500. CC Megoldó Alfréd CC 500.

Programozás alapjai 2. PótZH	2010.05.20. gyakorlat: G1/IB.146	Hiány:3 ZH:7,71
MEGOLD	**SCH/500.**	**Hftest: 0 ZHp:**

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, mobiltelefon nem használható.*

A feladatokat ***figyelmesen olvassa*** *el, megoldásukhoz* ***ne használjon fel STL*** *tárolót, kivéve, ha 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 6*0.5=3 pont

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

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

class Adat {
  int adat;
public:
  Adat(int i = 0) :adat(i)          { cout << adat << 'k';}
  Adat(const Adat&) :adat(4)        { cout << adat << 'c';}
  Adat& operator=(const Adat& a)  { cout << a.adat << 'e'; return *this;}
  ~Adat()                           { cout << 'd'; }
};

class Ma {
  Adat a;
  const char *s;
public:
  Ma(const char *s = "C", int al = 0)
      :s(s), a(al)                  { cout << s << "Mk"; }
  ~Ma()                             { cout << "Md"; }
};

int main() {
  Ma me6("MEGOLD", 6);    cout << endl;
  Ma b1 = me6;            cout << endl;
  b1 = me6;               cout << endl;
  Ma* p = new Ma[1];      cout << endl;
  delete[] p;             cout << endl;
  return(0);
}
```

6	k	M	E	G	O	L	D	M	k					
4	c													
6	e													
0	k	C	M	k										
M	d	d												
M	d	d	M	d	d									

2. Feladat Σ 3 pont

Deklaráljon C++ nyelven egy olyan osztályt (*Person*), ami személyi adtatok (név, születési év) tárolására alkalmas! Tételezze fel, hogy létezik a *Text* osztály, amit nem változtathat meg, amely tetszőleges hosszú karakterláncok tárolására alkalmas. A név tárolásához használja a *Text* típust! A *Person* osztály minden adata legyen külön-külön lekérdezhető és beállítható, de kívülről közvetlenül ne legyen elérhető! Legyen a destruktor virtuális! **Valósítsa** meg az osztály tagfüggvényeit! **Használja** a dőlt betűvel szedett azonosítókat! **Működjön** az elvárásoknak megfelelően az alábbi kódrészlet! (2p) **Legyen** az osztálynak egy ”kisebb, mint” operátora (operator<), amivel össze lehet hasonlítani két *Person*-t a koruk alapján! (1p)

```
Person s1, s2 = Person("Jancsi", 1980), s3("Juliska", 1985);
s1 = s2;
if (s3 < s2) cout << s3.getNev() << "a fiatalabb";
else if (s2 < s3) cout << s2.getNev() << "a fiatalabb";
else cout << "azonos koruak";
```

```
class Person {
    Text nev;
    int szulIdo;
public:
    Person(const char *n = "", int sz = 0) : nev(n), szulIdo(sz) {}
    void setNev(const Text & n) { nev = n; }
    void setSzul(int ev) { szulIdo = ev; }
    Text getNev() const { return nev; }
    int getSzul() const { return szulIdo; }
    bool operator<(const Person& sz) const {
        return szulIdo > sz.szulIdo;
    }
    virtual ~Person() {}
};
```

3. Feladat Σ 5 pont

A 2. feladatban felhasznált *Text* osztályt az alábbi sablonnal valósítottuk meg:

```
template <class T = char>
class myText {
typedef T char_typ;
    char_typ *str;  // pointer a stringre (lezáró nullával)
    int len;        // hossz, amibe nem számoljuk bele a lezáró nullát
public:
    explicit myText (const char_typ* s = 0);
    myText (const myText&);
    myText& operator=(const myText&);
    myText& operator+=(char_typ); // a string végéhez fűz egy karaktert
    myText& operator+=(const myText&); // a végéhez fűz egy stringet
    bool operator<(const myText&) const;
    operator const char_typ*() { return str; }
    ~myText () { delete [] str;}
};
template <class T>
```

```
myText<T>::myText (const char_typ* s) :len(0) {
    if (s != 0)
       for (const char_typ *t = s; *t; t++) len++;  // meghatározzuk a hosszát
    str = new char_typ[len+1];                      // len+1 helyet foglalunk
    if (s != 0)
       for (char_typ *t = str; *s; s++) *t++ = *s;  // átmásoljuk
    str[len] = 0;                                   // lezáró nulla
}
```

Valósítsa meg a sablon *másoló konstruktorát*, az *értékadó operátorát*, és az *operator+=(char_typ)* tagfüggvényét! (4p) **Definiáljon** egy *wText* típust, ami széles karakterekből álló karaktersorozat tárolására alkalmas! **Deklaráljon** ezzel a típussal egy *vezeteknevem* azonosítójú példányt, amit a saját *vezetéknevével* inicializál! (1p)

```
template <class T>
myText<T>::myText (const myText& s) {
    str = 0;
    *this = s;
}
template <class T>
myText<T>& myText<T>::operator=(const myText& s) {
    if (this != &s) {
        delete [] str;
        len = s.len;
        str = new char_typ[len+1];
        for (int i = 0; i <= len; i++) str[i] = s.str[i];
    }
    return *this;
}
template <class T>
myText<T>& myText<T>::operator+=(char_typ c) {
    char_typ *tmp = new char_typ[len+2];
    for (int i = 0; i < len; i++) tmp[i] = str[i];
    delete[] str;
    str = tmp;
    str[len++] = c;
    str[len] = 0;
    return *this;
}
Másik Csoportnak:
template <class T>
myText<T>& myText<T>::operator+=(const myText& s){
    char_typ *tmp = new char_typ[len+s.len+1];
    int i = 0;
    for (; i < len; i++) tmp[i] = str[i];
    for (int j = 0; j < s.len; j++) tmp[i++] = s.str[j];
    delete[] str;
    str = tmp;
    len = i;
    str[len] = 0;
    return *this;
}
```

```
typedef myText<wchar_t> wString;
wString vezeteknevem(L"Megodldo");
```

4. Feladat Σ 4 pont

Az STL generikus prioritási sorának (*priority_queue*) felhasználásával írjon programrészletet, amely először névsorrendben, majd koruk sorrendjében kiírja azok nevét, akiket egy iterátorral rendelkező tárolóban (*varolista*) tartunk nyilván. A tárolóban *Person* (lásd 2. feladat) típusú objektumokat tárolunk. Ha szükséges, készítsen predikátumot (függvényobjektum) is!

Segítségül megadjuk az *std::priority_queue* rövid leírását:

```
template <class T, class Container = vector<T>, class Compare = less<T> >
class priority_queue;
```

Az osztály tagfüggvényei a következők:

- **def. konstruktor**: létrehoz egy üres *Container* típusú tárolót és egy *Compare* típusú predikátumot
- **konstruktor két iterátorral**: mint a default, majd a tárolót feltölti az iterátorok által meghatározott elemekkel
- **bool empty()**: visszaadja, hogy üres-e a sor
- **int size()**: visszaadja a sorban tárolt elemek számát
- **const T& top()**:sor legnagyobb prioritású elemének referenciáját adja
- **void pop()**: törli a sor legnagyobb prioritású elemét
- **void push(const T& t)**: betesz egy új elemet a sorba

```
priority_queue<Person > q(varolista.begin(), varolista.end());
while(q.size()) {
    cout << q.top().getNev() << endl;
    q.pop();
}

struct comp {
    bool operator()(const Person& sz1, const Person& sz2) const {
        return sz1.getNev() < sz2.getNev();
    }
};

priority_queue<Person, vector<Person>, comp>
                        q2(varolista.begin(), varolista.end());
while(q2.size()) {
    cout << q2.top().getNev() << endl;
    q2.pop();
}
```

5. Feladat: A feladat megoldásához **használhatja az STL** elemeit! Σ 5 pont

Három független nyilvántartási rendszer adatait szeretnénk integrálni úgy, hogy az bővíthető legyen, és ne tároljunk benne többszörösen (redundánsan) adatokat. Jelenleg a magánszemélyeket (*Person*) a lakossági adatbázisban (*Lako*), az olimpikonokat (*Olimpikon*) a Magyar Olimpiai Bizottság (*MOB*), az egyetemi hallgatókat (*Hallgato*) a Neptun (*Neptun*) adatbázisában tartják nyilván. Természetesen lehet olyan olimpikon, aki egyben hallgató is.
A feladatanalízis során a szereplők attribútumait az alábbiakban határoztuk meg:

500. CC Megoldó Alfréd CC 500.

Person (*Person*)
- név (Text)
- születési év (egész)

Olimpikon (*Olimpikon*)
- név (Text)
- születési év (egész)
- legjobb olimpiai helyezés (egész)

MOB, Neptun, Lako
- tárolók

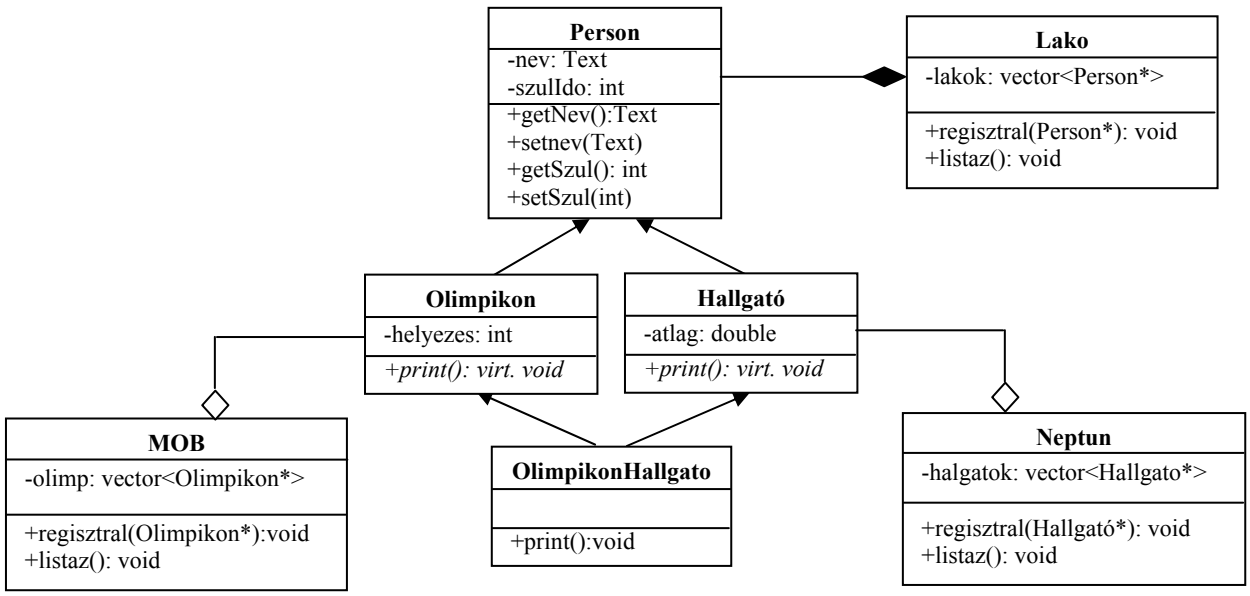
Hallgató (*Hallgato*)
- név (Text)
- születési év (egész)
- görgetett átlag (valós)

Olimpikon hallgató *(OlimpikonHallgato)*
- név (Text)
- születési év (egész)
- görgetett átlag (valós)
- legjobb olimpiai helyezés (egész)

Kezdeti modellünk csak minimális funkciókkal rendelkezik:
- objektumok létrehozása (minden attribútum legyen megadható a konstruktorban)
- objektumok megszüntetése
- új adat felvétele a megfelelő adatbázisba (*regisztral()*)
- az olimpikonok és a hallgatók névsorának kiírása a standard kimenetre (*MOB::listaz(), Neptun::listaz()*). A névsor soronként egy nevet tartalmazzon! Azon olimpikonok neve mellett, akik egyben hallgatók is, jelenjen meg az „olimpikon és hallgató” szöveg.

Tervezzen objektummodellt a feladat megoldására. **Osztálydiagrammal** mutassa be az egyes osztályok attribútumait, és kapcsolatát, melyen jelölje a tagváltozók és tagfüggvények láthatóságát is! (1p)
Deklarálja C++ nyelven a *MOB*, az *Olimpikon*, a *Hallgato* és az *OlimpikonHallgato* osztályokat! 2. feladatban készített *Person* osztályt felhasználhatja, azt nem kell újra deklarálnia! (2p)
Valósítsa meg a következő tagfüggvényeket: *MOB::regisztral(), OlimpikonHallgato* konstruktorát, és az osztályok azon metódusát, ami szükséges a *MOB* névsor elkészítéséhez (*MOB::listaz()*)! (2p) Működjön az alábbi kódrészlet!

```
Lako lakosok;     MOB mobadat;          Neptun neptunadat;
lakosok.regisztral(new Szemely("Jancsi", 1950));
Olimpikon* op = new Olimpikon("juliska", 1954, 30);
lakosok.regisztral(op);
mobadat.regisztral(op);
OlimpikonHallgato* oph = new OlimpikonHallgato("Cseh Laci",1987,2,4.13);
lakosok.regisztral(oph);
mobadat.regisztral(oph);
neptunadat.regisztral(oph);
mobadat.listaz();
```



```
class Olimpikon : virtual public Person {
    int helyezes;
public:
    Olimpikon(Text nev, int kor, int hely)
                                    : Person(nev, kor), helyezes(hely) {}
    virtual void print() { cout << getNev() << endl; }
};
class Hallgato : virtual public Person {
    double atlag;
public:
    Hallgato(Text nev, int kor, double atl)
                                    : Person(nev, kor), atlag(atl) {}
    virtual void print() { cout << getNev() << endl; }
};
class OlimpikonHallgato : public Hallgato, public Olimpikon  {
public:
    OlimpikonHallgato(Text nev, int kor, int hely, double atl)
     : Person(nev, kor), Olimpikon("", 0, hely), Hallgato("", 0, atl) {}
    void print() {cout << getNev() << " olimpikon es hallgato" << endl; }
};
class MOB {
    vector<Olimpikon*> olimpikonok;
public:
    void regisztral(Olimpikon* op) { olimpikonok.push_back(op); }
    void listaz();
};

void MOB::listaz(){
    for (size_t i = 0; i < olimpikonok.size(); i++) {
        olimpikonok[i]->print();
        cout << endl;
    }
}
```

Másik Csoportnak:

```
class Neptun {
    vector<Hallgato*> hallgatok;
public:
    void regisztral(Hallgato* op) { hallgatok.push_back(op); }
    void listaz();
};

void MOB::listaz(){
    for (size_t i = 0; i < hallgatok.size(); i++) {
        hallgatok[i]->print();
        cout << endl;
    }
}
```