Множественное наследование

- Класс может быть порождён из любого числа базовых классов.
- Наличие более чем одного непосредственного базового класса называется **множественным наследованием**.

```
class A { ... };
class B { ... };
class C { ... };

class D : public A, public B, private C
  { ... };
```

```
class A
 { public:
    void f1();
    virtual void g() = 0;
  };

class B
 { public:
    void f2();
    virtual void h() = 0;
  };

class C : public A, public B
 { public:
    void g();      // Замещение A::g()
    void h();      // Замещение B::h()
  };
```

```
void f1(A *p) { p->f1(); }
void f2(B *p) { p->f2(); }

void main()
 { C c;

   f1(&c);
   f2(&c);
   A *p = &c;
   p->g();              // Правильно
   p->h();    // Ошибка: функция h не является
        // членом класса A
   dynamic_cast<B *>(p)->h();   // Правильно
  }
```

```
class A { ... };
class B : public A, public A { ... };
```

```
class A { ... };
class B : public A { ... };
class C : public A { ... };
class D : public B, public C { ... };
```

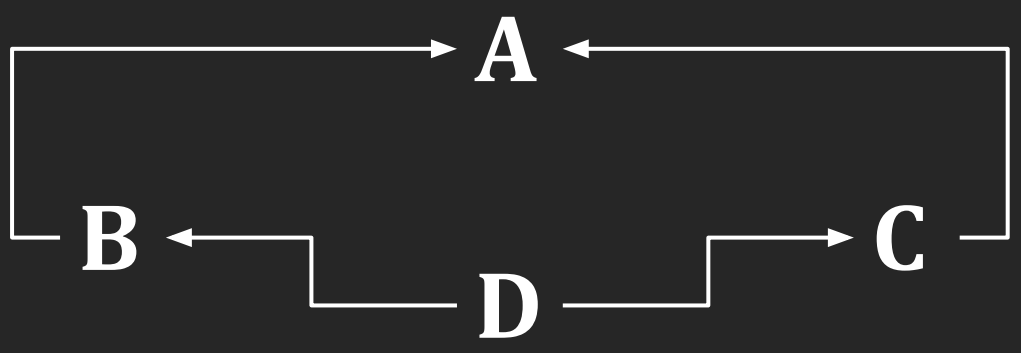
```
D *pd = new D;
A *pa = pd;         // Неоднозначность!
pa = (A*)pd;        // Неоднозначность!

// Приведение к указателю на A в объекте B
pa = (B*)pd;

// Приведение к указателю на A в объекте C
pa = (C*)pd;
```

```
class A { ... };
class B : public A { ... };
class C : public A { ... };
class D : public A, public B, public C
 { ... };

// Приведение к указателю на A
// непосредственно в объекте D
pa = pd;
// Приведение к указателю на A в объекте B
pa = (B*)pd;
// Приведение к указателю на A в объекте C
pa = (C*)pd;
```

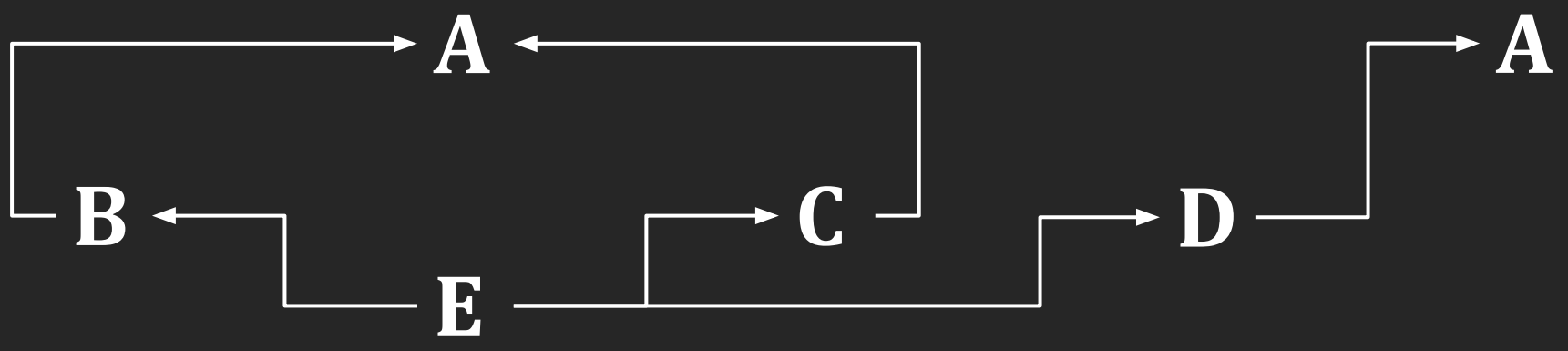
```
class A
 { protected:
    void f(int x);    // Защищённая функция
  };

class B1 : public A
 { public:
    void f(double x);
  };

class B2 : public A
 { public:
    void f(char x);
    using A::f;
  };
```

```
class C : public B1, public B2
 { public:
    void f(char *s);
    using A::f;
    using B1::f;
    using B2::f;
  };

void main()
 { C c;

   c.f(1);  // Вызов функции A::f(int)
   c.f(2.4);// Вызов функции B1::f(double)
   c.f('&');// Вызов функции B2::f(char)
   c.f("abc");// Вызов функции C::f(char*)
  }
```

```
class A { ... };
class B : virtual public A { ... };
class C : virtual public A { ... };
class D : public B, public C { ... };
```



```
class A { ... };
class B : virtual public A { ... };
class C : virtual public A { ... };
class D : public A { ... };
class E : public B, public C, public D
{ ... };
```



```
class A { ... };
class B : virtual public A { ... };
class C : virtual public A { ... };
class D : public A, public B, public C
 { ... };
```

```
class A
 { private:
    int n;
   public:
    A(int nn) : n(nn) { }
  };

class B1 : virtual public A
 { private:
    int n;
   public:
    B1(int a, int nn) : A(a), n(nn) { }
  };
```

```
class B2 : virtual public A
 { private:
    int n;
   public:
    B2(int a, int nn) : A(a), n(nn) { }
  };

class C : public B1, public B2
 { private:
    int n;
   public:
    C(int a, int b1, int b2, int nn) :
        A(a), B1(0, b1), B2(0, b2), n(nn) { }
  };
```

```
class A
 { private:
    int n;
   public:
    A(int nn) : n(nn) { }
  };

class B1 : public A
 { private:
    int n;
   public:
    B1(int a, int nn) : A(a), n(nn) { }
  };
```

```
class B2 : public A
 { private:
    int n;
   public:
    B2(int a, int nn) : A(a), n(nn) { }
  };

class C : public B1, public B2
 { private:
    int n;
   public:
    C(int a, int b1, int b2, int nn) :
          B1(a1, b1), B2(a2, b2), n(nn) { }
  };
```

```
class A
 { public:
    virtual ~A() { }
    virtual void g();
    virtual void h();
  };

class B1 : virtual public A
 { public:
    void g();
  };

class B2 : virtual public A
 { public:
    void h();
  };
```

```
class C : public B1, public B2 { };

void main()
 { C c;
   A *p = &c;

   p->g();     // Вызов функции B1::g
   p->h();     // Вызов функции B2::h
  }
```

```
class A
 { public:
    virtual ~A() { }
    virtual void g();
    virtual void h();
  };

class B1 : virtual public A
 { public:
    void g();
    void h();
  };

class B2 : virtual public A
 { public:
    void g();
    void h();
  };
```

```
class C : public B1, public B2
{ public:
    void g();
  };

void main()
 { C c;
   A *p = &c;

   p->g();
   p->h();
  }
```

```
class Base
 { public:
    virtual void open()  = 0;
    virtual void close() = 0;
             void print();
    virtual      ~Base() { }
  };

class D1 : public Base
 { public:
    void open();
    void close();
    void print();
  };
```

```
class D2 : public Base
 { public:
    void open();
    void close();
    void print();
  };

// Определяем тип для указателя
// на функцию-член класса Base
typedef void (Base::*PF)();
```

```
void main()
 { PF pf[] = { &Base::open,
               &Base::close,
               &Base::print };

   D1 d;
   (d.*pf[0])();
   (d.*pf[1])();
   (d.*pf[2])();
```

```
 Base *pb = new D2;
 (pb->*pf[0])();
 (pb->*pf[1])();
 (pb->*pf[2])();
}
```