Тема 10

Tuesday, July 2, 2024 2:15 PM

Тема 10

10	Наследяване. Видове наследяване. Параметри на функции (указатели и референции). Конструктори и деструктори при наследяване. Копиране при наследяване. Move семантики при наследяване. Пример с човек, студент и преподавател.

Наследяване

def: Der е наследник на Base, ако разширява неговите данни и поведение

Разлика м/у наследяване и композиция : логическа:

комп-я: "has a" relationship
насл-не: "is a" relationship

Видове наследяване

- private (по default при класовете)
- protected
- public (по default при struct)

Достъпът до член-данни/функциите от наследника е минимума от модификатора за достъп за съответните член-данни/ф-я и вида наследяване

```
class X {
  private:
    int a;
  protected:
    int b;
  public:
    int c;
};
```

```
class A : public X {
  // всичко се наследява
  a - няма достъп
  b - protected
  c - public
};
```

```
class B: protected X {
  a - няма достъп
  b - protected
  c - protected
};
```

* Ако искаме публично наслед., но искаме да ограничим достъпа до конкретна ф-я

```
struct Base {
  f()
```

```
struct Base {
    f()
};

struct Der : public Base
{ private:
    using Base::f;
};
```

```
class C : private / E
{
    a-наследе достъп
    b - private
    c - private
};
```

Параметри на функции

```
f(Base b);
f(const Base *ptr);
f(const Base &b);
```
} Можем да подаваме и Der и Base, защото в наследено на Der стои Base

```
g(Der d);
g(Der* d);
g(Der& d);
```
} само Der

* f(Base* arr) - не можем да подадем масив от Der, защото имаме проблем, когато искаме да достъпим елементите (Base ук-ля няма да прескочи и памети, на които се съхраняват данните на Der)

Конструктори при наследяване

Конструкторът на наследника може да вика кой к-р на базовия клас да е извика. По подразбиране се извиква def.

```
class Der : Base
{
    A
    B
    C
};
```

Извикването винаги е в този ред, независимо от инициализ. списък

Деструктори при наследяване

Дестр. на наследника извиква дестр. на базовия клас

~Der() ~C() ~B() ~A() ~Base()

~Der() ~C() ~B() ~A() ~Base()

Копиране при наследяване

Der(const Der& other): Base(other) → Base(const Base& other)

{
  copyFrom(other); - дръпам се само за Der !
}

```
Der& op = (const Der& other)
{
    if(this != &other)
    {
      Base :: operator = other;
      free();
      copyFrom(other ;  > дръпам се само за Der !
    }
    return *this
}
```

Move семантиката при наследяване

```
Der(Der&& other ): Base(std::move (other))
{
  moveFrom( std::move (other )); //moveFrom( Der &&)
}

Der& op= (Der&& other)
    {
      if( ... )
    {
    Base:: op = (std :: move(other));
      free();
    moveFrom(std :: move other )) ;
    }
    return *this;
```

و