500.AA Megoldó Aladár 500.A

Programozás alapjai 2. (inf.) 1. ZH 2016.04.04. gyak./lab. hiányzás: 0/0+1	G1-IB139/L4-R4K
M/117.	kZH: 7.0+6.5

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

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. 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 feladatrészben (beugró) minimum 5 pontot el kell érnie ahhoz, hogy a többit értékeljük.

1. feladat: Beugró **Σ 10 pont**

a) Definiáljon C++ osztályt, melynek a konstruktora a szabványos kimenetre kiírja a paraméterként kapott karaktersorozatot, a másoló konstruktora, hogy „Copy”, az értékadó operátora hogy „Assign”, a destruktora pedig, hogy „ByeBye”! (2p)

Egy lehetséges megoldás:

```
using std::cout;
struct F1 {
  F1(const char* s) { cout << s; }
  F1(const F1&) { cout << "Copy"; }
  F1& operator=(const F1&) {
       cout << "Assign"; return *this; }
  ~F1() { cout << "ByeBye"; }
};
```

b) Az **a)** feladatban készített osztály felhasználásával írjon olyan programrészletet, amiben legalább egyszer meghívódik mind a négy tagfüggvény! Adja meg, hogy melyik utasítás/sor hatására mit ír ki a program! (2p)

Egy lehetséges megoldás:

```
{
  F1 a("Hello");   // Hello
  F1 b = a;        // Copy
  a = b;           // Assign
}                  // byeByeByeBye
```

c) Mi a hiba az alábbi programrészletben? (1p)

```
{ const int i = 12; void fv(int& j);
  fv(i); }
```

Nem konstans referenciaként konstans referenciát akarunk átadni.

d) Mi a hiba az alábbi programrészletben? (1p)

```
int *ip = new int[10];
delete[10] ip;
```

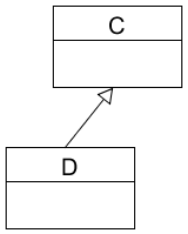
delete[**10**]

A `delete[]` kulcsszó, nem lehet semmit a zárójelek közé írni.

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

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

f) Valósítsa meg C++ nyelven az alábbi osztálydiagramon szereplő osztályokat! (2p)



Néhány lehetséges megoldás:

- `class C { }; class D : public C { };`
- `sruct C { }; struct D : private C { };`
- `class C { }; class D : C { };`
- `sruct C { }; struct D : C { };`

2. Feladat **Σ 10 pont**

Egész számok dinamikus tárolására alkalmas osztályt (Array_1) kell létrehozni. Meg kell valósítani az alábbi műveleteket:

`operator[]`	Egy adat direkt elérése (indexelés). Nincs indexhatár ellenőrzés-
`operator*`	A tömb minden elemét megszorozza egy egész értékkel jobbról.
`operator*`	A tömb minden elemét megszorozza egy egész értékkel balról.
`count`	megszámolja, hogy az adott érték hányszor fordul elő a tömbben.
`size`	megadja a tömb méretét

Az osztály megvalósításán többen dolgoznak egyszerre, akikkel megállapodtak 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.

```
typedef unsigned int size_t;
class Array_1 {
  int *pData;                          // pointer az adatokra
  size_t siz;                          // tömb mérete
public:
  Array_1(size_t n = 10);              // n elemet tárolni képes tárolót hoz létre
  size_t size() const;                 // megadja a tömb méretét
  int& operator[](size_t);             // indexoperátor; nincs indexhatár ellenőrzés
  int operator[](size_t) const;        // indexoperátor; nincs indexhatár ellenőrzés
  int count(int x) const;              // x előfordulási számát adja meg
  Array_1& operator=(const Array_1&);
  virtual ~Array_1() { delete[] pData; }

};

Array_1 operator*(int alpha, const Array_1& rhs) {
  return rhs * alpha;
}
```

A tagfüggvények elkészítését felosztották egymás között. Önre **az értékadó operátor(=), a count** és **a destruktor** megírása jutott. **Valósítsa** meg a **feladatul kapott** tagfüggvényeket! 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:

```
Array_1& Array_1::operator=(const Array_1& rhs) {
  if (this != &rhs) {
    delete[] pData;
    siz = rhs.siz;
    pData = new int[siz];
    for (size_t i = 0; i < siz; i++)
      pData[i] = rhs.pData[i];
  }
  return *this;
}

int Array_1::count(int x) const {
  int cnt = 0;
  for (size_t i = 0; i < siz; i++)
    if (pData[i] == x) ++cnt;
  return cnt;
}
```

3. Feladat Σ 10 pont

Tételezze fel, hogy a 2. feladat Array_1 osztálya a specifikációnak megfelelően elkészült! Az osztály módosítása nélkül **hozzon létre** egy olyan osztályt *(MyArray_1)*, ami kompatibilis az Array_1 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 (<<), ami képes kiírni a *MyArray_1* objektumban tárolt adatokat egy ostream típusú objektumra! Az adatokat vessző válassza el egymástól!

Mutassa be egy rövid programrészleten az **inserter** és mindkét **`at()`** tagfüggvény használatát! Kapja el az esetlegesen keletkező kivételt!

Egy lehetséges megoldás:

```
struct MyArray_1 :public Array_1 {
  MyArray(size_t n = 10) :Array_1(n) {}

  int& at(size_t ix) {
    if (ix >= size()) throw "Hibas index";
    return (*this)[ix];
  }
  int at(size_t ix) const {
    if (ix >= size()) throw "Hibas index";
    return (*this)[ix];
  }
};

std::ostream& operator<<(std::ostream& os, const MyArray_1& a) {
  for (size_t i = 0; i < a.size(); i++)
    os << a[i] << ',';                  // utolsó után is tesz vesszőt
  return os;
}

MyArray_1 ma(12);
for (size_t i = 0; i < ma.size(); i++)
  ma[i] = i;
Array_1* ap = &ma;                      // kompatiblis
const MyArray_1 mb = ma;
try {
  std::cout << ma.at(8) << std::endl;
  std::cout << mb.at(8) << std::endl;
  std::cout << ma << std::endl << mb << std::endl;
  ma.at(20);
} catch (const char *) {
  std::cout << "megvagy";
}
```

4. Feladat Σ 10 pont

Péksüteményeket áruló bolt áruiból és a beérkező vevőkből álló rendszert szeretnénk modellezni. Minden péksüteménynek (*Peksutemeny*) van neve (*string*). A boltban van édes sütemény (*EdesSutemeny*), melynek fontos attribútuma, hogy hány gramm (*double*) cukrot tartalmaz. Van olyan is, amiben töltelék van (*ToltottEdesSutemeny*), melynél a cukortartalom mellett a töltelék megnevezése is fontos (*string*). Természetesen vannak sós sütemények is (*SosSutemeny*), melyeknél a név mellett a zsírtartalom (*double*) a legfontosabb attribútum. A modellben a beérkező vevők (*Vevo*) a pulton levő árukról információt kérhetnek, ami paraméterként átadott *std::ostream* típusú objektumra íródik ki. A modellben megvalósítandó műveleteket az alábbi kódrészlettel mutatjuk be:

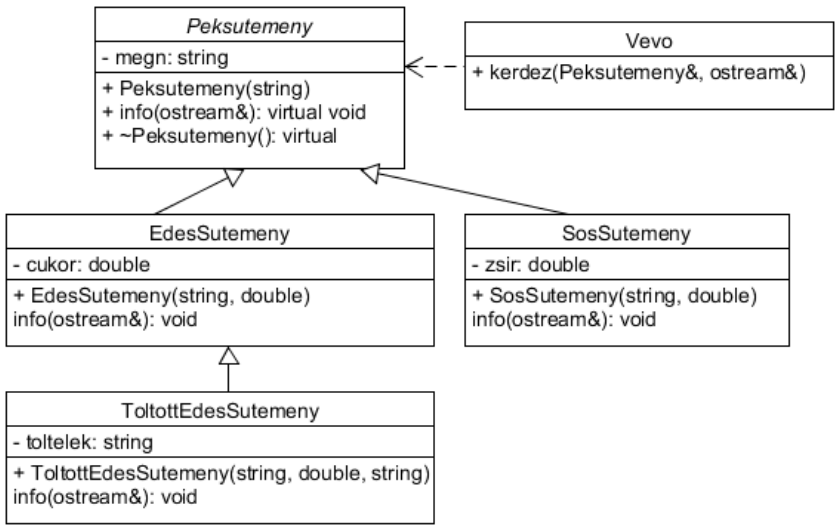
```
Vevo jeno;                                  // Vevő példány létrejön
EdesSutemeny csiga("Kakakos Csiga", 50);    // Kakaós csiga 50 gramm cukorral
ToltottEdesSutemeny retes("Turos retes", 80, "turo"); // Túrós rétes, 80g + túró
SosSutemeny kifli("Sos kifli", 2.4);        // Sós kifli 2.4 gramm zsírral
jeno.kerdez(csiga, std::cout);              // csiga adatai: név, cukor
jeno.kerdez(retes, std::cout);              // rétes adatai: név, cukor, töltelék
jeno.kerdez(kifli, std::cout);              // kifli adatai: név, zsír
```

Tervezzen meg és **rajzoljon** fel egy olyan osztályhierarchiát, ami alkalmas a feladat megvalósítására, és könnyen bővíthető újabb péksüteményekkel a *Vevo* osztály 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 *Peksutemeny*, *EdesSutemeny* és *Vevo* osztályokat! (Nem kell minden tagfüggvényt megvalósítani! Ld. köv. feladat!) **Valósítsa meg** a *Peksutemeny* osztály minden tagfüggvényét, valamint az *EdesSutemeny* osztály konstruktorát!
Valósítsa meg továbbá a megtervezett modell összes olyan függvényét, ami meghívódik a fenti példában a

```
jeno.kerdez(csiga, std::cout);
```

utasítás végrehajtása során!

Egy lehetséges megoldás:




```
class Peksutemeny {
    String  megn;
public:
    Peksutemeny(String nev = "") :megn(nev) {}
    virtual void info(std::ostream& os) { os << megn << std::endl; }
    virtual ~Peksutemeny() {}
};
class EdesSutemeny : public Peksutemeny {
    double cukor;
public:
    EdesSutemeny(String nev = "", doubule cukor = 0) :Peksutemeny(nev), cukor(cukor) {}
    void info(std::ostream& os) {
        Peksutemeny::info(os);
        os << cukor << std::endl;
    }
};
struct Vevo {
    void kerdez(Peksutemeny& suti, std::ostream& os) {
        suti.info(os);
    }
};
```