250. AA Megoldó Alfréd AA 250.

Programozás alapjai 2. (inf.) zárthelyi 2006.05.18. gyakolat: /	Érdemjegy:
a1234c8 **terem:Ch.Max/2.**	Hftest: **0**

Minden beadandó lap tetejére írja fel balra a feladat számát, jobbra a nevét és tankörét! A megoldások során feltételezheti, hogy minden szükséges input adat az előírt formátumban rendelkezésre áll.
A feladatok megoldásához csak a letölthető C és C++ összefoglaló, valamint egy A4-es lapra saját kézzel írt „puska" használható. Számítógép, notebook, menedzser kalkulátor, organiser, rádiótelefon nem használható.

F.	Max.	Elért.
1	5	
2	2	
3	2	
4	3	
5	8	
Σ	**20**	

---

1. Feladat 5 pont

Írjon generikus dinamikus tömb osztályt, amely:

- képes tetszőleges típusú elemeket tárolni!
- indexelhető a szokásos index operátorral ([ ]). Ha az indexelés során kapott index nagyobb, mint a tömb mérete, akkor az növelje meg a méretét a szükséges méretűre, a keletkező üres helyeket töltse fel a tárolt elemtípus default értékével (default konstruktor)! Ügyeljen arra, hogy az index operátorral elért elem balértékként is szerepelhessen!
- meg tudja mondani egy elemről, hogy benne van-e a tömbben (*isElement*)!
- átadható érték szerint függvényparaméterként!
- helyesen kezeli a többszörös értékadást (*v1=v2=v3,* ahol *v1, v2, v3* ugyanazzal a típussal példányosított template-ek)!
- Milyen követelményeket támaszt az osztály a tárolandó típusokkal szemben (pl. alapértelmezett konstruktor rendelkezésre állása)?
- Írjon egy egyszerű, maximum 10 utasításból álló programrészletet, amely bemutatja az osztály szolgáltatásainak használatát!

---

2. Feladat 2 pont

Döntse el, hogy hibásak-e az alábbi programrészletek! Ha igen, adja meg a hiba **helyét** és magyarázatát!
A programrészletekben elírások nincsenek, az összes meghívott függvény ill. hivatkozott osztály létezik, ha nincs megadva a kódrészletben, akkor feltételezze a helyes működést. A memóriaszivárgás is hibának minősül! Válaszait a kódrészletek mellett ill. alatt adja meg!

a)

```
Sta::Sta (const Sta& a){... *this = a;}
Sta& Sta::operator=(Sta a){...};
```

Hiba oka, ha van:

Végtelen ciklusba kerül a másoló konstruktor és az operator=, mert egymást hívják.

c)

```
class A {
      const int c = 2;
};
```

Hiba oka, ha van:

A konstans tagokat a konstruktor inicializáló listájában **kell** inicializálni.

3. Feladat 2 pont

Mit ír ki a szabványos kimenetre a program? Válaszához használja a négyzetrácsos területet!

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

class A {
public:
    A(const int i = 0)   { cout << 'k' << i; }
    A(const A& a)        { cout << 'c'; }
    virtual ~A()         { cout << 'd'; }
};

class B :public A {
    const char *pa;
public:
    B(const char *n = "") :pa(n), A(-1) { cout << 'K' << pa; }
    B(int i) :A(i)       { cout << 'L'; }
    ~B()                 { cout << 'D'; }
};

int main() {
    B b("234");          cout << '\n';
    A a = b;             cout << '\n';
    A* p2 = new B(2);    cout << '\n';
    delete p2;           cout << '\n';
    return(0);
}
```

k	-	1	K	2	3	4				
c										
k	2	L								
D	d									
d	D	d								

4. Feladat 3 pont

Mutassa be, hogy hogyan lehet egy saját osztály (***Aa***) számára átdefiniálni a << operátort annak érdekében, hogy a C++ szabványos kimenetét (pl. *cout*) használhassuk! Pl.: ***Aa a; cout << a;***

Az **operator<<(ostream&, Aa&) globális** függvény átdefiniálásával **lehet**. Ha a kiíráshoz **Aa** privát, vagy protected tagjaihoz is hozzá kell férni, akkor ezt barát függvényként kell definiálni az **Aa** osztályban. Pl:

```
class Aa {     ....
     friend ostream& operator<<(ostream&, const Aa&);  };
ostream & operator<<(ostream& os, const Aa& a) { .... return(os); }
```

Súlyos hiba tagfüggvényként megvalósítani!

5. Feladat 8 pont

A Húsvéti Nyúl háromfajta ajándékot (*Present*) oszt: tojást (*Egg*), csokit (*Chocolate*) és cukrot (*Candy*). Mindegyik különböző értékű, az alapegység egy statikus változója az alaposztálynak (*basePrice=50*), a tojás darabja ennek konstansszorosa (*eggFactor=0.3*), míg a csoki esetében ez a tényező *chocolateFactor=2.4*, a cukornál *candyFactor=1.2*. Programunk egy tárolóban (*Storage*) tartja nyilván nyuszi ajándékkészletét. A tároló maximum 300 ajándék tárolására alkalmas. A tároló egyik funkciója, hogy képes kilistázni a tárolt ajándékok nevét (tojás, csoki, cukor) és összértékét.

- **Tervezze** meg és vázolja fel az OO modell osztálydiagramját! Használja fel a fenti dőlt betűs azonosítókat!
- **Implementálja** az osztályokat és konstansokat figyelve arra, hogy esetlegesen egyes konstansokat is tagként vagy statikus tagként érdemes implementálni. Ne legyen egy függvénytörzsben sem feleslegest kód! Egy új ajándéktípus esetleges felvételéhez ne kelljen a már meglévő osztályokat módosítani!
- **Írjon** egy egyszerű programrészletet, ami megmutatja a három különböző típusú ajándék felvételét egy tárolóba, valamint kiírja a tárolóban levő ajándékokat és összértéküket!

1. feladat egy lehetséges megoldása:

```
#ifndef FELADAT1_A_H
#define FELADAT1_A_H
// feladat1_a.hpp – generikus tömb sablonja

template <class T>
class Array {
      T *tp;                          // Elemek tömbjére mutató pointer
      int nElement;                   // elemek száma
public:
      Array() { nElement = 0; }
      Array(const Array&);            // copy konstruktor
      bool isElement(const T&);
      T& operator[](int i);
      Array& operator=(const Array& e);
      ~Array() {
         if (nElement) delete[] tp;   // csak akkor törlünk, ha volt eleme
    }
};
template<class T>
Array<T>::Array(const Array& e) {   // copy konstruktor
      nElement = 0;                 // operator= híváshoz előkészítjük
      *this = e;                    // most már mehet
}
template <class T>
bool Array<T>::isElement(const T& a) {
      for(int i = 0; i < nElement; i++)// végigmegy a tömb elemein, van-e ilyen elem
         if(tp[i] == a)
            return true;
      return false;
}
template<class T>
T& Array<T>::operator[](int i) {
      if (i < 0) throw "Indexelesi hiba";// hibajelzés (char* nem elegáns!)
      if (i >= nElement) {      // ha nem elég hosszú
         T *p = new T [i+1];
         int j = 0;
         for(; j < nElement; j++)// átmasolja a megnyújtott tömbbe
            p[j] = tp[j];
         for(; j < i; j++)
            p[j] = T();         // feltölti a def. értékkel az új elemeket
         if (nElement)
            delete[] tp;        // ha volt eleme, törli
         tp = p;
         nElement = i + 1;
      }
      return tp[i];
}

template<class T>
Array<T>& Array<T>::operator=(const Array& e) { // értékadás operátor
      if(this != &e) {                    // nem önmagának ad értéket, akkor átmásol
         if (nElement) delete[] tp;
         tp = new T [e.nElement];
         nElement = e.nElement;
         for (int i = 0; i < nElement; i++)
            tp[i] = e.tp[i];
      }
      return *this;
}

#endif
```

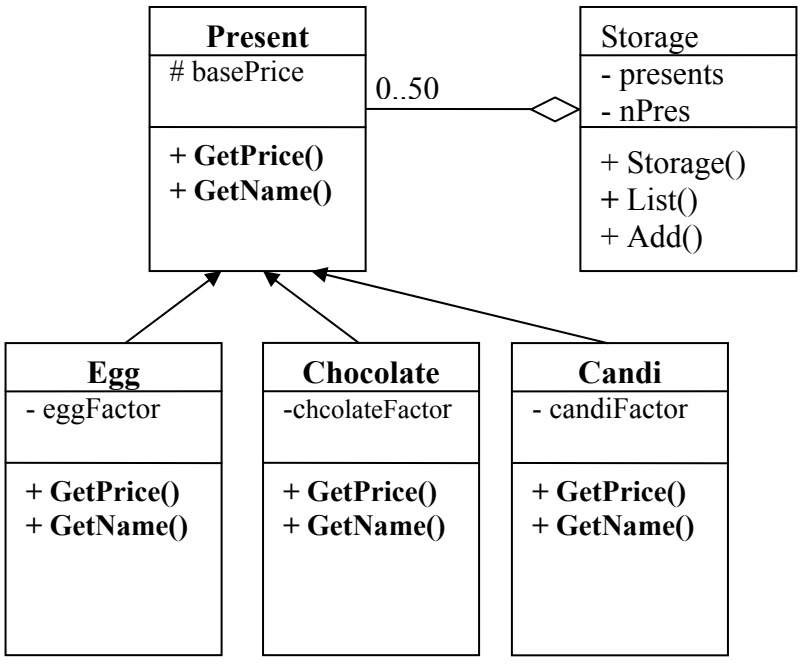
```
// tesztporgram

#include <iostream>
using namespace std;

#include "feladat1a.hpp"

void main()
{
    Array<int> ia;

    ia[0] = 8;
    ia[12] = 23;
    cout << ia[2];
    cout << ia.isElement(3);

    Array<int> ia2 = ia;
    Array<int> ia3;
    ia3 = ia;
    ia2 = ia3;
    ia2 = ia2;
    cout << ia.isElement(23);
}
```

Követelmények a tárolandó típussal szemben:

- legyen default konstruktora
- legyen == operátora
- legyen = operátora

A javításnál nagyon negatívan értékeltük, ha nem a feladathoz tartozó kód került a megoldásokba. Ez ugyanis arra utal, hogy a legális puskáról lemásolt valamit, de fogalma sincs, hogy mit.

5. feladat egy lehetséges megoldása:



A feladat egy tipikus példája a heterogén gyűjtemény alkalmazásának: **Eltérő** tulajdonságú objektumokat akarunk **együtt** kezelni. A tárolóban az **alaposztályra mutató pointereket** tároljuk (nem jó, ha az alaposztályt tárolunk). Pointerek és **virtuális** függvények segítségével elérjük a származtatott osztályok megfelelő metódusait (GetPrice(), GetName()).

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

class Present {
protected:
  static  double basePrice;
public:
  virtual double getPrice() {return basePrice;}
  virtual const char* getName() = 0;
};
double Present::basePrice = 50; // Ez a static miatt fontos

class Egg:public Present {
  const static double eggFactor;
public:
  Egg() {}
  double getPrice() {return basePrice*eggFactor;}   // Tojás ára
  const char* getName() {return "Egg";}             // neve
};
const double Egg::eggFactor = 0.3; // Ez a static miatt fontos

class Chocolate:public Present {
  const static double chocolateFactor;
public:
  Chocolate() {}
  double getPrice() {return basePrice*chocolateFactor;} // Csoki ára
  const char* getName() {return "Chocolate";}           // neve
};
const double Chocolate::chocolateFactor = 2.4; // Ez a static miatt fontos

class Candy:public Present {
  const static double candyFactor;
public:
  Candy() {}
  double getPrice() {return basePrice*candyFactor;}   // cukor ára
  const char* getName() {return "Candy";}             // neve
};
const double Candy::candyFactor = 1.2; // Ez a static miatt fontos

class Storage {
  enum { NPRES = 200 };     // Trükk, hogy ne kelljen define, vagy globális konstans
  Present *presents[NPRES]; // Fontos, hogy az alaposztály pointerét tároljuk
  int nPres;                // tárolt darabszám
public:
  Storage():nPres(0) {}
  void addPresent(Present& p);  // lehetne pointer is, de szerencsés értékként
  void list();
};

void Storage::addPresent(Present& p) {
  if (nPres < NPRES)
    presents[nPres++] = &p;
}

void Storage::list() {
  double total = 0;
  for (int i=0; i<nPres; i++) {
      cout << i+1 << ": " <<presents[i]->getName() <<endl;
      total += presents[i]->getPrice();
  }
  cout << " Osszesen: " << total << endl;
}
```

```
int main()
{
  Storage s;

  Egg e1, e2;
  Candy c1;

  s.addPresent(e1);
  s.addPresent(c1);
  s.addPresent(e2);
  s.addPresent(*(new Chocolate));  // referenciaval a dinamikus obj. rondább

  s.list();

  return 0;
}
```

Nagyon hibás, és teljesen ellentmond az OO tervezésnek, ha valamilyen módon megpróbálja megkülönböztetni a tárolóban a tárolt objektumokat pl. cím vagy id alapján.

Fontos az objektumok felelősségének helyes kezelése. A tároló nem ismerheti a tárolt objektumok belsejét.

Nem azért használjuk a C++ nyelvet, hogy C programokat kódoljunk vele!!