Правила доступа. Множественное наследование

Тема 7

protected – члены класса

Хорошо:

`protected`- функции

Задание операций для использования в производных классах

Плохо:

`protected`- данные

Приводят к проблемам сопровождения

protected члены класса

```
class Unit
{
protected:
    void Move() {}
};
```

```
class Rabbit :public Unit {
public:
    void MoveForGrass(){
        Move();}
};
```

```
int main()
{
    Rabbit r;
    r.MoveForGrass();
    r.Move();
}
```

Нет доступа к защищенным членам класса

Доступ к базовым классам

public

protected

private

public наследование

```
class Employee
{
public:
    Date GetFire()const;
protected:
    void Bonus(double sum);
};
```

```
class Manager: public Employee
{
    double f(){return 1000;}
public:
    bool SetBonus()
    {
        if(…)
            Bonus(f());
    }
};
```

```
class Supervisor: public Manager
{…};
```

```
void test(Employee * emp,
          Manager *mng,
          Supervisor* sup)
{
    sup->Bonus(5000);
    emp->GetFire();
    mng->GetFire();
    sup->GetFire();
    emp = mng;
    emp = sup;
    mng = sup;
    sup = emp;
    mng = emp;
    sup = mng;
}
```

private наследование

```
class A
{
public:
        void f() {}
        void g() {}
};
```

```
class B : private A
{

};
```

```
void fun(A * a)
{
        a->f();
}
```

```
int main() {
        A a;
        B b;
        a.g();
        b.f();
        fun(&a);
        fun(&b);
}
```

private наследование

```
class A
{
public:
        void f() {}
        void g() {}
};
```

```
class B : private A
{
public:
        A::f;
};
```

```
void fun(A * a)
{
        a->f();
}
```

```
int main() {
        A a;
        B b;
        a.g();
        b.f();
        fun(&a);
        fun(&b);
}
```

private наследование

```
class A
{
public:
        void f() {}
protected:
        void g() {}
};
```

```
class B : private A
{
public:
        A::g;
};
```

```
void fun(A * a) {
        a->f();
}
```

```
int main() {
        A a;
        B b;
        a.g();
        b.g();
        b.f();
        fun(&a);
        fun(&b);
}
```

private наследование

```
class A
{
public:
        void f() {}
protected:
        void g() {}
};
```

```
class B : private A {
public:
        void foo() {
            g();
            A *p = this;
        }
};
```

```
void fun(A * a) {
        a->f();
}
```

```
int main() {
        A a;
        B b;
        a.g();
        b.foo();
        b.f();
        fun(&a);
        fun(&b);
}
```

private наследование

```
class A
{
public:
    void f() {}
protected:
    void g() {}
};
```

```
class B : private A {
public:
        A::g;
};
```

```
class C :public B {
public:
    void foo(){
        f();
        g();
        B* p = this;
        p->g();
        p->f();
        A* pa = this;
        pa->f();
} };
```

```
int main() {
        A a;
        B b;
        C c;
        a.g();
        b.g();
        b.f();
        c.g();
}
```

protected наследование

```
class Elem {
public:
    Elem(int maxAge_) :age(0), maxAge(maxAge_) {}
    bool Live() {
        age++;
        if (IsDead())
            return false;
        return true;
    }
    bool Eat();
    bool Move() { ChangeX(); ChangeY(); return true; }
private:
    bool IsDead() {  return (age == maxAge); }
    int age, maxAge;
    int x, y;
    void ChangeX();
    void ChangeY();
};
```

protected наследование

```
class Rabbit : protected Elem
{
public:
    Rabbit(int maxAge_, double v_): Elem(maxAge_), v(v_)
    {}
    bool Live() {
        if (!Elem::Live())
            return false;
        Eat();
        Elem::Move();
    }
protected:
    bool Eat();
private:
    double v;
};
```

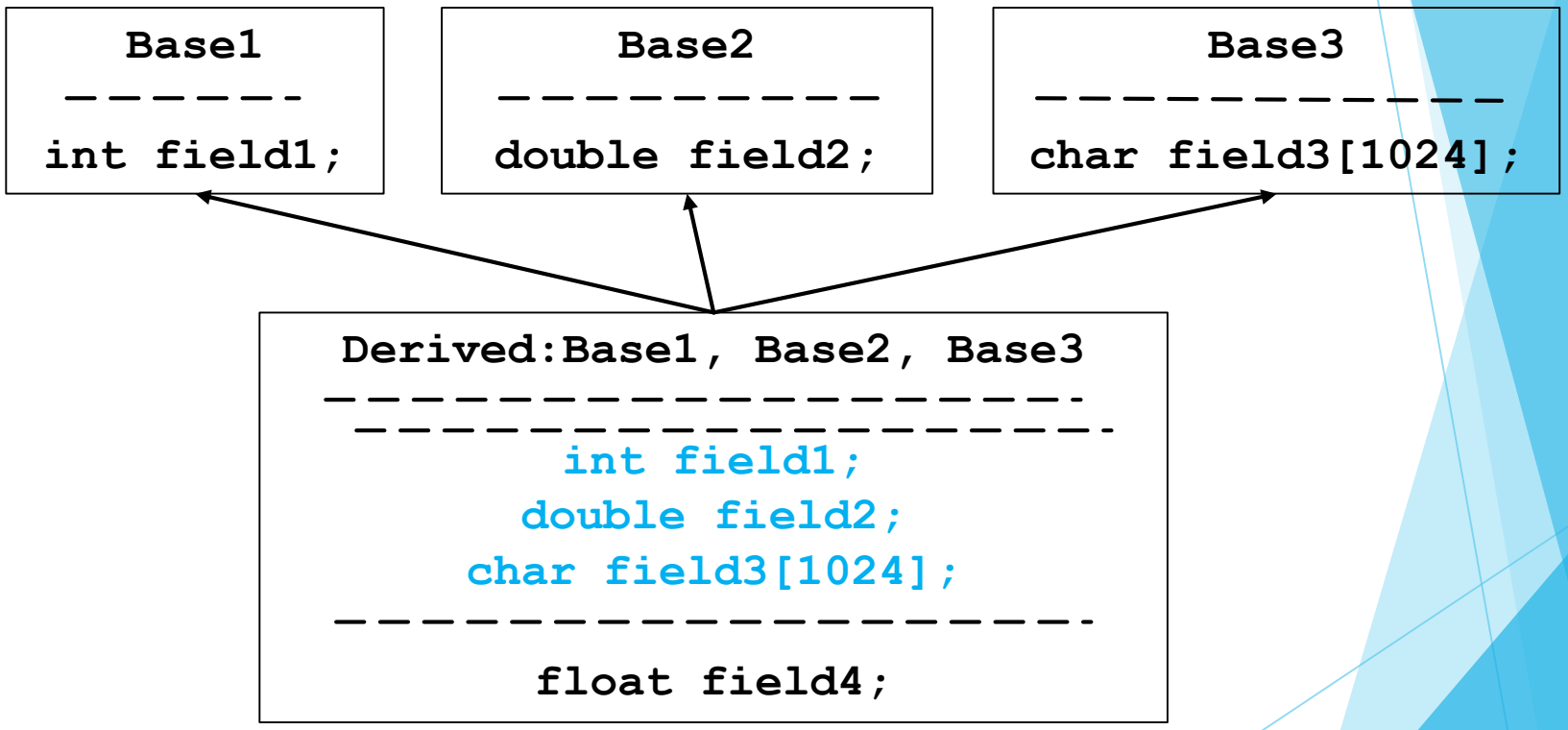
protected наследование

```
int main()
{
    Rabbit r(5, 27);
    r.Live();
    r.Move();
    r.Eat();
}
```

Множественное наследование -

класс является потомком более чем одного класса, то есть наследует элементы сразу нескольких классов

Иерархия классов при множественном наследовании



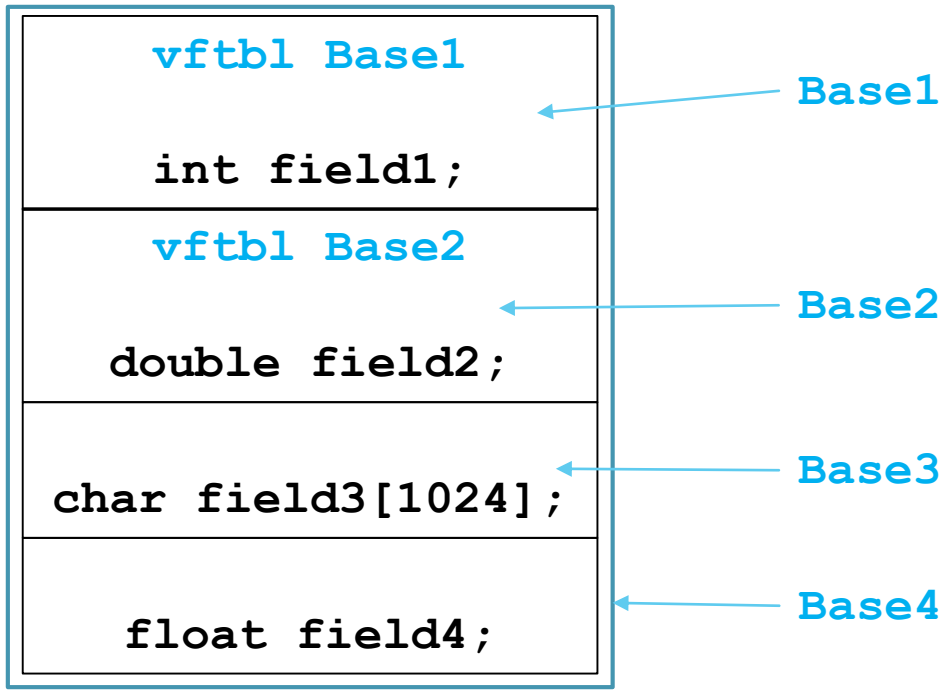
Объявления классов при множественном наследовании

```
class Base1
{
  int field1;
public:
  virtual void f()=0;
};
```

```
class Base1
{
  double field2;
public:
  virtual void g()=0;
};
```

```
class Base3
{
…
char field3[1024];
…
};
```

```
class Derived:
        public Base1, public Base2, public Base3
{
   float field4;
public:
   virtual void f();
   virtual void g();
};
```


vftbl Base1
int field1;
vftbl Base2
double field2;
char field3[1024];
float field4;
Base1
Base2
Base3
Base4

Конструкторы при множественном наследовании

Конструкторы базовых классов вызываются в той последовательности, в которой перечислены в заголовке производного класса, а не в последовательности, указанной в списке инициализаторов конструктора производного класса.

Проблемы множественного наследования

- Неоднозначность – совпадение идентификаторов в разных базовых классах

- Повторное включение кода - многократное вхождение одного класса в число предков другого класса

Множественное наследование

```
class Base1{
public:
        Base1(int x): value(x){}
        int GetData()const {return value;}
private:
        int value;
};
```

```
class Base2{
public:
        Base2(char c) : letter(c) {}
        char GetData()const {return letter;}
private:
        char letter;
};
```

Множественное наследование

```
class Child : public Base1, public Base2
{
public:
    Child (int i, char c, float f): Base1(i), Base2(c), real(f) {}
    float GetReal() const {return real;}
    void Print() const
    {
        std::cout << "Integer: "  << Base1::GetData() <<std::endl
                  << "Symbol: "   << Base2::GetData() <<std::endl
                  << "Real: "     << real             <<std::endl;
    }
private:
    float real;
};
```

Множественное наследование

```
int main(){
    Base1 b1(10), *base1Ptr = nullptr;
    Base2 b2('S'), *base2Ptr = nullptr;
    Child ch(7, 'D', 3.5);
    std::cout << b1.GetData()          <<std::endl;
    std::cout << b2.GetData()          <<std::endl;
    ch.Print();
    std::cout << ch.Base1::GetData()   << std::endl
              << ch.Base2::getData()   << std::endl
              << ch.getReal()          << std::endl;
    base1Ptr = &ch;
    std::cout << base1Ptr->getData()   << std::endl;
    base2Ptr = &ch;
    std::cout << base2Ptr->getData()   << std::endl;
    return 0;
}
```

Виртуальные базовые классы

Данные виртуальных базовых классов однократно входят в производный класс, независимо от того, сколько раз виртуальный базовый класс входит в граф наследования.

Порядок инициализации различных частей создаваемого класса в C++

1. Конструктор последнего производного класса вызывает конструкторы подобъектов виртуальных базовых классов. Инициализация виртуальных базовых классов выполняется в глубину, в порядке слева направо.

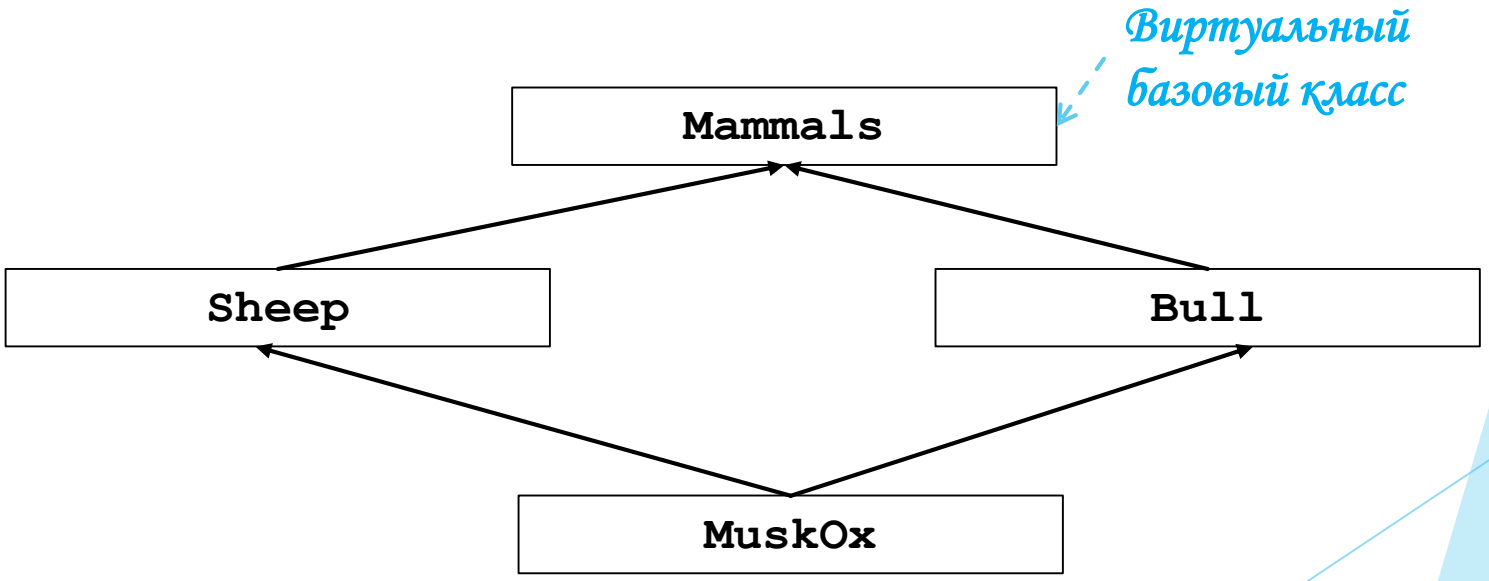
2. Конструируются подобъекты непосредственных базовых классов в порядке их объявления в определении класса.

3. Конструируются нестатические подобъекты-члены в порядке их объявления в определении класса.

4. Выполняется тело конструктора.

Ромбовидное наследование

Задача. Разработать иерархию классов млекопитающих. Абстрактный базовый класс – Млекопитающее. Его наследники – классы Бык и Овца. Наследник классов Бык и Овца – класс Овцебык



Класс Mammals

В базовом классе нет конструктора по умолчанию

```
class Mammals
{
public:
     Mammals(string _name):name(_name)  {
          std::cout << "Constructor mammals. Name: " << name << std::endl;
     }
     virtual ~Mammals()    {
          std::cout << "Destructor mammals. Name: "  << name << std::endl;
     }
     void Live() {
          Eat();
     }
protected:
     virtual void Eat() = 0;
private:
     string name;
};
```

Класс Sheep

Mammals – виртуальный базовый класс

```
class Sheep : virtual public Mammals {
public:
    Sheep(string _name, double _wool): Mammals(_name), wool(_wool) {
        std::cout << "Constructor sheep. Wool: " << wool << std::endl;
}
    virtual ~Sheep() {
        std::cout << "Destructor sheep. Wool: "  << wool << std::endl;
}
protected:
    virtual void Eat() {
        std::cout << "Sheep eat flowers. Wool became " << ++wool
                  << std::endl;
}
private:
    double wool;
};
```

Класс Bull

Mammals – виртуальный базовый класс

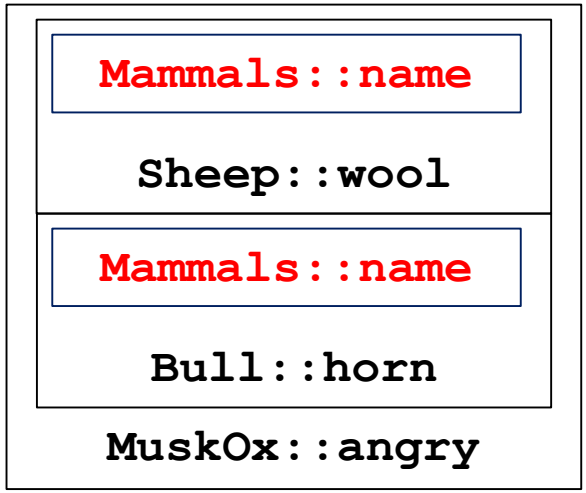
```
class Bull : virtual public Mammals
{
public:
     Bull(string _name, double _horn): Mammals(_name), horn(_horn) {
          std::cout << "Constructor bull. Horn: " << horn << std::endl;
     }
     virtual ~Bull() {
          std::cout << "Destructor bull. Horn: "  << horn << std::endl;
}
protected:
     virtual void Eat() {
          std::cout << "Bull eat grass. Horn became " << ++horn << std::endl;
     }
private:
     double horn;
};
```

Класс MuskOx

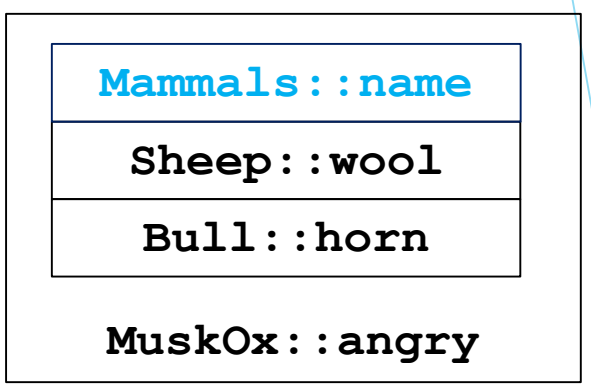
Необходимо явным образом вызвать конструктор Mammals

```
class MuskOx : public Bull, public Sheep
{
public:
     MuskOx(string _name, double _horn, double _wool, double _angry)  : Mammals(_name),
               Sheep(_name, _wool), Bull(_name, _horn), angry(_angry) {
          std::cout << "Constructor musk-ox. Angry: " << angry << std::endl;
     }
     virtual ~MuskOx() {
          std::cout << "Destructor musk-ox. Angry: "   << angry << std::endl;
     }
protected:
     virtual void Eat() {
          std::cout << "MuskOx eat grass and flowers. Angry became: " << --angry
                    << std::endl;
}
private:
     double angry;
};
```

При обычном наследовании:



При ромбовидном наследовании:

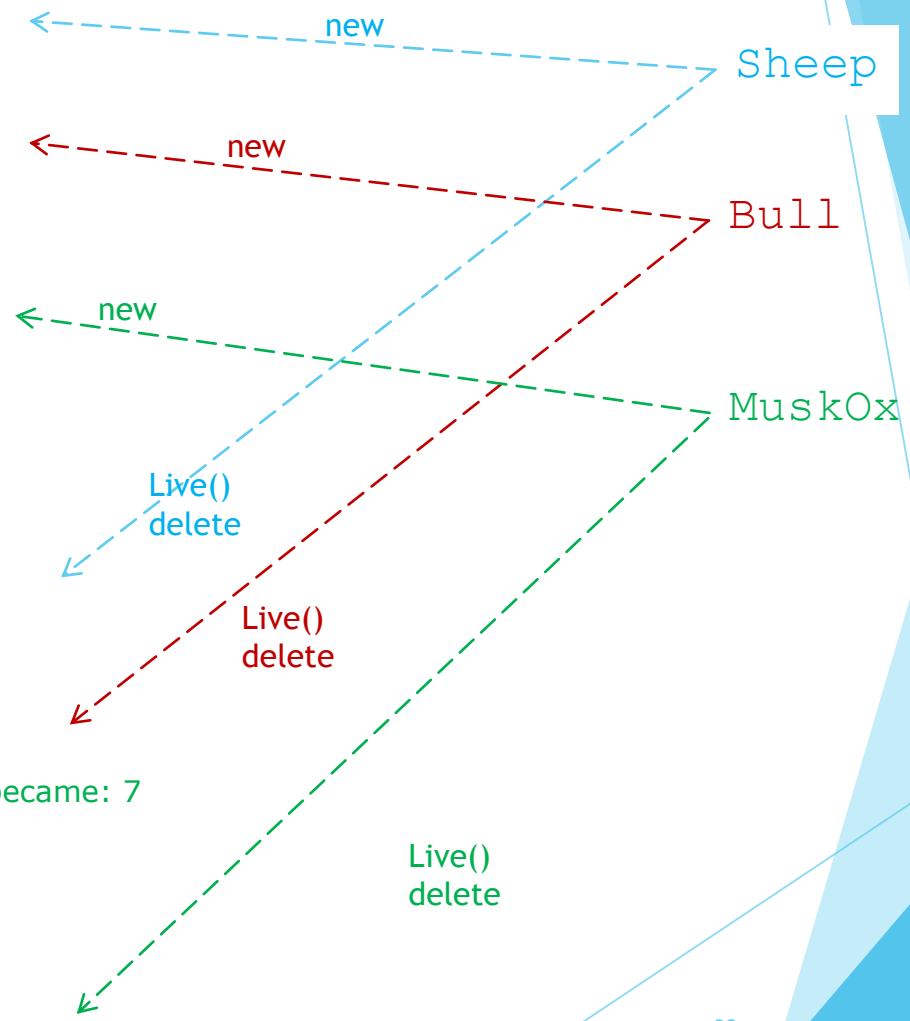


Функция main

```
int main()
{
    Mammals * mammals[3];
    mammals[0] = new Sheep("Zor\'ka", 5);
    mammals[1] = new Bull("Mishka", 8);
    mammals[2] = new MuskOx("Vas\'ka", 12,11,8);
    for(int i=0; i < 3; i++)
    {
        mammals[i]->Live();
        delete mammals[i];
    }
    return 0;
}
```

Результат работы программы

Constructor mammals. Name: Zor'ka
Constructor sheep. Wool: 5
Constructor mammals. Name: Mishka
Constructor bull. Horn: 8
Constructor mammals. Name: Vas'ka
Constructor bull. Horn: 12
Constructor sheep. Wool: 11
Constructor musk-ox. Angry: 8
Sheep eat flowers. Wool became 6
Destructor sheep. Wool: 6
Destructor mammals. Name: Zor'ka
Bull eat grass. Horn became 9
Destructor bull. Horn: 9
Destructor mammals. Name: Mishka
MuskOx eat grass and flowers. Angry became: 7
Destructor musk-ox. Angry: 7
Destructor sheep. Wool: 11
Destructor bull. Horn: 12
Destructor mammals. Name: Vas'ka



Конец