417. AA Megoldó Alfréd AA 417.

Programozás alapjai 2. (inf.) zárthelyi 2008.05.14. gyakorlat: /	Érdemjegy:
Megold/1.	Hftest: **1200**

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 és C++ összefoglaló használható. Számítógép, notebook, menedzser kalkulátor, organiser, rádiótelefon nem használható.

*A feladatokat **figyelmesen olvassa** el, megoldásukhoz **ne használjon fel STL** tárolót, kivéve, a feladat ezt külön engedi! **Ne írjon felesleges** függvényeket ill. kódot, a feladatra koncentráljon!*
Jó munkát! *Értékelés: 0-8:1, 9-11:2, 12-14:3, 15-17:4, 18-20:5*

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

1. Feladat 3 pont

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

```
#include <iostream>
using namespace std;
class A {
  const char *s;
public:
  A(const char *s = "A") :s(s)   { cout << s << 'k'; }
  virtual ~A()                   { cout << 'd'; }
};
class B :public A {
public:
  B(const char *s ="ABCDEF") :A(s){ cout << 'K'; }
  B(const B&)                    { cout << 'C'; }
  ~B()                           { cout << 'D'; }
};
int main() {
  A ab4;         cout << endl;
  B b1("lo4");   cout << endl;
  B b2 = b1;     cout << endl;
  A *ap = new B("lo4"); cout << endl;
  delete ap;     cout << endl;
  return(0);
}
```

A	k													
l	o	4	k	K										
A	k	C												
l	o	4	k	K										
D	d													
D	d	D	d	d										

Nagyon sokan nem vették figyelembe, hogy ha felüldefiniáljuk a másoló konstruktort, és abban explicit nem hívjuk meg az ős osztály másoló konstruktorát, akkor az ős alapértelmezett konstruktora hívódik meg. Előadáson többször elmondtam, laboron pedig legalább 2-szer gyakoroltuk ezt, a tartalmazott objektumok hasonló viselkedésével együtt!

2. Feladat Σ 3 pont

Egy modellezési feladathoz egy olyan osztályt (*F12*) kell készítenie, ami egyrészt átadható a **`rajzol(Rajz &)`** függvénynek, másrészt átadható a **`mukodtet(Modell &)`** függvénynek. A *Rajz* ill. *Modell* osztályok deklarációját nem tudja megváltoztatni. Annyit tud róluk, hogy mindkettőből egy-egy függvényt kell megvalósítania a feladatához. A *Rajz* osztály esetében ez a **`void paint()`** függvény, a *Modell* osztály esetében ez a **`void run()`** függvény.
Deklarálja az *F12* osztályt és definiálja a **`paint()`** és **`run()`** tagfüggvényeket! A **`paint()`** függvény a standard kimenetre írja ki hogy „Paint_1”, a **`run()`** függvény pedig azt, hogy „Run_1”! (2 pont)
Mutasson példát arra, hogy hogyan nézhet ki a *Rajz* osztály deklarációja! A kódrészlet tartalmazza az osztály deklarációjának első sorát és a **`paint()`** függvény deklarációját! (1 pont)

Egy lehetséges megoldás:

```
class F12 :public Rajz, public Modell {
public:
      void paint() {
            cout << ”Paint_1”;
      }
      void run() {
            cout << ”Run_1”;
      }
};

class Rajz .... {
...
      virtual void paint();
....
};
```

Többszörös öröklés egyik legfontosabb alkalmazása, hogy agy osztály több arcot tudjon mutatni. Előadáson több példa volt erre. Laboron is gyakoroltuk. Aki pedig megcsinálta a hftest 12. feladatát, az értette, hogy miért F12-nek hívták ezt az osztályt :)

3. Feladat Σ 5 pont

Egy **generikus** *Vec4* osztályt kell létrehozni, ami egy dinamikusan nyújtózkodó tömböt valósít meg. A tömb kezdeti mérete, és elemeinek kezdőértéke a konstruktorban adható meg. A tömb az indexelés során képes nyújtózkodni. Ilyenkor az új elemek a konstruktorban megadott értéket (*iv*) veszik fel. 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**.

```
template <class T> class Vec4 {
  T*  v;            // Pointer a dinamikus adatterületre. Ezen a területen tároljuk az adatokat
  int siz;          // Tárolt elemek száma, azaz ekkora dinamikus területet foglalunk.
  T   iv;           // Kezdeti érték, ezzel kell feltölteni az új elemeket.
public:
  Vec4 (int n = 0, T iv = T());// Létrehoz egy n elemű vektort, feltölti iv értékkel, inicializál
                    // nulla elemszámhoz is foglal területet, hogy később ne legyen vele baj
  Vec4 (const Vec4&); // Másoló konstruktor.
  Vec4& operator=(const Vec4&);// Értékadó op. Támogatja a többsz. értékadást. Önmagát is ell.
  void resize(int n); // Átméretezi a vektort. Ha n kisebb, mint a pillanatnyi méret, akkor az
            // első n elem megmarad, a többi elveszik. Ha nagyobb, akkor az új elemeket feltölti iv-vel
  int size() const; // Visszaadja a vektorban tárolt elemek számát
  T& operator[](int);// Index operátor, ami képes nyújtani a tömböt (nyújtáskor iv értékkel tölt)
  void erase();     // Felszabadítja a dinamikus területet. Olyan állapotba hozza az objektumot,
                    // hogy hibamentesen működjön tovább (pl. értékadás)
  ~Vec4();          // Megszünteti az objektumot
};
```

417. AA Megoldó Alfréd AA 417.

A tagfüggvények elkészítését felosztották egymás között. Önre a **a konsruktorok** megírása jutott. **Valósítsa** meg 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 (3 pont).

Definiáljon egy olyan függvénysablont *(add)*, amivel egy `Vec4<T>` vektort egy újabb elemmel lehet bővíteni! (2 pont) **Működjenek** helyesen a következő kódrészletek:

```
Vec4<int> ab, ab2(8, -4);
add(ab, 1);        // a vektor 0. eleme 1 lett
add(ab, 2);        // a vektor 1. eleme 2 lett
```

```
ab.erase();
ab = ab2 = ab2;
Vec4<int> ab3 = ab;
cout << ab3[412];
```

„A” csoport megoldása (kontruktorok):

```
template <class T>
Vec4<T>::Vec4(int n, T iv) :iv(iv), siz(n){
  v = new T[siz];
  for (int i = 0; i < siz; i++)
    v[i] = iv;
}
template <class T>
Vec4<T>::Vec4(const Vec4& f) {
  v = new T[siz = 0];
  *this = f;
}
```

„B” csoport megoldása (másoló+értékadó):

```
template <class T>
Vec4<T>::Vec4(const Vec4& f) {
  v = new T[siz = 0];
  *this = f;
}
template <class T>
Vec4<T>& Vec4<T>::operator=(const Vec4& f) {
  if (this != &f) {
    delete[] v;
    v = new T[siz = f.siz];
    for (int i = 0; i < siz; i++)
      v[i] = f.v[i];
    iv = f.iv;
  }
  return *this;
}
```

„C” csoport megoldása (index+size):

```
template <class T>
T& Vec4<T>::operator[](int ix) {
  if (ix >= siz) {
    reize(ix+1);
  }
  return v[ix];
}
template <class T>
int Vec4<T>::size() const {
  return siz;
}
```

„D” csoport megoldása (resize+size):

```
template <class T>
void Vec4<T>::resize(int n) {
  if (n != siz) {
    T *tmp = new T[n];
    int m = n < siz ? n : siz;
    for (int i = 0; i < m; i++)
      tmp[i] = v[i];
    delete[] v;
    v = tmp;
    while (i < n)
      v[i++] = iv;
    siz = n;
  }
}
template <class T>
int Vec4<T>::size() const {
  return siz;
}
```

2. részfeladat megoldása minden csoportnak: (Vigyázat, az add esetünkben nem tagfüggvény!)

```
template <class T>
void add(Vec4<T>& vec, T val) {
  vec[vec.size()] = val;
}
```

Akik nem jöttek rá, hogy a másolóban fel lehet használni az értékadást ill. az indexben pedig a resize()-t, azok kicsi többet írtak, de az sem volt sok:

```
template <class T>
Vec4<T>::Vec4(const Vec4& f){
  v = new T[siz = f.siz];
  for (int i = 0; i < siz; i++)
    v[i] = f.v[i];
  iv = f.iv;
}
```

```
template <class T>
T& Vec4<T>::operator[](int ix) {
  if (ix >= siz) {
    T *tmp = new T[ix+1];
    for (int i = 0; i < siz; i++)
      tmp[i] = v[i];
    delete[] v;
    v = tmp;
    while (i <= ix)
      v[i++] = iv;
    siz = ix + 1;
  }
  return v[ix];
```

```
}
```

4. Feladat Σ 4 pont

Az **STL vector** sablonjának, **vagy a 3. feladat Vec4** sablonjának felhasználásával **készítsen** valós értékeket tároló verem osztályt! Valósítsa meg következő műveleteket:

- *push* – elemet tesz verem tetejére
- *pop* – elemet vesz le verem a tetejéről. A függvény visszatérési értéke a kivett elem legyen!
- *empty* – *true* értéket ad, ha a sor üres.

Az osztály legyen átadható érték szerint függvényparaméterként, és kezelje helyesen a többszörös értékadást (*s1=s2=s3*)! (Az STL vector fontosabb műveletei: at, front, back, push_back, pop_back, insert, empty, resize, =, [])

Egy lehetséges megoldás:

```
class Verem {
    Vec4<double> adat;   // STL vector sablonnal is hasonló a megoldás
public:
    void push(double d) {
        adat[adat.size()] = d;
    }
    double pop() {
        double d = adat[adat.size()-1];
        adat.resize(adat.size()-1);
        return d;
    }
    bool empty() {
        return adat.size() == 0;
    }
};
```

Akik felüldefiniálták a másoló konstruktort és/vagy az értékadást, azok feleslegesen dolgoztak, hiszen a Verem osztály nem kezel dinamikus adattagot, csak a tartalmazott objektum (adat), ami viszont a feltételezésünk szerint jól működik.
Csak akkor kell a másoló konstruktort és/vagy az értékadást felüldefiniálni, ha az alapértelmezés szerinti működés nem jó valamiért!

A feladat szerint vagy az STL vector sablonját, vagy a 3. feladat Vec4 sablonját fel kellett használni!

5. Feladat Σ 5 pont

Egy étterem informatikai rendszerében az egyes **asztalok rendelését** egy-egy rendelési listában szeretnénk tárolni. Egy rendelés több tételből állhat, mint pl: leves, főétel, ital, desszert, stb. A tételek megnevezése mellett mindegyikről különböző adatokat kell tárolni: bor esetében az évjárat, leves esetében az adag mérete, főétel esetében a köret megnevezése. Egy rendelés maximum 54 tételből állhat, amit *list()* tagfüggvénnyel lehet kiíratni. Pl:

Leves: Erdei vargányaleves javasasszony módra, csésze
Ital: Soproni Cabernet Franc, 1904
Főétel: Rozmaringos párolt nyúlgerinc, hagymás törtburgonya.

- **Tervezzen** meg egy olyan OO modellt, mely alapját képezheti a rendelést támogató programnak! Kezdetben csak 2 tétel (*Foetel*, *Leves*) tárolását tegye lehetővé, de legyen a modell könnyen bővíthető! Használhat STL tárolókat is! Rajzolja fel a modell osztálydiagramját! **Használja** fel a következő osztályneveket: ***Foetel, Leves, Asztal, Tetel***!
- **Implementálja** a fenti osztályokat! Ne legyen egy függvénytörzsben sem felesleges kód! Az osztályokat olyan módon készítse el, hogy újabb ételfajta felvételekor a már meglevő kódot ne kelljen módosítani!
- **Írjon** egy egyszerű programrészletet, ami felvesz 2 tételt a példában szereplő ételek közül egy rendelési listára, és kilistázza azt!
- **Osztálydiagram** segítségével mutassa meg, hogy hogyan lehet az elkészített modellt felhasználni egy olyan étterem (***Etterem***) osztályhoz, melyben 8-44 ***Asztal*** van!

Egy lehetséges megoldás:

```
class Tetel {
    string megn;                // Használjuk az STL string sablonját
public:
    Tetel(const char *n) :megn(n) {}
    virtual void kiir() { cout << megn;}
};
class Leves :public Tetel {
    string adag;
public:
    Leves(const char *megn, const char *adag) :Tetel(megn), adag(adag) {}
    void kiir() {
        cout << "Leves: ";
        Tetel::kiir();
        cout << ", " << adag << endl;
    }
};
class Foetel :public Tetel {
    string koret;
public:
    Foetel(const char *megn, const char *koret) :Tetel(megn), koret(koret) {}
    void kiir() {
        cout << "Foetel: ";
        Tetel::kiir();
        cout << ", " << koret << endl;
    }
};
class Asztal {
    vector<Tetel*> v;           // Használjuk az STL string sablonját
public:
    void rendel(Tetel *t) {
        v.push_back(t);
    }
    void rendeles() {
        for (vector<Tetel*>::size_type i = 0; i < v.size(); i++)
            v[i]->kiir();
    }
};
............
 Asztal a1;
 a1.rendel(new Leves("Erdei vargányaleves jasasasszony módra", "csésze"));
 a1.rendel(new Foetel("Rozmaringos párolt nyúlgerinc", "hagymás törtburgonya"));
 a1.rendeles();
```

417. AA Megoldó Alfréd AA 417.

Nem tudom, hogy milyen úton terjed, de kérem, hogy a nevemben rúgják bokán (minimum) azt:

1. Aki a **konstruktorban** (bármelyikben) **`delete`** utasítást használ. Nem állítom, hogy tejesen lehetetlen olyan kódot írni, amiben ez értelmes lenne, de a konstruktor a létrehozásért van és nem a megszüntetésért. Különösen megdöbbentő a **`delete this`** fordulat. Ez csak azoknak ajánlott, akik a faágra kiülve, maguk alatt szokták fűrészelni a fát!
2. Aki a konstruktorban **`return *this`** utasítást használ!
3. Aki **`if (this != ...)`** utasítást használ a másoló konstruktorban úgy, hogy a ... helyén a paraméterként kapott obj. példány címe van. Ez ugyanis mindig teljesülni fog, kár leírni!
4. Aki az egyenlőség operátort akarja visszavezetni a másoló konstruktor hívására. Igaz, hogy ezzel keletkezik egy másolat, de rossz helyen, amit valahogy a helyére (a **`this`** által mutatott objektumba) kellene tenni, ami általában bonyolult, és könnyen elrontható.
5. Aki azt képzeli, hogy egy pointer értékét csak a **`new`** operátor tudja megváltoztatni.

Köszönöm!