Билет 3.18. Списки инициализации в конструкторах. Обязательная инициализация полей перед входом в конструктор. Особенности инициализации полей, являющихся ссылками и константами.

Def. При инициализации в области видимости конструктора возникают проблемы с инициализацией полей класса (при входе в область видимости поля уже проинициализированы по умолчанию и при "инициализации" в конструкторе на самом деле происходит копирование полей). Решить это можно с помощью *списков инициализации* в конструкторах.

Синтаксис:

```
struct Integer{
    int field;
    bool complex;
    Integer(int arg, bool complex): field(arg), complex(complex)
    {\\realization}
};
```

Правило : В списке инициализции инициализация происходит не в том порядке, в каком написаны члены списка, а в том, в каком порядке они объявлены как поля класса. Следовательно, в списке инициализии члены нужно перечилсять в таком же порядке, в каком они находятся в полях, иначе может возникнуть UB/CE. Нельзя совмещать списки инициализации и делегирование конструкторов - CE.

```
class A{
    const int& a;
public:
    A(const int& a): a(a){}
}

void f(){
    A a(5);
}
```

Это проблема, потому что 5 - rvalue, и ссыдка на него уничтожится, так что после инициализации будет битая ссылка.

Note. Так как при инициализации внутри конструктора на самом деле происходит копирование полей(это другой namespace), то поля-ссылки и поля-константы должны быть проинициализированы до входа в конструктор. То есть *при их объявлении, либо в списке инициализации.*

Билет 3.19. Понятие наследования, синтаксис объявления наследника, разница между private и public наследованием.

Def. *Наследование* — это одна из основных концепций ООП, позволяющая создавать классы на основе других классов, при этом заимствуя их функционал.

Def. При *приватном наследовании* только сам наследник внутри себя (и его друзья) знают о факте наследования, то есть имеют доступ к полям/методам родителя. А извне мы не можем через объект класса-наследника обратиться к чему угодно из класса-родителя.

Def. При *публичном* мы знаем извне, что данный класс — наследник, и потому можем обращаться к чему угодно, лежащему в классе-родителя (если оно public, конечно же).

Синтаксис наследования:

```
class Base {};
class PublicDerived : public Base {};
class ProtectedDerived : protected Base {};
class PrivateDerived : private Base {};
```

```
class Base {
    int b;
};

class Derived : public Base {
public:
    int a = 5;
    void f(int);
};
```

У класса Derived будут все поля класса Base, плюс свои поля и методы.

Друзья не наследуются. Приватные поля и методы не могут быть унаследованы. В C ++ конструкторы и деструкторы не наследуются. Однако они вызываются, когда дочерний класс инициализирует свой объект. То есть при создании наследника всегда создается родитель (со всеми полями и т.п.) Конструкторы вызываются один за другим иерархически, начиная с базового класса и заканчивая последним производным классом. Деструкторы вызываются в обратном порядке.

Сначала поиск метода производится в классе-потомке, а если его там нет, поиск поднимается на ступень выше.

Если в наследнике есть метод, который принимает объект родительского типа, то нельзя будет обратиться к его защищенным полям, но если принимается объект того же класса, что и сам наследник, то к защищенным полям можно обратиться.

```
class Base {
protected:
    int a;
};

class Derived : public Base {
public:
    int a = 5;
    void f(const Base& x) {
        std::cout << x.a; //doesn't work
    void g(const Derived& x) {
        std::cout << x.a; //will work
};
```

У структур по умолчанию наследование публичное, а у классов - приватное.