500. AA Megoldó Alfréd AA 500.

Programozás alapjai 2. NZH	2010.05.13. gyakorlat: /	Hiány:0 ZH:0
	MEGOLD **IB.027/51.**	Hftest: **0**

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 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	6	
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 Alap {
  int alap;
public:
  Alap(int i = 0) :alap(i)          { cout << alap << 'k';}
  Alap(const Alap&) :alap(360)      { cout << alap << 'c';}
  Alap& operator=(const Alap& a)    { cout << a.alap << 'e'; return *this;}
  virtual ~Alap()                   { cout << 'd'; }
};

class Uj :public Alap {
  const char *s;
public:
  Uj(const char *s = "A", int al = 0)
      :s(s), Alap(al)               { cout << s << "Uk"; }
  Uj(const Uj& u) : s(u.s)          { cout << s << "Uc";}
  ~Uj()                             { cout << "Ud"; }
};

int main() {
  Uj me0("MEGOLD", 0);   cout << endl;
  Uj b1 = me0;           cout << endl;
  b1 = me0;              cout << endl;
  Alap* p = new Alap[2]; cout << endl;
  delete[] p;            cout << endl;
  return(0);
}
```

0	k	M	E	G	O	L	D	U	k					
0	k	M	E	G	O	L	D	U	c					
0	e													
0	k	0	k											
d	d													
U	d	d	U	d	d									

2. Feladat Σ 3 pont

Egy gyártósor működését szeretnénk modellezni. A gyártás során az elemi alkatrészekből (*Alkatresz*) egyre összetettebb alkatrészek (*OszetettAlkatresz*) készülnek, majd ezek felhasználásával eljutunk a végtermékig. A modellben heterogén gyűjteményként kívánjuk kezelni az összetett alkatrészeket, melyek maguk is alkatrészek. Az alkatrészek közös adata a cikkszám (tetszőleges karaktersorozat). **Deklarálja** az alaposztályt (*Alkatresz*) C++ nyelven úgy, hogy az később felhasználható legyen tetszőleges alkatrész, pl. *OsszetettAlkatresz* származtatására is! Az alaposztály tegye lehetővé a következő műveletek elvégzését:

- cikkszám megadása a konstruktorban.
- cikkszám(ok) kiíratása (*print*) a paraméterként kapott std::ostream típusú objektumra. Ügyeljen arra, hogy később a származtatással létrejövő összetett alkatrész minden elemi alkatrésze is kiírható legyen!
- Elemszám megállapítása (*elemszam*) (hány db elemi alkatrészből áll az adott alkatrész). Különböző alkatrészek esetén a megállapítás algoritmusa eltérhet, pl. az összetett alkatrész elemszáma az azt alkotó összes elemi alkatrész száma lesz!

Ügyeljen ara, hogy a származtatott osztályban a későbbiekben lesz dinamikus adat! **Valósítsa** meg az elemi alkatrész (*Alkatresz*) osztály tagfüggvényeit! **Használja** a dőlt betűvel szedett azonosítókat! A cikkszám tárolásához használja a *string* osztályt, ami képes tetszőleges hosszú karaktersorozatot tárolni! (2p) **Sorolja** fel a *string* osztály azon metódusait, melyet megoldásában felhasznált! **Működjön** az elvárásoknak megfelelően az alábbi kódrészlet! (1p)

```
Alkatresz csavarD = Alkatresz("12-456-89"), tmp("");
tmp = csavarD;
```

```
class Alkatresz {
protected:
    string cikkszám;
public:
    Alkatresz(string cikk) : cikkszám(cikk) {}
    virtual void print (ostream& os) { os << cikkszám << endl;}
    virtual int elemszam() { return 1; }
    virtual ~Alkatresz() {}
};
```

Nincs szükség külön másoló konstruktorra és operator= -re mert jó a default !

A megoldás használja a string osztály konstruktorait (másoló, const char*), destruktorát, operator= és az operator<<(ostream& string) globális operatort, ami nem tagfüggvény.

3. Feladat Σ 5 pont

A 2. feladat gyártási modelljében egy összetett alkatrész tetszőleges számú alkatrészből áll, melyek bármelyike lehet elemi vagy összetett (vagy másféle) is. A modellezési feladat folytatásaként **deklarálja** az összetett alkatrész (*OsszetettAlkatresz*) osztályt C++ nyelven! Az osztály tegye lehetővé a következő műveletek elvégzését:

- Összetett alkatrész cikkszámának megadása a konstruktorban.
- Új alkotóelem (alkatrész) hozzáadása az összetett alkatrészhez (*add*). Az új elemet az alkatrészjegyzék végére tegye!
- alkatrész törlése az alkatrészjegyzék végéről (*del*). A törlés során a törölt alkatrészt semmisítse meg!
- cikkszámok kiíratása (*print*) a paraméterként kapott std::ostream típusú objektumra. Az összetett alkatrész valamint az összes alkotóelem cikkszámának lista-szerű kiíratása.
- Elemszám megállapítása (*elemszam*) (hány db elemi alkatrészből áll az adott alkatrész).

Az osztály az alkotóelemeket úgy tárolja, hogy mindig pontosan annyi memóriaterületet foglaljon, ami éppen szükséges a tároláshoz (ne többet, ne kevesebbet)! Az objektumpéldány megszűnésével az alkotóobjektumok is szűnjenek meg! Oldja meg, hogy az *OsszetettAlkatresz* osztály ne legyen másolható és az értékadás operátora se legyen elérhető! Tételezze fel, hogy a 2. feladat *Alkatresz* osztálya létezik! **Használja** a dőlt betűvel szedett azonosítókat! **Valósítsa** meg a konstruktort, valamint a *print()* és *az add()* tagfüggvényeket!

```
class OsszetettAlkatresz : public Alkatresz {
    Alkatresz **tar;     // pointereket tárolunk dinamikus területen
    int darab;
    OsszetettAlkatresz(const OsszetettAlkatresz&);
    OsszetettAlkatresz& operator=(const OsszetettAlkatresz&);
public:
    OsszetettAlkatresz(string);
    void print(ostream&);
    int elemszam();
    void add(Alkatresz*);
    void del();
    ~OsszetettAlkatresz();
};
OsszetettAlkatresz::OsszetettAlkatresz(const char* cikk):Alkatresz(cikk){
    tar = new Alkatresz*[darab = 0];
}
void OsszetettAlkatresz::print(ostream& os) {
    os << "osszetett (" << darab << "db): ";
    Alkatresz::print(cout);
    for (int i = 0; i < darab; i++)
        tar[i]->print(os);
}
void OsszetettAlkatresz::add(Alkatresz* resz) {
    Alkatresz **tmp = new Alkatresz*[darab + 1];
    for (int i = 0; i < darab; i++)
        tmp[i] = tar[i];
    tmp[darab++] = resz;
    delete[] tar;
    tar = tmp;
}
```

4. Feladat Σ 4 pont

Írjon függvénysablont (**my_for**), ami az STL *for_each* sablonjához hasonlóan (annak felhasználása nélkül) végiglépdel egy tároló elemein és minden elemen végrehajtja a paraméterként megadott függvényt ill. függvényobjektumot! A függvény két előrehaladó iterátort kap paraméterként, ami kijelöli a jobbról nyílt intervallum kezdetét és végét, továbbá paraméterként kap egy egyparaméterű végrehajtandó függvényt is. A *my_for függvény* void visszatérési értékű legyen! (2 pont) Amennyiben helyesen készíti el a függvénysablont, akkor az alábbi kódrészlet kiírja a standard kimenetre, hogy „Haho C++”.

```
char v[] = "Haho C++";
my_for (v, v+8, putchar);
```

Mutassa be a függvénysablon alkalmazását egy olyan feladatra, amelyben double típusú értékeket tartalmazó tárolóból (**store**) ki kell írni a szabványos kimenetre az összes adatot! Minden adatot külön sorba írjon! **Definiálja** a kiíráshoz szükséges függvényt, vagy függvényobjektumot! (1p). Tételezze fel, hogy a **store** objektumnak van iterátora, és létezik a szokásos *begin(), end()* tagfüggvénye! **Írja** meg a megfelelő programrészletet, ami a *my_for* sablon felhasználásával kiírja az összes adatot a **store** nevű tárolóból (1p)!

```
template <typename Iter, class Pred>
void my_for(Iter beg, Iter end, Pred pred) {
    while (beg != end)
        pred(*beg++);
}
```

```
struct Print{
    void operator()(double d) {
        cout << d << endl;
    }
};
  my_for(store.begin(),store.end(), Print());
```

Vagy függvénnyel megoldva:

```
void Printf(double d) {
        cout << d << endl;
};
  my_for(store.begin(),store.end(), Printf);
```

500. AA Megoldó Alfréd AA 500.

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

Egy egyetem (*Egyetem*) hallgatóit (*Hallgato*), oktatóit (*Oktato*), demonstrátorait (*Demonstrator*), egyszóval polgárait (*Polgar*) és azok kapcsolatát szeretnénk modellezni. A modell a későbbiekben bővülni fog, pl. a technikai dolgozókkal, ezért fontos, hogy könnyen bővíthető legyen. Fontos továbbá, hogy kerüljük az adatok redundáns (többszörös) tárolását. Így pl. ha valamelyik szereplő több szerepkörben jelenik meg, akkor a közös adatait (pl. név) csak egyszer tároljuk. A legjobb példa erre a demonstrátor, akinek oktatói és hallgatói szerepköre is van. A feladatanalízis során a szereplők attribútumait az alábbiakban határoztuk meg:

Egyetem (*Egyetem*)
- egyetem neve (string)
- egyetem polgárai

Oktató: (*Oktato*)
- név (string)
- fizetés/megbízási díj (double)

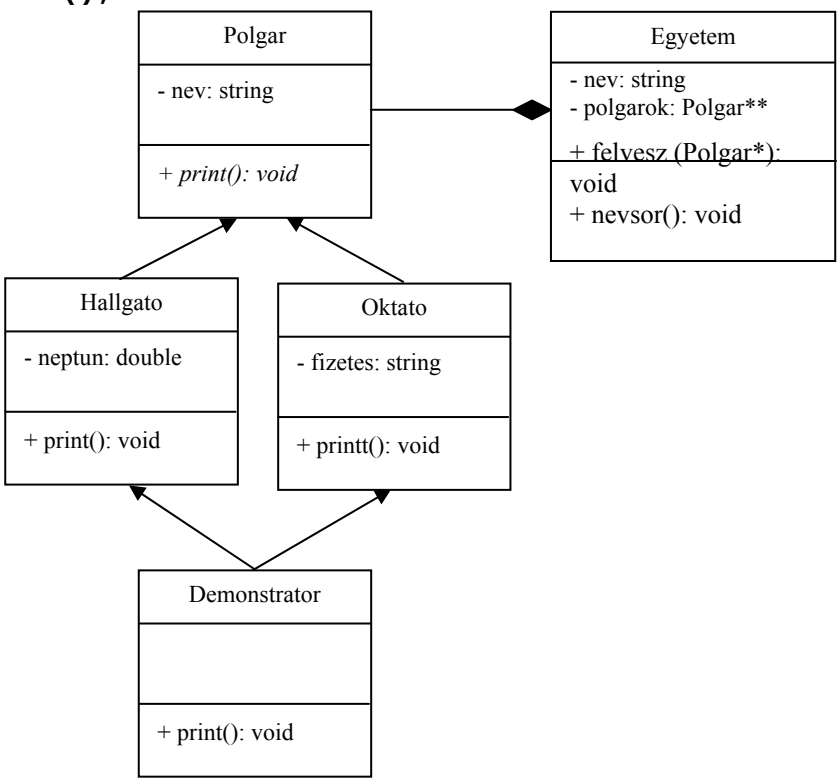
Hallgató (*Hallgato*)
- név (string)
- neptunkkód (string)

Demonstrátor (*Demonstrator*)
- név (string)
- neptunkkód (string)
- fizetés/megbízási díj (double)

Kezdeti modellünk csak minimális funkciókkal rendelkezik:
- objektumok létrehozása (minden attribútum - kivéve a lista jellegű elemeiket - legyen megadható a konstruktorban)
- objektumok megszüntetése
- polgár (oktató, hallgató, demonstrátor, stb) felvétele az egyetemre (*felvesz()*)
- egyetemi névsor kiírása a standard kimenetre (*nevsor()*). A névsorban soronként egy nevet tartalmazzon! hallgatók neve mellett jelenjen meg a „hallgató” a demonstrátorok neve mellet pedig „a demonstrátor 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 modellt megvalósító osztályokat! Használja a dőlt betűs azonosítókat! (2p)
Valósítsa meg a következő tagfüggvényeket: *felvesz(), Demonstrator* osztály konstruktora, és az osztályok azon metódusát, ami szükséges névsor elkészítéséhez (*nevsor()*)! (2p) Működjön helyesen az alábbi kódrészlet!

```
Egyetem bme("BME");
bme.felvesz(new Oktato("Prof. Nagyon Okos Edi", fizetes1));
bme.felvesz(new Oktato("Dr. Szoke Barna", fizetes2));
bme.felvesz(new Hallgato("Streber Szilard Sajo", KOD));
bme.felvesz(new Demonstrator("Doktor Andusz", masik_KOD, megbDij));
bme.nevsor();
```



500. AA Megoldó Alfréd AA 500.

A sárgával kiemelt részek helyett lehetne egy pontosvessző is, mert nem kellett megvalósítani.

```
class Polgar {
    string nev;
public:
    Polgar(string nev) : nev(nev) {}
    virtual void print() { cout << nev; }
    virtual ~Polgar() {}
};
class Oktato : virtual public Polgar {
    double fizetes;
public:
    Oktato(string nev, double fizu) : Polgar(nev), fizetes(fizu) {}
};
class Hallgato : virtual public Polgar {
    string neptun;
public:
    Hallgato(string nev, string nept) : Polgar(nev), neptun(nept) {}
    void print() { Polgar::print(); cout << " hallgato"; }
};
class Demonstrator : public Hallgato, public Oktato  {
public:
    Demonstrator(string nev, string nept, double fizu)
                      : Polgar(nev), Hallgato("", nept), Oktato("", fizu)
{}
    void print() { Polgar::print(); cout << " demonstrator hallgato"; }
};
class Egyetem {
    string nev;
    vector<Polgar*> polgarok;
public:
    Egyetem(string nev) :nev(nev) {}
    void felvesz(Polgar* polg) { polgarok.push_back(polg); }
    void nevsor();
    ~Egyetem();
};

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