117. AA Megoldó Alfréd AA 117.

Programozás alapjai 2. (inf.) pót-pótzárthelyi 2011.05.26. gyak. hiányzás:	kZHpont:
MEG123 IB.028/117.	**NZH:0 PZH:n**

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, C++ és STL összefoglaló használható. Számítógép, notebook, menedzser kalkulátor, organiser, mobiltelefon nem használható.*

*A feladatokat **figyelmesen olvassa** végigl, megoldásukhoz **ne használjon fel STL** tárolót, kivéve, ha a feladat ezt külön engedi! **Ne írjon felesleges** függvényeket ill. kódot, a feladatra koncentráljon! Jó munkát!* *Elégséges szint: 8p*

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

1. Feladat 6*0.5=3 pont

Kollegája két feladatot kapott: 1. készítenie kellett egy olyan **generikus osztálysablont**, mely egy dinamikus tömböt valósít meg. A konstruktoron/destruktoron kívül csak a balértékként is használható index ([ ]) operátort kellett megvalósítania. 2. Be kellett mutatnia az osztály működését egy olyan ***Komplex* osztály** segítségével, ami hozzá tud adni komplex értékhez valós számot a + operátorral, és ki tudja írni magát egy std::ostream típusú objektumra a szokásos << operátorral. Sajnos kollegája elég sok hibát vétett. **Javítsa ki, írja át** a megfelelő helyeken az alábbi kódot úgy, hogy az a fenti elvárásoknak megfelelően működjön!

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

template <typename T>
class Tar {
    T *p1;
    unsigned int s;
    Tar(const Tar&);
    Tar& operator=(const Tar&);
public:
    explicit Tar(unsigned int s) :s(s) { p1 = new T[s]; }
    T& operator[](unsigned int i) {
        if (i >= s) throw range_error("Tar::index");
        return p1 [i];
    }
    virtual ~Tar() { for (int i = 0; i < size; i++) delete[] p1 [i]; }
};

class Komplex {
    double re, im;
public:
    Komplex(double re = 0, double im = 0) :re(re), im(im) {}
    Komplex& operator+(const Komplex k, double c) const {
        Komplex tmp;
        tmp.re = k.re + c; tmp.im = im;
        return *thistmp;
    }
    friend ostream& operator<<(ostream& os, const Komplex& k) {
        os << "(" << k.re << "," << k.im << ")";
        return os;
    }
};

int main() {
    Tar<Komplex> a(101);
    a[0] = a[1] + 10;
    cout << a[0] << endl;
    return 0;
}
```

117. AA Megoldó Alfréd AA 117.

2. Feladat Σ 5 pont

Készítsen C++ nyelven egy generikus halmaz (*Halmaz*) osztályt, amely sablonparaméterként kapja a halmaz maximális méretét (számosságát), és egy osztályt is, amit kivételként dob, ha az elemek száma meghaladná a maximumot! Ez utóbbinak legyen egy standard (STL) hibaosztály az alapértelmezése! Valósítsa meg a következő operátorokat:

- elem hozzáadása a halmazhoz a += operatorral
- összes elem törlése (clear)
- annak eldöntése, hogy egy adott elem benne van-e a halmazban (isElement)

Az osztály legyen átadható érték szerint, és működjön helyesen a többszörös értékadás is!
Sorolja fel, hogy a generikus adatra vonatkozóan mely tagfüggvények meglétét tételezi fel a megvalósítás!
Hozzon létre egy olyan halmazt, amely max. 10 *double* elemet tartalmazhat! **Specializálja** ennek a halmaznak az isElement tagfüggvényét úgy, hogy két elemet akkor tekintsen azonosnak, ha a különbségük abszolútértéke kisebb, mint 1e-10!

Egy lehetséges megoldás:

```
template <class T, int size, class E = std::range_error>
class Halmaz {
    T t[size];
    int n;
public:
    Halmaz() : n(0) {}
    bool isElement(const T& e) const {
        for (int i = 0; i < n; i++) {
            if (e == t[i])
                return true;
        }
        return false;
    }
    Halmaz& operator+=(const T& e) {
        if (!isElement(e)) {
            if (n >= size)
               throw E("Halmaz::operator+=");
            t[n++] = e;
        }
        return *this;
    }
    void clear() {
        n = 0;
    }
};
```

Generikus adat tagfüggvényei: `Konstruktor/destruktor, op=, op==`

2 db char elemet tartalmazó vektor: `Halmaz<double, 10> v3;`

Specializáció:

```
template <>
bool Halmaz<double, 10>::isElement(const double& e) const {
    for (int i = 0; i < n; i++) {
        if (fabs(e - t[i]) < 1e-10)
            return true;
    }
    return false;

}
```

A specializációnak csak a formális helyességét értékeltük, nem az algoritmust.

3. Feladat Σ 3 pont

Tételezze fel, hogy a második feladatban elkészített sablon jól működik! **Ennek felhasználásával készítsen** C++ nyelven egy olyan generikus függvényt (*szamol*), amely megszámolja, hogy a paraméterként kapott, két előre haladó iterátorral megadott sorozatban, hány olyan elem van, ami benne van a szintén paraméterként kapott halmazban! Ha a feladatot jól oldja meg, akkor az alábbi programrészletet lefuttatva a cnt változó értéke 3 lesz.

```
Halmaz<int, 1170> h1;
h1 += 1;
h1 += 8;
int t[] = { 1, 2, 3, 8, 1 };
int cnt = szamol(t, t+5, h1);
```

Egy lehetséges megoldás:

```
template<class Iter, class H>
int szamol(Iter elso, Iter utolso, const H &halm) {
    int db = 0;
    while (elso != utolso) {
        if (halm.isElement(*elso++))
            db++;
    }
    return db;
}
```

4. Feladat Σ 4 pont

Írjon C++ programot, amely a standard inputról fájl végéig beolvasott szövegben előforduló szavakat ábécé sorrendben kiírja a standard outputra! Minden szó mellé írja ki, hogy hányszor fordult elő a szövegben. A kis- és nagybetűket tekintse azonosnak. Feltételezheti, hogy a szövegben az angol ábécé betűin kívül csak whitespace (szóköz, tabulátor, újsorjel) karakterek fordulnak elő. A megoldáshoz **használhat** STL tárolókat és algoritmusokat.

Egy lehetséges megoldás:

```
#include <iostream>
#include <string>
#include <algorithm>
#include <cctype>
#include <map>

using namespace std;

int main() {
    map<string, int> cnt;

    string word;
    while (cin >> word) {
        transform(word.begin(), word.end(), word.begin(), tolower);
        cnt[word]++;
    }
    for (map<string, int>::iterator i1 = cnt.begin(); i1 != cnt.end(); i1++)
        cout << i1->first << ": " << i1->second << endl;
    return 0;

}
```

117. AA Megoldó Alfréd AA 117.

5. Feladat Σ 5 pont

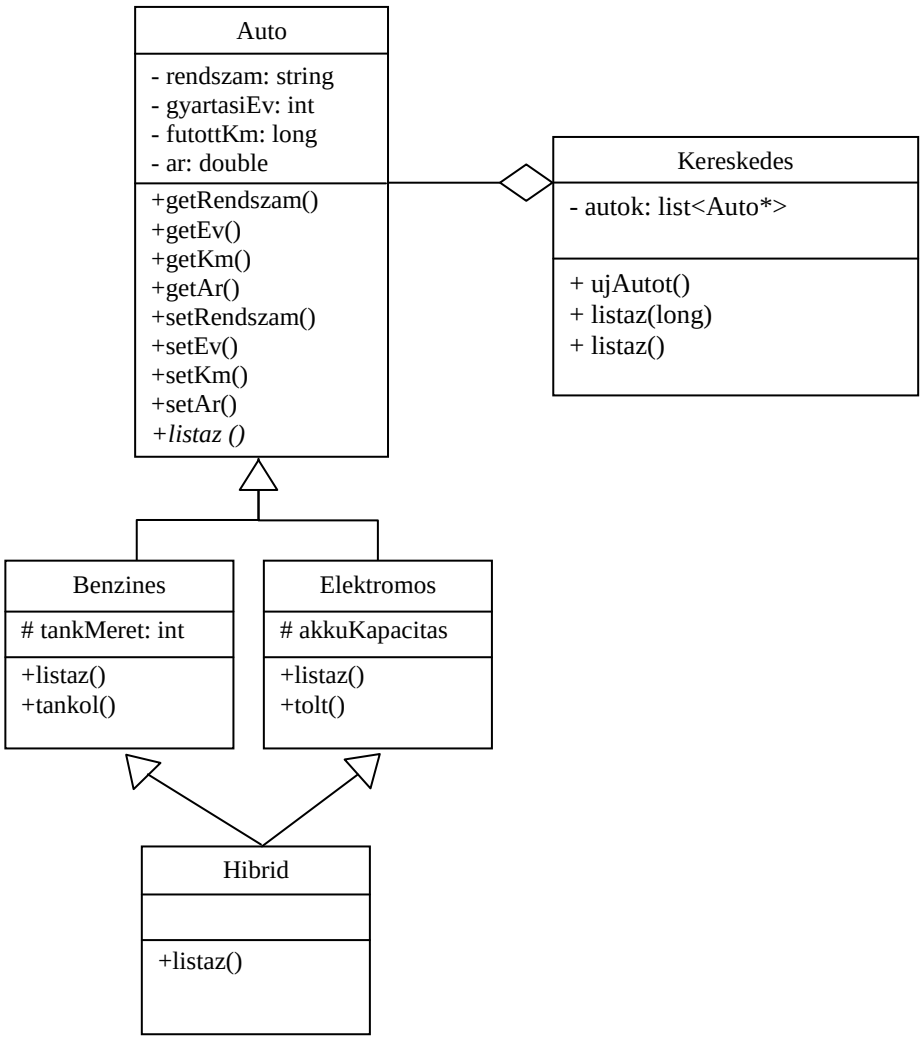
Egy használtautó kereskedést (*Kereskedes*) szeretnénk számítógépes információs rendszerrel segíteni. A kereskedésben árult autóknak (*Auto*) számos közös alapadata van. Ezek a következők: rendszám (*string*), gyártási év (*int*), futott kilométerek (*long*) és az ár (*double*). A meghajtás szempontjából jelenleg 3 fajtát kínálnak: benzinest (*Benzines*), teljesen elektromos meghajtásút (*Elektromos*), és vegyes üzeműt (*Hibrid*). A benzines autónak fontos jellemzője az üzemanyagtartály mérete (*int*), míg az elektromos autónak az akkumulátorok kapacitása (*double*). A hibrid autónak üzemanyagtartálya és akkumulátora is van. Üzemanyagot a benzines és a hibrid is a benzinkútnál tankol (*tankol*), de a hibrid – az elektromoshoz hasonlóan – tölteni (*tölt*) is tud. A megtervezendő információs rendszerben egyetlen tárolóban szeretnénk nyilvántartani az autókat. A rendszernek a következő funkciókat kell ellátni:

- Autó felvétele a nyilvántartásba (*ujAuto*).
- Autó törlése a nyilvántartásból rendszám alapján (*eladas*).
- A kereskedőnél kapható összes autó adatának listázása a standard outputra (*listaz*).
- Adott kilométertől kevesebbet futott, a kereskedőnél kapható összes autó adatának listázása a standard outputra (*listaz*).

Feladatok:

- **Tervezzen** egy OO modellt a problémának megfelelően, amelyben van *Kereskedes*, valamint *Benzines*, *Elektromos* és *Hibrid* autó. Legyen a modell könnyen bővíthető újabb autófélével (pl. Diesel), valamint közös és egyedi attribútumokkal (pl. szín, extrák, megtehető távolság, katalizátor). Rajzolja fel a modell osztálydiagramját! Használja a dőlt betűkkel megadott neveket!
- **Deklarálja** az objektumokat C++ nyelven!
- **Implementálja** (valósítsa meg) az objektumok összes tagfügvényét, ami az összes autó adatának kilistázásához kell!

A megoldáshoz **használhat** STL tárolókat, algoritmusokat is!



```
class Auto {
    string rendszam;
    int gyartasiEv;
    long futottKm;
    double ar;
public:
    string getRendszam() const;
    int getEv() const;
    long getKm() const;
    double getAr() const;
    void setRendszam(string);
    void setEv(int);
    void setKm(long);
    void setAr(double);
    virtual void listaz() const = 0;
    virtual ~Auto() {}
};
```

```
class Benzines :virtual public Auto {
protected:
    int tankMeret;
public:
    void listaz() const;
    void tankol();
};

class Elektromos :virtual public Auto {
protected:
    double akkuKapacitas;
public:
    void listaz() const;
    void tolt();
};

class Hibrid :public Benzines, public
Elektromos {
public:
    void listaz() const;
};
```

```
class Kereskedes {
    list<Auto*> autok;
public:
    void ujAuto(Auto *a);
    Auto *eladas(string rendszam);
    void listaz(long);
    void listaz() {
        for (std::list<Auto*>::iterator i = autok.begin(); i != autok.end(); i++)
            (*i)->listaz();
    }
    ~Kereskedes();
};

void Auto::listaz() const {
    cout << rendszam <<" "<< gyartasiEv << " " <<  futottKm << "km" << ar << endl;
}

void Benzines::listaz() const {
    cout << "Benzines:" << endl;
    Auto::listaz();
    cout << tankMeret << endl;
}

void Elektromos::listaz() const {
    cout << "Elektromos:" << endl;
    Auto::listaz();
    cout << akkuKapacitas << endl;
}

void Hibrid::listaz() const {
    cout << "Hibrid: " << endl;
    Auto::listaz();
    cout << tankMeret << endl;
    cout << akkuKapacitas << endl;
}
```