Язык C++. Лекция 7

Мещерин Илья

29 октября 2018 г.

5.2б) Явный вызов методов предка

```
struct Base{
    void f();
};

struct Derived : public Base{
    void f(int x);
};

int main(){
    Derived d;
    d.Base::f();
}
```

Явно пишем откуда взять нужную функцию. Если наследование приватное, то все равно ошибка компиляции.

```
struct Derived : public Base{
     using Base::f;
     void f(int x);
};
```

Второй способ решения вопроса. В таком случае вызывать функцию *f()* можно как обычно.

5.2в) Пример

Оффтоп - если при наследовании не написать тип наследования, то по умолчанию *private* (у *struct* он *public*)

```
struct Base{
public:
    void f();
};

struct Derived : public Base{
private:
    void f(int x);
};

int main(){
    Derived d;
    d.f();
}
```

Ошибка компиляции. **Проверка доступа происходит после поиска имен.**
5.2г) Пример

```
struct Granny{
   int a;
};

struct Mom : private Granny{
   int b;
};

struct Son : public Mom{
   int c;
   void f(Granny &g){

   }
}
```

Ошибка компиляции. Название типа *Granny* запрещено внутри класса *Son*.

```
struct Son : public Mom{
   int c;
   void f(:: Granny &g){

   }
}
```

А так уже писать можно, т.к. в глобальной области видимости запретов нет.

```
struct Granny{
   int a;
   friend struct Son;
};
```

Все равно ошибка компиляции, т.к. ошибка в выражении *s.Granny::a* возникает после точки, а не после двух двоеточий.

```
struct Mom : private Granny{
   int b;
   friend struct Son;
};
```

Такой способ уже решает проблему.
5.2д) Пример

```
struct Granny{
protected:
   int a;
};

struct Mom : public Granny{
   int b;
   friend void f();
};
```

```
void f(){
    Mom m;
    m.a;
}
```

Все ОК, но вообще отношение дружбы не транзитивно.

5.3) Порядок вызова конструкторов и деструкторов при наследовании