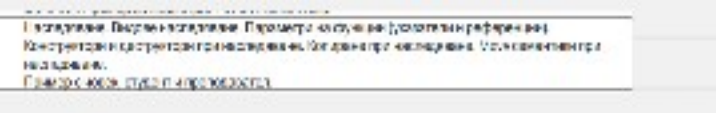
Тема 10



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 / функците от наследника е минимума от модификаторът за достъп за съответните член-данни / ф-е и видът наследяване

```
class X {
  private:
    int a;

  protected:
    int b;

  public:
    int c;

}
```

```
class A : public X {

    //всичко се наследява

    a - нямаме достъп
    b - protected
    c - public

}
```

```
class B: protected X {

    a - нямаме достъп
    b - protected
    c - protected

}
```

```
class C: private X {

    a - нямаме достъп
    b - private
    c - private

}
```

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

```
struct Base {
    f()
}

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

Параметри на функции

```
f( Base b );
f( const Base * ptr);
f( const Base & b );
```

Можем да подаваме и Der и Base, защото в началото 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 паметта, на която се съхраняват данните на Der)

Конструктори при наследяване

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

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

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

Деструктори при наследяване

Дестр. на наследника извиква дестр на базовия клас

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

Копиране при наследяване

→ Base(const Base & other)

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

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

```
Der &  op = ( const Der & other )
         {
           if(this != &other)
             {
              free( );
              copyFrom(other);
             Base :: operator = other;

             }
            return *this
          }
```

> грижат се само за Der !

Move семантики 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;

          }
```