A Programozás Alapjai 2
Objektumorientált szoftverfejlesztés

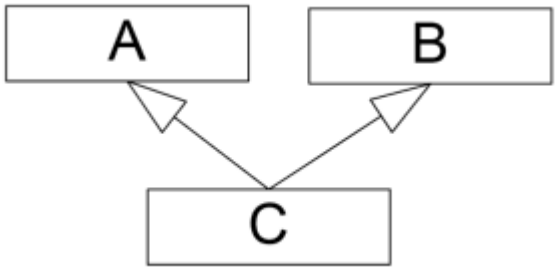
Dr. Forstner Bertalan
forstner.bertalan@aut.bme.hu

BME AUT
Department of Automation and Applied Informatics

Többszörös öröklés

Többszörös öröklés

- Egy adott osztálynak több ősosztálya van.
  - Több, mint kettő is lehet.
- Mindegyik tulajdonságait (attribútumok) és viselkedését (műveletek) örökli.



Két célja lehet

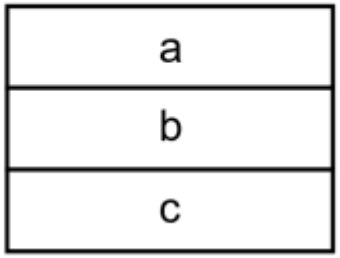
- Implementáció öröklése: Ritkán használják többszörös öröklésnél, ugyanis problémákat okozhat.
  - > Automatikusan rendelkezésre áll egyes függvények implementációja az őszosztályban
  - > Egy helyen kell csak módosítani változtatás esetén

Két célja lehet

- Implementáció öröklése: Ritkán használják többszörös öröklésnél, ugyanis problémákat okozhat.
  - > Automatikusan rendelkezésre áll egyes függvények implementációja az őszosztályban
  - > Egy helyen kell csak módosítani változtatás esetén
- Interfész öröklése: Az osztály műveletei az interfésze.
  - > Volt: Interfészt egy absztrakt osztállyal lehet készíteni.
  - > Ha több interfészt implementál egy osztály, mindből le kell származtatni.
  - > A leszármaztatás miatt behelyettesíthető mindnek a helyére.
  - > A különböző típusú objektumok egységesen kezelhetőek lesznek

Példa

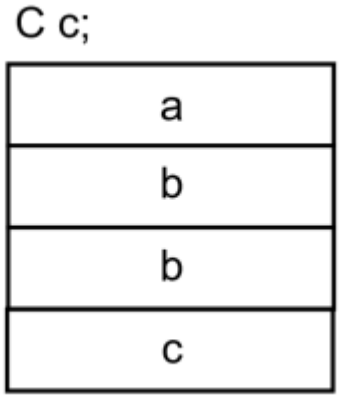
C c;



```
class A {
public:
   int a;
};

class B {
public:
   int b;
   void f() { cout << "B->f" << endl; }
};

class C : public A, public B {
  public:
   int c;
};
```

Példa

- És problémák



```
class A {
public:
   int a;
   int b;
   void f() { cout << "A->f" << endl; }
};

class B {
public:
   int b; //problema: neki is van b-je
   void f() { cout << "B->f" << endl; } //es f()-je
};
class C : public A, public B {
  public:
   int c;
};
```

```
C c;
//c.a = 5; hiba, ambiguous
c.A::a = 5;
c.B::f();
```

Interfészörökés példa

- Autóeladással és tanácsadással foglalkozó cég
- Két külső komponens:
  - Számlanyomtató
  - Könyvelési

A példa

Interfészörökés példa

```
class IWare {
public:
    virtual int getPrice() const = 0;
};

class IAccountable {
public:
    virtual int getVAT() const = 0;
};



void PrintPrice(const IWare& param)
{
    cout << "Price: " << param.getPrice();
}

void AccountingDoSomethingWithVAT(const IAccountable& param) {
    cout << "VAT = " << param.getVAT();
}
```

Elérés pointeren keresztül

- Gond lehet az elérés az ősosztály pointerén keresztül
  - hiszen a pointer típusa határozza meg az elvégezhető műveletet

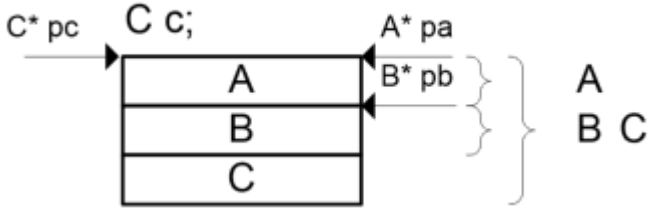
- Első **Példa** kiegészítése

Elérés pointeren keresztül

- Szabály1: a pointeren keresztül ***csak az adott osztálybeli rész*** érhető el
  - ez a műveletekre is igaz

Elérés pointeren keresztül

- Szabály1: a pointeren keresztül ***csak az adott osztálybeli rész*** érhető el
  - ez a műveletekre is igaz
- Szabály2: a pointer mindig ***a pointer típusának megfelelő rész elejére*** mutat.
  - Emiatt a példában a B*-ra castolás megváltoztatja a pointer értékét, el is tolja.
  - Ezt a fordító megoldja, ilyen intelligens a cast.

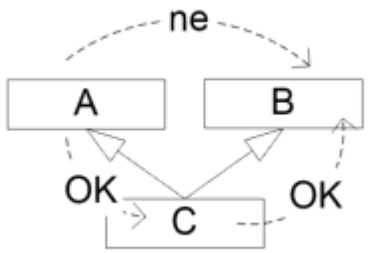


Keresztbe castolás

- Keresztbe ne castoljunk, mert akkor az eltologatás nem tud működni.
  - A kereszbe cast előtt castoljunk vissza fel a szülőre.
  - Keresztbe cast (pl. A*→B*) automatikusan nincs is (fordítói hiba), de explicit kiírva lefordul és hibás lesz.
- Példa

Keresztbe castolás

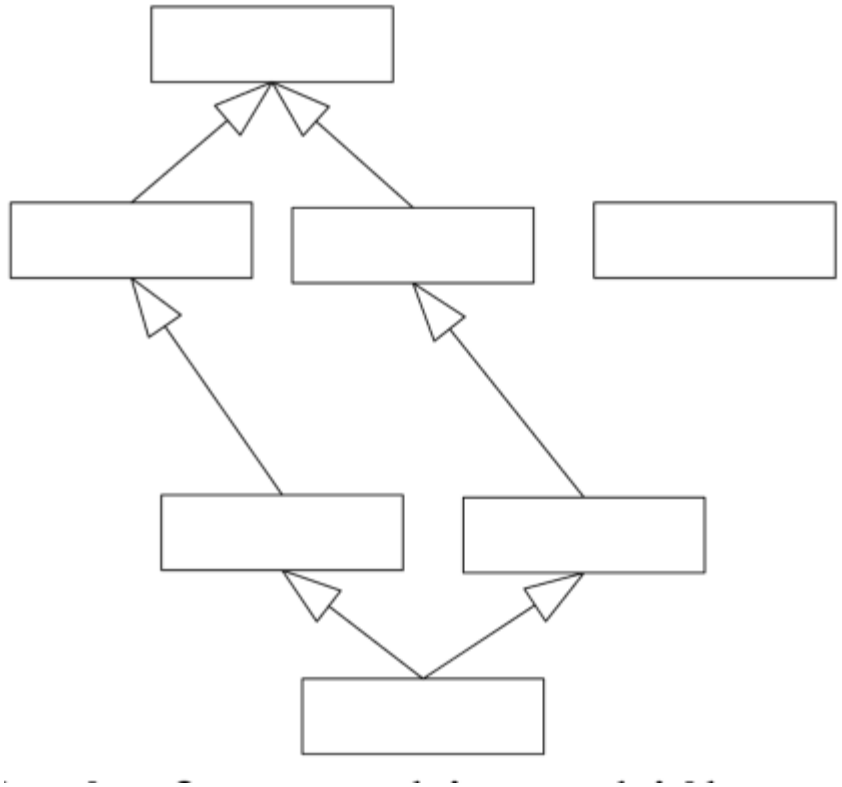
- Keresztbe ne castoljunk, mert akkor az eltologatás nem tud működni.
  - > A keresztbe cast előtt castoljunk vissza a leszármazottra.
  - > Keresztbe cast (pl. A*→B*) automatikusan nincs is (fordítói hiba), de explicit kiírva lefordul és hibás lesz.
- Példa



Miért veszélyes a void* ?

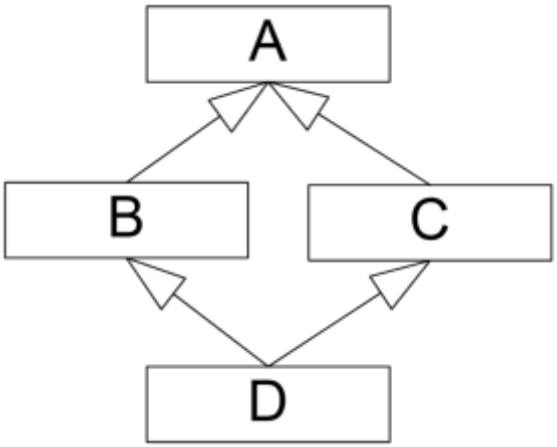
- Példa

Újabb probléma

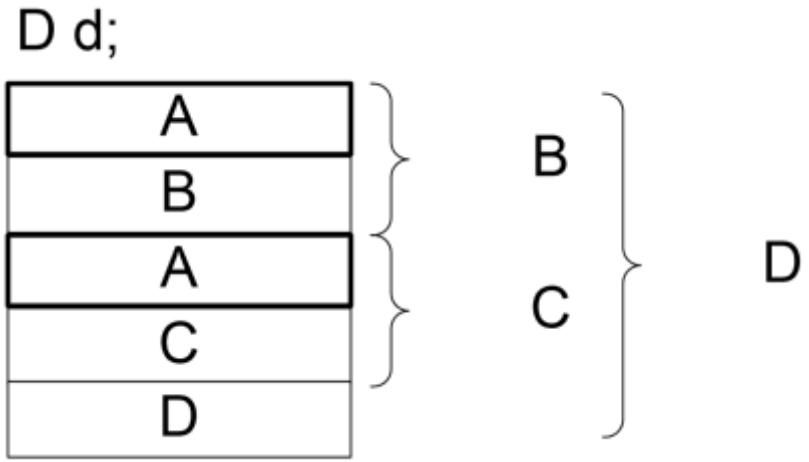


Gyémánt forma

- Példa

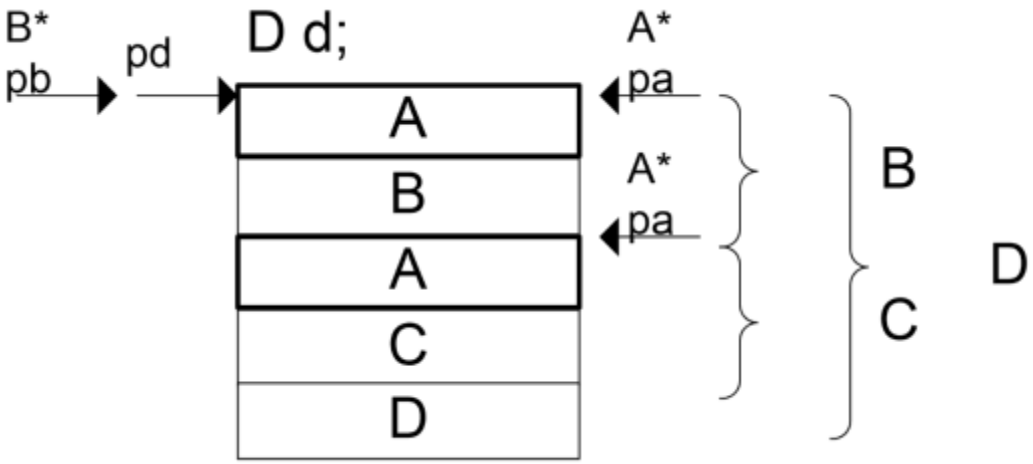


Memóriakép



```
class A {
public:
        int a;
        A() { a = 123; }
};

class B :public A {};
class C :public A {};
class D :public B, public C {
public:
        void print() { cout << B::a; } //egyertelmuve kell tenni

};

int main(int argc, char* argv[]) {
        D d;
        d.B::a = 5; //Itt is
        return 0;
}
```

Pointer konverzió



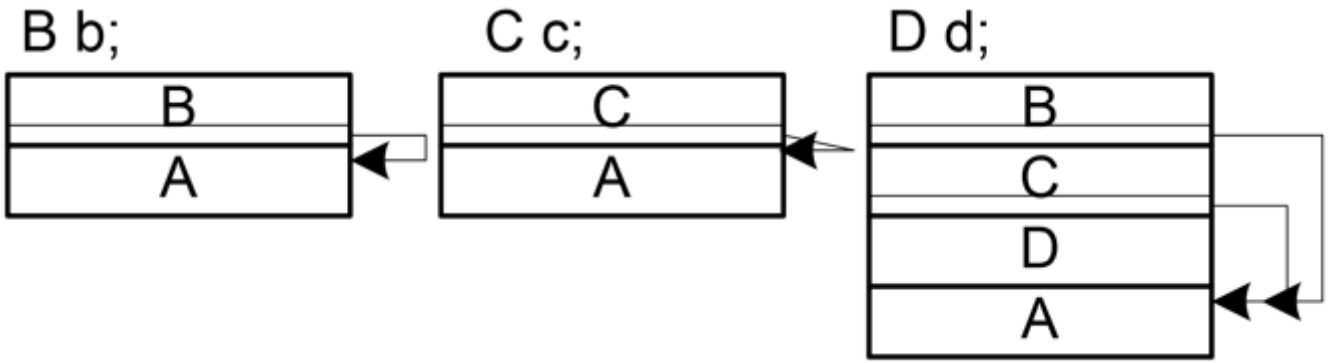
```
D d;
D* pd = &d; // OK, alapeset.
B* pb = &d; //Automatikus konverzio.
//Ugyanugy, mint pb=pd
C* pc = &d; //El kell tolni a pointert, hogy legyen ertelme

A* pa = &d; //gond, ketertelmu
pa = (B*)&d; //OK
```

Virtuális öröklés

- A két úton is érkező ős csak egyszer szerepeljen
- Semmi köze a **virtuális függvényekhez**
  - Azok sokkal fontosabbak.
- Szintaktika: az **öröklés leírása** helyére be kell írni a *virtual* kulcsszót.
- class B :public virtual A { …

Virtuális öröklés

- Egy lehetséges megvalósítás



Konstruktor hívási sorrend többszörös öröklésnél

- Virtuális ősosztályok konstruálása
- Nem-virtuális közvetlen ősosztályok konstruálása
- Saját részek konstruálása
  - Pointerek: virtuális ősosztály
  - Pointerek: Virtuális függvények táblája
  - Tagváltozók inicializálása
  - Konstruktortörzs lefutása

Destruktorok hívási sorrendje

- Ugyanez, csak vissza:

- Saját destruktortörzs meghívása
- Tagváltozók destruktora
- Közvetlen, nem-virtuális ősosztályok destruktora
- Virtuális ősosztályok destruktora

Konstrukciós sorrend

- Problémába futhatunk
- Példa: az alakzatok

Konstrukciós sorrend

- Problémába futhatunk
- Példa: az alakzatok

```
virtual void Print() { cout << "Shape" << endl; }
```

Tagváltozók inicializálásának sorrendje

- Nem az inicializálási listában felsorolt sorrend számít, hanem:
  - milyen sorrendben származtattunk le
  - milyen sorrendben definiáltuk a tagváltozókat
- Példa