható. Ezt csak az alapfeladatok 75%-os teljesítése mellett 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. tablet, notebook, mobiltelefon) 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 feladatban 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**		
IM.	10		

1. feladat: **Σ 10 pont**

a) Definiáljon C++ osztályt síkbeli pontok egész koordinátáinak tárolására (*Point*)! Legyen olyan konstruktora, amivel mindkét koordináta megadható. Legyenek lekérdezhetők a koordináták külön-külön! Az adattagok kívülről közvetlenül ne legyenek elérhetők! Egy kódrészletben hozzon létre az osztályból egy **pont11** nevű példányt x=11 és y=1 értékkel! (2p)

```
// Egy lehetséges megoldás:
class Point {
  int x, y;
public:
  Point(int x, int y) :x(x), y(y) {}
  int getX() const { return x; }
  int getY() const { return y; }
};
  Point pont11(11,1);
```

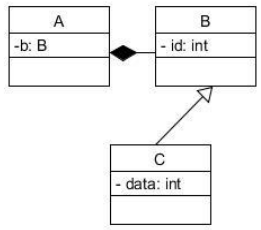
b) Az **a)** feladatban készített osztályhoz készítsen olyan *inserter* operátort, amivel a tárolt koordináták egy *std::ostream* típusú adatfolyamra kiírhatók! Az operátor legyen fűzhető! Egy kódrészlettel mutassa be az operátor használatát! Megjegyzésben adja meg azt is, hogy mit ír ki a kódrészlet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

```
// NEM TAGFÜGGVÉNYE a POINT osztálynak!!!
std::ostream& operator<<(std::ostream& os, const Point& p) {
  return os << p.getX()<< ',' << p.getY();
}
std::cout << Point(1,2); // 1,2
```

c) 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 feladatra kapott pontokkal. (3p)

Referencia típusú tagváltozó csak inicializáló listán inicializálható.	**I**	H
Absztrakt osztálynak nem kell, hogy legyen virtuális tagfüggvénye.	I	**H**
Minden osztályból létre lehet hozni tömböt.	I	**H**
Privát (`private`) adattagot a származtatott objektumból mindig el lehet érni közvetlenül.	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**

d) Valósítsa meg C++ nyelven az alábbi osztálydiagramon szereplő osztályokat úgy, hogy minden adattagnak legyen határozott kezdőértéke! Írjon kódrészletet, ami létrehoz egy objektumot az ***A*** osztályból! Adja meg, hogy mely konstruktorok futottak! (3p)




```
class B {
  int id;
public:
  B() : id(0) {}
};

class C : public B {
  int data;
public:
  C() :data(0) {}
};
```

```
class A { B b; };
{ A a; }   // B::B() és A::A(),
```

2. Feladat **Σ 10 pont**

Maximum **n** csúcspontból álló poligon tárolására alkalmas osztályt (*Poligon*) szeretnénk létrehozni. A csúcspontokat egy dinamikus tömbben tároljuk. **n** értékét a konstruktorban adjuk meg. A létrehozás és megszüntetés mellett meg kell valósítani az alábbi műveleteket:

operator[]	Egy csúcspont adatainak direkt elérése (indexelés). Nincs indexhatár ellenőrzés.
operator+=	Új csúcspont hozzáadása. Nem ellenőrzi, hogy belefér-e.
operator+=	Poligon csúcspontjaival bővíti a poligon pontjait. Nem ellenőriz.
size	Megadja a csúcspontok számát.
capacity	Megadja a csúcspontok maximális számát.

Az osztály megvalósításán többen dolgoznak egyszerre, akikkel megállapodtak, hogy a csúcspontokat az első feladatban elkészült síkbeli pontok tárolására alkalmas osztály (*Point*) segítségével tárolják. Megállapodtak továbbá az osztály belső adatszerkezetében és a tagfüggvények funkcióiban. A megállapodást az alábbi kommentezett deklaráció rögzíti, amin **nem lehet változtatni,** de új **tagfüggvényekkel kiegészíthető**. Követelmény, hogy az osztály **legyen átadható érték szerint** függvényparaméterként, és **működjön** helyesen **a többszörös értékadás** is. Feltételezhető, hogy az osztályból további osztályokat származtatunk.

```
typedef unsigned int size_t;
class Poligon {
  size_t capac;                       // tároló kapacitása (maximális mérete)
  size_t siz;                         // tárolt pontok száma
  Point *pData;                       // pointer az adatok tömbjére
public:
  Poligon(size_t n = 500);            // maximum n csúcspont tárolására alkalmas tárolót hoz létre
  size_t capacity() const;            // megadja a csúcspontok maximális számát
  size_t size() const;                // megadja a csúcspontok számát
  Point& operator[](size_t);          // indexoperátor; nincs indexhatár ellenőrzés
  Point operator[](size_t) const;     // indexoperátor; nincs indexhatár ellenőrzés
  Poligon& operator+=(const Point&); // új csúcspontot ad a poligonhoz (nincs ellenőrzés)
  Poligon& operator+=(const Poligon&);// Poligon csúcspontjait hozzáadja a poligonhoz (nincs ellenőrzés)

  Poligon(const Poligon&);
  Poligon& operator=(const Poligon&);
  virtual ~Poligon();

  operator const Point*() const;          // IMSC
  Poligon& operator--();                  // IMSC
  Poligon operator--(int);                // IMSC

};
```

Egészítse ki az osztály deklarációját úgy, hogy az a követelményeknek megfelelően minden szükséges tagfüggvény deklarációját tartalmazza (csak deklaráció)!

Valósítsa meg az alábbi tagfüggvényeket osztályon kívül definiált függvényekkel!

konstruktor, destruktor, egyszerű értékadás (=)
Poligon& operator+=(const Point&);
size_t size() const;

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.

IMSC feladat:
Egészítse ki a Poligon osztályt úgy, hogy az paraméterként átadható legyen a következő függvénynek:

```
void unknownAlgorithmWithPoints(const Point *p, size_t n);
```

Így szeretnénk használni:

```
Poligon poli; ...
unknownAlgorithmWithPoints(poli, poli.size());
```

Legyen továbbá a Poligon osztálynak olyan **--**operátora, ami törli az utolsó pontot, ha van ilyen! Mindkét autodekremens operátort készítse el! Működjenek az elvárásoknak megfelelően!

```
// Egy lehetséges megoldás:
Poligon::Poligon(size_t n) : capac(n), siz(0), pData(new Point[capac]) {}

Poligon::~Poligon() {
    delete[] pData;
}

Poligon& Poligon::operator=(const Poligon& rhs) {
    if (this != &rhs) {
        delete[] pData;
        capac = rhs.capac;
        siz = rhs.siz;
        pData = new Point[capac];
        for (size_t i = 0; i < siz; i++)
            pData[i] = rhs.pData[i];
    }
    return *this;
}

Poligon& Poligon::operator+=(const Point& p) {
    pData[siz++] = p;
    return *this;
}

size_t Poligon::size() const {
    return siz;
}

// Másik csoportokban kértük:
Poligon::Poligon(const Poligon& rhs) {
    siz = rhs.siz;
    capac = rhs.capac;
    pData = new Point[capac];
    for (size_t i = 0; i < siz; i++)
        pData[i] = rhs.pData[i];
}

Poligon& Poligon::operator+=(const Poligon& p) {
    for (size:t i = 0; i < p.siz; i++)
      pData[siz++] = p.pData[i];
    return *this;
}

size_t Poligon::capacity () const { return capac; }

Point& Point::operator[](size_t i){ return pData[i];

Point Point::operator[](size_t i) const { return pData[i];
```

```
// IMSC
Poligon::operator const Point*() const { return pData; }  // 2p

Poligon& Poligon::operator--() {    // 3p
    if ( siz > 0) siz--;
    return *this;
}

Poligon Poligon::operator--(int) {  // 3p
    Poligon tmp = *this;
    operator--();
    return tmp;
}
```

3. Feladat Σ 10 pont

Tételezze fel, hogy a második feladat *Poligon* osztálya a specifikációnak megfelelően elkészült! Az előadáson megismert adapter tervezési minta alapján **hozzon létre** egy olyan osztályt (*MyPoligon*), ami kompatibilis a *Poligon* osztállyal, és vannak ***at()*** tagfüggvényei is! Ezek az indexeléshez hasonlóan működnek, azaz egy adott indexű elem elérésére használhatók, de ellenőrzik az indexhatárokat, s hiba esetén **const char*** kivételt dobnak!

Készítsen olyan fűzhető inserter operátort (<<) is, ami kilistázza az osztályban tárolt csúcspontokat egy *std::ostream* típusú objektumra! Az adatokat **vessző** válassza el egymástól! Az utolsó adat után soremelés legyen, ne legyen vessző!

Mutassa be egy rövid programrészleten az **inserter** és **mindkét *at()*** tagfüggvény használatát! A kódrészletben készítsen egy olyan sokszöget is, ami legalább 3 pontból áll! Kapja el az esetlegesen keletkező kivételt!

```
// Egy lehetséges megoldás:
struct MyPoligon : Poligon {
    MyPoligon(size_t s = 500) : Poligon(s) {}
    MyPoligon(const Poligon& p) : Poligon(p) {} // nem vártuk el, de konverzió miatt szükséges lehet
    Point& at(size_t i) {
       if (i >= size()) throw "Hibas index";
       return (*this)[i];
    }
    Point at(size_t i) const {
       if (i >= size()) throw "Hibas Index";
       return (*this)[i];
    }
};

// NEM TAGFÜGGVÉNY !!!
std::ostream& operator<<(std::ostream& os, const MyPoligon& p) {
    for (size_t i = 0; i < p.size();) {
       os << p[i++];
       if (i < p.size()) os << ",";
       else os << std::endl;
    }
    return os;
}

try {
  MyPoligon poli(10);
  poli+= Point(1,2);
  poli+= Point(8,5);
  poli+= Point(1,3);
  std::cout << poli << std::endl;
  std::cout << poli.at(2);
  const MyPoligon ures(0);
  std::cout << ures.at(2);
} catch(const char*) {
  ...
}
```

4. Feladat Σ 10 pont

Kocsma italkészletét és vevőkörét szeretnénk nyilvántartani. Minden italnak (Beverage) van neve (name, string) és térfogata (volume, double). Az italok között vannak szeszesitalok (Alcoholic), ezeket az alkoholfok (strength, double) jellemzi; vannak üdítők (Soda), amiket a cukortartalom (int, sugar). Később a készlet újabb italokkal bővülhet. Ha egy vevő (Consumer) kap egy italt (drink), akkor a paraméterként átadott kimeneten megjelenik, hogy éppen mit iszik, és hogy eddig mennyi folyadékot ivott.
A modellben megvalósítandó műveleteket az alábbi kódrészlettel mutatjuk be:

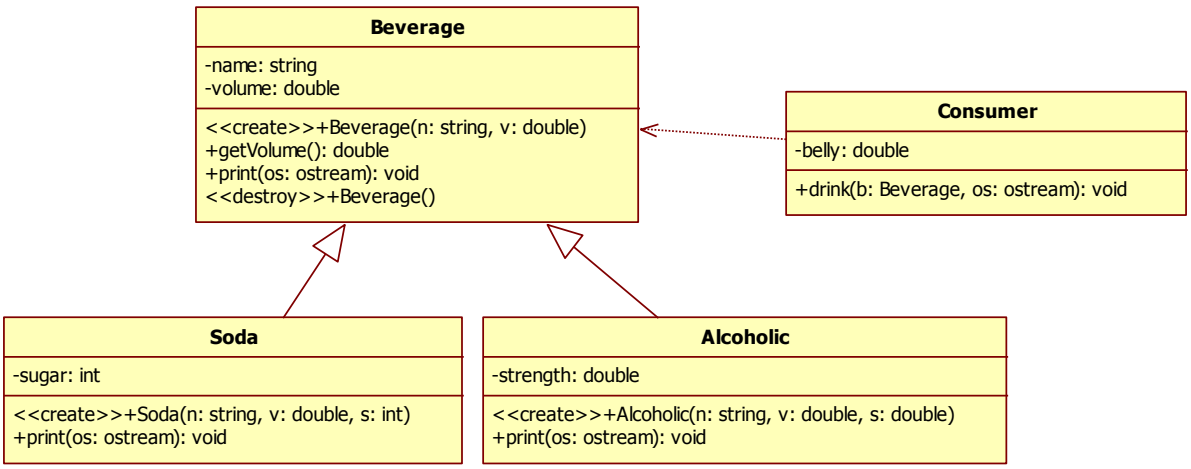
```
Consumer janos;                          // Vevő példány létrejön
Alcoholic hej("Hej-nekem", 0.33, 5.5);   // 0.33l Hej Nekem! sör, 5.5% V/V
Alcoholic csik("Igazi Csoki sir", 0.4, 4.5); // 0.4l Igazi Csoki sir, 4.5% V/V
Soda coala("pss", 0.5, 23456);           // 0.5l Coala, 23456g cukor/liter
janos.drink(hej, std::cout);             // hej adatai + 0.33l
janos.drink(csik, std::cout);            // csik adatai + 0.73l
janos.drink(coala, std::cout);           // coala adatai + 1.23l
```

Tervezzen meg és **rajzoljon** fel egy olyan osztályhierarchiát, ami alkalmas a feladat megvalósítására, és könnyen bővíthető újabb italtípusokkal a felsorolt osztályok módosítása nélkül! Az osztálydiagramban jelölje az adattagok és metódusok láthatóságát is, valamint a virtuális függvényeket is! A *string* osztályt nem kell lerajzolni, arra típusként hivatkozzon! Ügyeljen a helyes jelölésekre!
Deklarálja C++ nyelven a *Beverage*, *Alcoholic* és *Consumer* osztályokat! (Nem kell minden tagfüggvényt megvalósítani!
Valósítsa meg a *Beverage* osztály minden tagfüggvényét, valamint az *Alcoholic* osztály konstruktorát! Minden attribútum legyen **privát** láthatóságú!
Valósítsa meg továbbá a megtervezett modell összes olyan függvényét, ami meghívódik a fenti példában a

```
janos.drink(hej, std::cout);
```

utasítás végrehajtása során!

// Egy lehetséges megoldás:




```
class Beverage {
      string name;
      double volume;
public:
      Beverage(string n, double v) : name(n), volume(v) {}
      double getVolume() { return volume; }
      virtual void print(ostream& os) {
            os << name << "; " << volume;
      }
      virtual ~Beverage() {}
};

class Alcoholic : public Beverage {
      double strength;
public:
      Alcoholic(string n, double v, double s) : Beverage(n,v), strength(s) {}
      void print(ostream& os) {
            Beverage::print(os);
            os << "; " << strength;
      }
};
```

```
class Consumer {
      double belly;
      public:
      Consumer() : belly(0) {}
      void drink(Beverage& b, ostream& os) {
           belly += b.getVolume();
           b.print(os);
           os << "... " << belly << "l" << endl;
      }
};
```