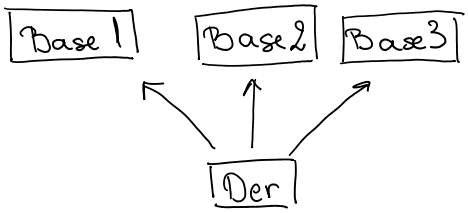
Тема 12

Tuesday, July 2, 2024 9:32 AM

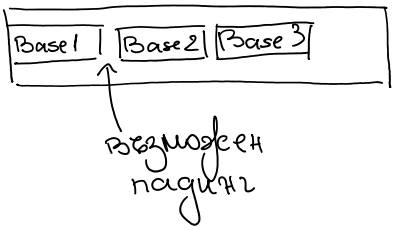
Тема 12

	Пример за фигури.
12	Множествено наследяване. Диамантен проблем. Колекции от обекти в полиморфна йерархия. Копиране и триене. Пример с ферма
13	Шаблони. Необходими функции в шаблонен клас/шаблонна функция. Темплейтни специализации

Множествено наследяване



Представяне на Der в паметта:



Можем да достъпваме Der през

```
Der * ptr = &d;
Base1 * ptr1 = &d; // сочи към началото на Der
Base2 * ptr2 = &d;  } използва специален механизъм, който изчислява колко
Base3 * ptr3 = &d;  } да се отмести ук-ля от началото на Der
```

```
&d == ptr1
ptr1 ≠ ptr2 ≠ ptr3
ptr1 ≠ ptr3
```

Конструктор на Der

- отговорен за конструктори на Base1, Base2, Base3

последователност на изпълнение на конструктори:

```
class Der : Base1, Base2, Base3 {

        A obj1;
        B obj2;

};
```

Base1(), Base2(), Base3(), A(), B(), Der()

Деструктор

~Der(), ~B(), ~A(), ~Base3(), ~Base2(), ~Base1()

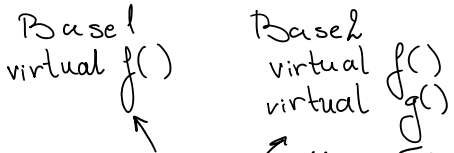
Оператор =

```
{ if (this != other)
```

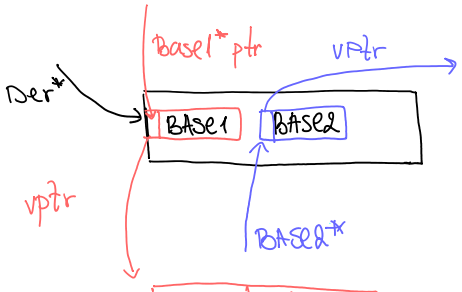
Оператор =

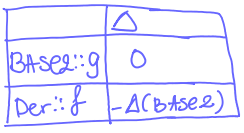
```
{ if (this != other)
  { Base1::op=(other);
    Base2::op=(other);
    Base2::op=(other);

    free();
    copyFrom(other);
  }
```

Виртуални таблици - по една за всеки родител



// трябва да презапише f, за да няма конфликт на имена





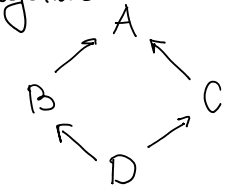
- Вирт. таблица на Der в контекста на Base2

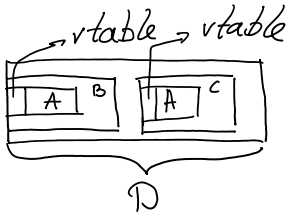
- Вирт. таблици на Der в контекста на Base1

Δ - vtable има параметър Δ, който ок зва с колко трябва да се отместят указателят, за да намери посочения обект (в байтове)

Диамантен проблем - при обикновено наследяване получаваме ≥2 инстанции на един и същи обект в наследника

- Понякога е желано поведение

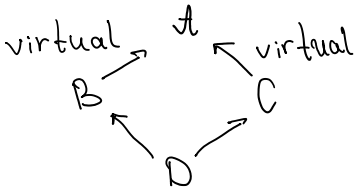




Конструктор A(), B(), A(), C(), D() - D не опазеля как е създава A
Деструктор ~D(), ~C(), ~A(), ~B(), ~A()

* Ако в A има променлива, която искаме да достъпим от D, или не се компилира (ambiguous), или ще се вземе тази от най-лявата инстанция (зависи от нивото на warning-а)

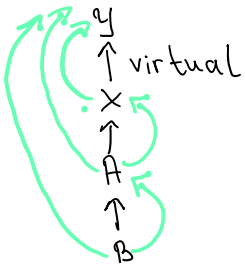
Виртуално наследяване - решение на диамантения проблем



-> Очакваме A да се споделя от повече от 1 наследник

- Всеки наследник на B и C (пряк или непряк) е отговорен за създаването на A

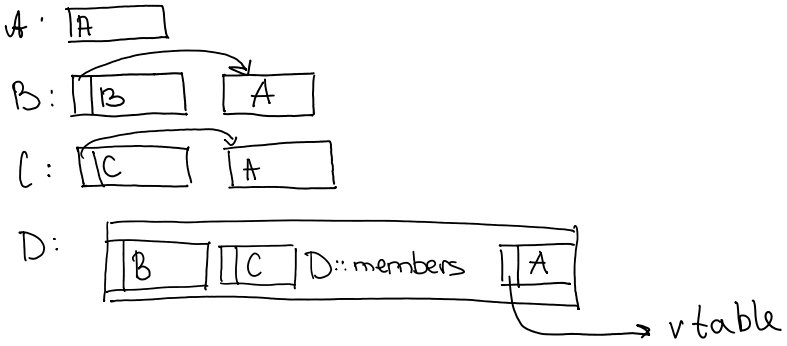
Пример:



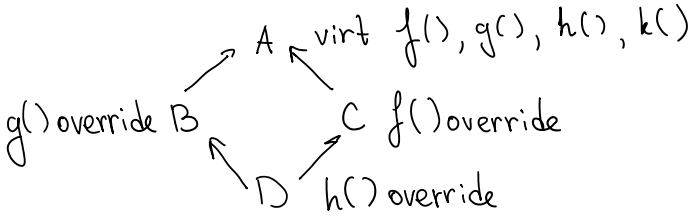
A указва кой констр. на X и на Y да се извика. Ако не се определи изрично, се извиква Y(), независимо от това, кой конструктор на Y се извиква от к-ра на X

→ - отговорен за създаването на

Представяне в паметта



По този начин D има само една вирту на таблица и една инстанция, което спестява памет.



	Δ
A::k()	0
B::g()	-Δ(A)
C::f()	-Δ(A) + Δ(C)
D::h()	-Δ(A)

Конструктор и деструктор

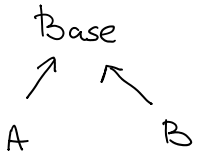
```
{
    D obj; // A(), B(), C(), D()
} // ~D(), ~C(), ~B(), ~A
```

Колекция от обекти с полиморфна йерархия

За да пазим полиморфни обекти като колекция, създаваме масив от указатели към базов клас,

За да го мениджира, създаваме wrapper-клас със свое копиране, триене и тн.

```
class Container {
 Base** data;
};
```




```
Base {
   virtual ~Base();
   virtual Base* clone() const =0;
};
```

```
Container::free() {
   for(i to n)
      delete data[i]; // извикват се правилните деструктори, тъй като ~Base е виртуален
   delete [] data;
}
```

```
Container::Container(const Container& other) {
    data = new Base* [ ... ];
    for(i to n) {
        data[i] = other.data[i] -> clone();
```

```
Base* A::clone () override {
        return new A (*this);
    }

Base* B::clone () override {
        return new B (*this);
    }
```