# 128. AA Megoldó Alfréd AA 128.

| Programozás alapjai 2. | PótZH | 2010.05.20. gyakorlat: | Hiány:0 | ZH: |
|---|---|---|---|---|
| | | **MEGOLD** **SEHOL/5.** | **Hftest: 0**<br>**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 | 4 | |
| 3 | 5 | |
| 4 | 4 | |
| 5 | 4 | |
| **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 = 9) :adat(i)            { cout << adat << 'k';}
  Adat(const Adat&) :adat(2)          { cout << adat << 'c';}
  Adat& operator=(const Adat& a)      { cout << a.adat << 'e'; return *this;}
  ~Adat()                             { cout << 'd'; }
};

class Ma {
  Adat *ap;
  const char *s;
  Ma(const Ma&);
  Ma& operator=(const Ma&);
public:
  Ma(const char *s = "A", int i = 0)
      :s(s)                           { ap = new Adat(i); cout << s << "Mk";}
  ~Ma()                               { delete ap; cout << "Md"; }
};
int main() {
  Ma me6("MEGOLD", 6);      cout << endl;
  Adat a1, a2(6);           cout << endl;
  a1 = a2;                  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 | | | | | |
| 9 | k | 6 | k | | | | | | | | | | | |
| 6 | e | | | | | | | | | | | | | |
| 0 | k | A | M | k | | | | | | | | | | |
| d | M | d | | | | | | | | | | | | |
| d | d | d | M | d | | | | | | | | | | |

2. Feladat Σ 4 pont

Tervezzen generikus dinamikus tömb (*Tomb*) osztályt! A tömb mérete a konstruktorban legyen megadható. Az osztály:

- Legyen képes tetszőleges típusú elemeket tárolni!
- Legyen indexelhető a szokásos index operátorral ([ ]). Ha a kapott index nagyobb vagy egyenlő, mint a tömb mérete, akkor dobjon *std::range_error* kivételt! Ügyeljen arra, hogy az index operátorral elért elem balértékként is szerepelhessen!
- Legyen átméretezhető a *resize*() tagfüggvénnyel! Amennyiben az új méret nagyobb mint a régi, akkor a keletkező elemek a tárolt elemtípus alapértelmezett értékével (default konstruktor) legyenek feltöltve!
- Legyen lekérdezhető a tömb aktuális mérete (*size()*)!
- Legyen átadható érték szerint függvényparaméterként!
- Kezelje helyesen a többszörös értékadást (t1=t2=t3, ahol t1, t2, t3 ugyanazzal a típussal, de nem feltétlen azonos mérettel létrehozott objektum példányok)!

# 128. AA Megoldó Alfréd AA 128.

**Készítse el** a sablont C++ nyelven úgy, hogy a tagfüggvényeket ne valósítsa meg (ne definiálja)! (1.5p) **Valósítsa** meg (definiálja) a következőket: *alapértelmezett* és *egyparaméteres konstruktor*, *destruktor*, *operator[]*, *resize* (2.5p)

Egy lehetséges megoldás:

```
template <class T>
class Tomb {
    T* adatp;               // pointer az adatok tömbjére
    unsigned int siz;       // ennyi adatunk van
public:
    Tomb(unsigned int siz = 0);
    Tomb(const Tomb& t);    // nem kellett megvalósítani
    unsigned int size();    // nem kellett megvalósítani
    void resize(unsigned int ns);
    T& operator[](unsigned int i);
    Tomb& operator=(const Tomb& t); // nem kellett megvalósítani
    ~Tomb();
};
template <class T>
Tomb<T>::Tomb(unsigned int siz) : siz(siz) {
    adatp = new T[siz];
    for (unsigned int i = 0; i < siz; i++)
        adatp[i] = T();     // alapértelmezett értékkel töltjük fel
}
template <class T>
Tomb<T>::~Tomb() { delete[] adatp; }
template <class T>
void Tomb<T>::resize(unsigned int ns) {
    if (ns != siz) {
        T* tmp = new T[ns];
        unsigned int i = 0;
        unsigned int j = siz < ns ? siz : ns;
        for (; i < j; i++) tmp[i] = adatp[i];
        while (i < ns) tmp[i++] = T();// alapértelmezett értékkel töltjük fel
        delete[] adatp;
        adatp = tmp;
        siz = ns;
    }
}
template <class T>
T& Tomb<T>::operator[](unsigned int i) {
    if (i < siz) return adatp[i];
    throw range_error("Indexelesi hiba");
}
```

**Másik csoportnak resize és [] helyett:**

```
template <class T>
Tomb<T>::Tomb(const Tomb& t) {
    adatp = 0;              // visszavezetjük az értékadásra
    *thist = t;
}
template <class T>
Tomb<T>& Tomb<T>::operator=(const Tomb& t) {
    if (thist != &t) {
        delete[] datap;
        siz = t.siz;
        adatp = new T[siz];
        for (unsigned int i = 0; i < siz; i++) adatp[i] = t.adatp[i];
    }
    retrun *this;
}
```

3. Feladat Σ 5 pont

Tételezze fel, hogy a 2. feladat *Tomb* sablonja hiánytalanul rendelkezésre áll. Készítsen a sablon felhasználásával *int* elemeket tároló halmaz (Halmaz) osztályt! Az osztálynak legyen előre haladó iterátora, amivel végig lehet járni a halmaz elemeit! Az osztály tegye lehetővé az alábbi műveleteket:

- elem beszúrása a halmazba (*berak()*);
- annak eldöntése, hogy egy elem benne van-e a halmazban (*eleme_e()*);
- halmaz számosságának lekérdezése (*meret()*);
- szokásos iterátor műveletek (*, ++, !=, *begin()*, *end()*). (csak pre incremens)

Az osztály kezelje helyesen a többszörös értékadást és legyen átadható a halmaz függvényparaméterként. Segítségül megadjuk az osztály hiányos deklarációját. Megoldását a pontozott helyekre írja (több helyet hagytunk)!

A pontok helyén a megoldás látható sárga kiemeléssel:

```
class Halmaz {
    Tomb<int> adat;
public:
    class iterator {         // a halmaz iterátora
        Halmaz* hpoi6;       // pointer a halmazra (az iterátor erre vontakozik)
        unsigned akt_index; // erre az elemre „mutat” az iterátor
    public:
        iterator(Halmaz* h = 0, unsigned i = 0) : hpoi6(h), akt_index(i) {}

        iterator& operator++() {
            akt_index++;
            return *this;
        }

        int& operator*() {

            return hpoi6->adat[akt_index];
        }

        bool operator!=(const iterator &it6) {
            return hpoi6 != it6.hpoi6 || akt_index != it6.akt_index;
        }
    };

    iterator begin() {
        return iterator(this);
    }

    iterator end() {
         return iterator(this, meret());
    }

    // halmaz tagfüggvényei
    void berak(int e) {
        if (eleme_e(e)) return;
        adat.resize(adat.size() + 1);
        adat[adat.size() - 1] = e;
    }

    bool eleme_e(int e) {
        for (unsigned i = 0; i < adat.size(); i++)
            if (adat[i] == e) return true;
        return false;
    }

    unsigned int meret() {
        return adat.size();
    }
};
```

4. Feladat Σ 4 pont

**Készítsen** egy olyan függvénysablont (*valaszt*), amely képes kiválasztani egy tárolóból a paraméterként kapott szempontnak leginkább megfelelő (pl. minimális) elemet. A függvénysablon a szokásos módon függvényparaméterként kap két iterátort, és egy predikátum objektumot, vagy függvényt, ami két elemről eldönti, hogy melyik felel meg jobban az adott kiválasztási szempontnak! A visszatérési érték a kiválasztott elem legyen! (2p)
Működjön az elvárásoknak megfelelően a következő kódrészlet!

```
double v[] = { 1.2, 4.5, 6.7, -3.6, -2 };
cout << valaszt<double>(v, v+5, less<double>()) << endl;
```

**Mutassa** be az elkészített sablon működését a 3. feladat Halmaz osztályából létrehozott objektumpéldányra! Tételezze fel, hogy a 3. feladat Halmaz osztálya hiánytalanuk elkészült! (0.5p)

**Készítsen** egy olyan generikus predikátumobjektumot, vagy függvényt, amellyel az abszolútértékek minimuma kiválasztható (1p)! Sorolja fel, hogy mely operatorok meglétét tételezte fel ez utóbbi részfeladatban (0.5p) !

Egy lehetséges megoldás:

```
template <class T, class I, class S>
T valaszt(I elso, I utolso, S pred) {
    T tmp;                          // kell egy temp
    if (elso != utolso)             // lehet, hogy üres az intervallum
        tmp =*++elso;               // elsőt eltesszük
    while (elso != utolso) {
        if (pred(*elso, tmp))       // predikátum mondja meg, hogy eltegyük-e
            tmp = *elso;
        ++elso;
    }
    return tmp;                     // a kiválasztott
}
```

```
Halmaz h1;
cout << valaszt<int>(h1.begin(), h1.end(), less<int>()) << end;
```

```
template <class T>
struct Abs_minimum {
     bool operator()(T t1, T t2) {
          if (t1 < 0) t1 = -t1;
          if (t2 < 0) t2 = -t2;
          return t1 < t2;
     }
};
```

5. Feladat: Σ 4 pont

Egy egyetem (*Egyetem*) polgárait (*Polgar*), tanszékeit (*Tanszek*), oktatóit (*Oktato*), demonstrátorait (*Demonstrator*) és azok kapcsolatát szeretnénk modellezni. A demonstrátorok olyan oktatók, akik hallgatók is egyben. A feladatanalízis során a szereplők attribútumait az alábbiakban határoztuk meg:

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

**Tanszék**: (*Tanszek*)
- tanszék neve (string)
- tanszékvezető (Oktato)
- tanszéken oktatók (demonstrátor is)

**Polgár** (*Polgar*)
- név (string)

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

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

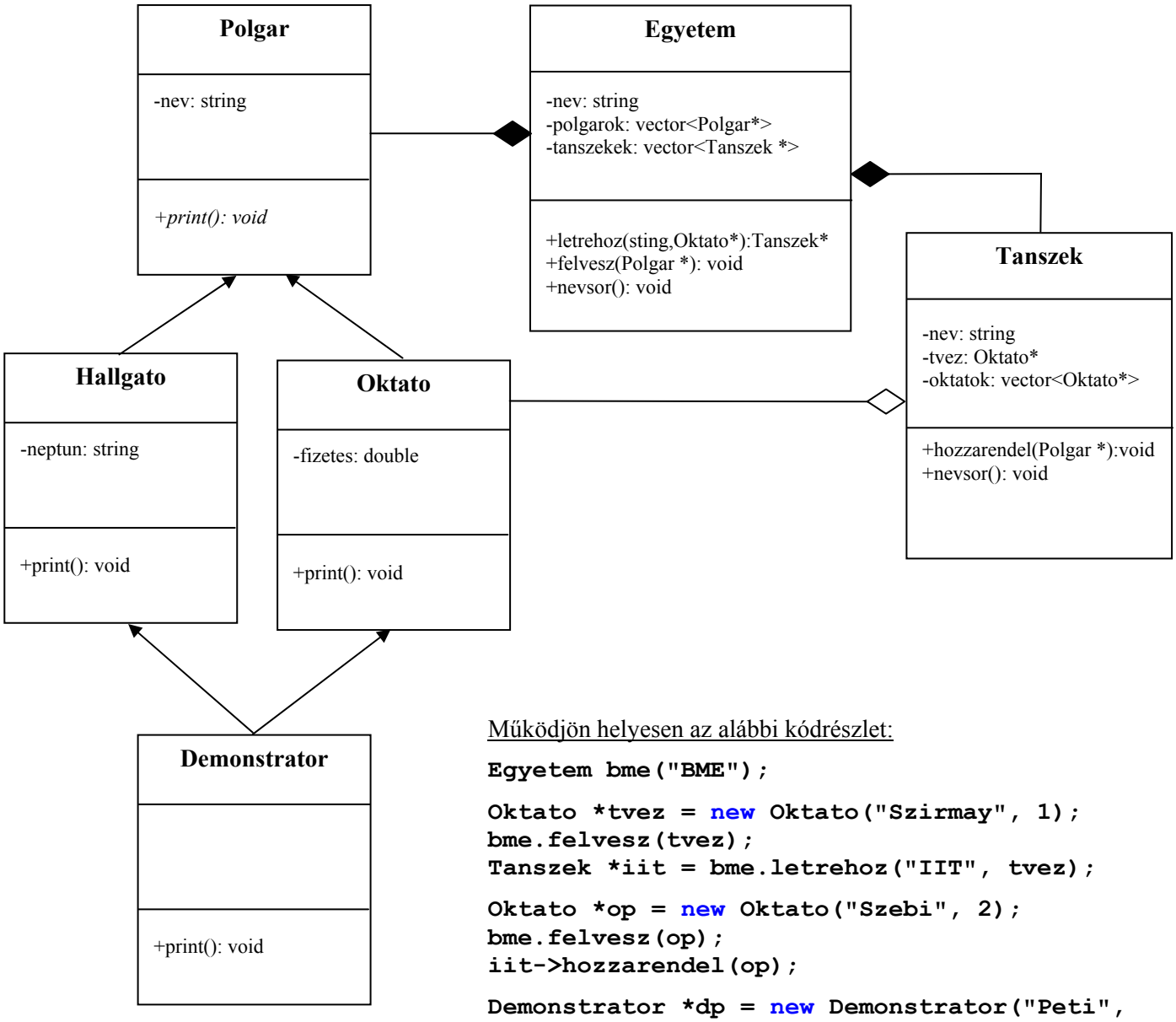
**Demonstrátor** (*Demonstrator*)
- név (string)
- neptunkód (string)
- fizetés/megbízási díj (double)

# 128. AA Megoldó Alfréd AA 128.

Egy oktató ill. demonstrátor több tanszéken is oktathat. Kezdeti modellünk csak minimális funkciókkal rendelkezik:

- objektumok létrehozása (minden attribútum - kivéve a listák elemei - legyen megadható a konstruktorban);
- objektumok megszüntetése;
- tanszék létrehozása az egyetemen belül (*letrehoz()*) (ld. kódrészlet);
- polgár felvétele az egyetemre (*felvesz()*);
- oktató ill. demonstrátor tanszékhez rendelése (*hozzarendel()*);
- polgár, oktató, hallgató és demonstrátor nevének kiírása a standard kimenetre (*print()*); (demonstrátorok esetében a név mellett a „demonstrator hallgato" szöveg is jelenjen meg)
- tanszéken oktatók ill. egyetem polgárainak névsora (*nevsor()*).

Az objektummodellt megterveztük, de sajnos az osztálydiagramból az összes felirat eltűnt! **Egészítse** ki a mellékelt osztálydiagramot! Jelölje a tagváltozók és tagfüggvények láthatóságát is! (1p)
**Deklarálja** C++ nyelven a modellt megvalósító osztályokat! Használja a dőlt betűs azonosítókat! (1.5p)
**Valósítsa** meg az *Egyetem* osztály *konstruktorát*, *destruktorát*, minden objektum *print()* tagfüggvényét, valamint a *Tanszek hozzarendel()* és *nevsor()* metódusát! (1.5p)
A megoldásban **használja fel** az *std::vector* vagy az *std::list* sablont és az *std::string* osztályt!




Működjön helyesen az alábbi kódrészlet:

```
Egyetem bme("BME");

Oktato *tvez = new Oktato("Szirmay", 1);
bme.felvesz(tvez);
Tanszek *iit = bme.letrehoz("IIT", tvez);

Oktato *op = new Oktato("Szebi", 2);
bme.felvesz(op);
iit->hozzarendel(op);

Demonstrator *dp = new Demonstrator("Peti",
                                    "ha1234", 3);
bme.felvesz(dp);
iit->hozzarendel(dp);

iit->nevsor();
bme.nevsor ();
```

Egy lehetséges megoldás:

```
class Polgar {
    string nev;
public:
    Polgar(string nev);
    virtual void print() { cout << nev; }   // virtuális!
    virtual ~Polgar();                      // virtuális!
};
class Oktato : virtual public Polgar {
    double fizetes;
public:
    Oktato(string nev, double fizu);
};
class Hallgato : virtual public Polgar {
    string neptun;
public:
    Hallgato(string nev, string nept);
};
class Demonstrator : public Hallgato, public Oktato  {
public:
    Demonstrator(string nev, string nept, double fizu);
    void print() { Polgar::print(); cout << " demonstrator hallgato"; }
};

class Tanszek {
    string nev;
    Oktato *tvez;
    vector<Oktato*> oktatok;
public:
    Tanszek(string nev, Oktato *tvez);
    void hozzarendel(Oktato* okt) { oktatok.push_back(okt); } // pointert vár!
    void nevsor();
};
void Tanszek::nevsor() {
    cout << "Tanszek: " << nev << "tanszekvezeto: ";
    tvez->print();  cout << endl;
    for (size_t i = 0; i < oktatok.size(); i++) {
        oktatok[i]->print(); cout << endl;
    }
}
class Egyetem {
    string nev;
    vector<Polgar*> polgarok;    // pointereket kell tárolni, mert heterogén
    vector<Tanszek*> tanszekek;  // itt nem lenne fontos a pointer, de...
public:
    Egyetem(string nev) : nev(nev) {}
    Tanszek* letrehoz(string nev, Oktato *tvez);// pointert ad vissza(ld. kód)
    void felvesz(Polgar* polg);  // pointert vár (ld. kód)
    void nevsor();
    ~Egyetem();
};

Egyetem::~Egyetem() {
    for (size_t i = 0; i < polgarok.size(); i++)
        delete polgarok[i];      // megszűnnek a polgárok is
    for (size_t i = 0; i < tanszekek.size(); i++)
        delete tanszekek[i];     // megszűnnek a tanszékek is
}
```