Пример 1. Прямая и косвенная базы.

```
class A
{
public:
	A(char* s) { cout<<"Creature A"<<s<<";"<<endl; }
};

class B : public A
{
public:
	B() : A(" from B") { cout<<"Creature B;"<<endl; }
};

class C : public B, public A
{
public:
	C() : A(" from C") { cout<<"Creature C;"<<endl; }
};

void main()
{
	C obj;
}
```

Пример 2. Виртуальное наследование.

```
class A
{
public:
	A(char* s) { cout<<"Creature A"<<s<<";"<<endl; }
};

class B : virtual public A
{
public:
	B() : A(" from B") { cout<<"Creature B;"<<endl; }
};

class C : public B, virtual public A
{
public:
	C() : A(" from C") { cout<<"Creature C;"<<endl; }
};

void main()
{
	C obj;
}
```

Пример 4. Доминирование.

```
class A
{
public:
	void f() { cout<<"Executing f() from A;"<<endl; }
	void f(int i) { cout<<"Executing f(int) from A;"<<endl; }
};

class B : virtual public A
{
public:
	void f() { cout<<"Executing f from B;"<<endl; }
	using A::f; // плохо!!!
};
```

```
class C : virtual public A
{
};

class D : virtual public C, virtual public B
{
};

void main()
{
	D obj;

	obj.f();
	obj.f(1);
}
```

Пример 5. Доминирование.

```
class A
{
public:
	void f() { cout<<"Executing f from A;"<<endl; }
};

class B : virtual public A
{
public:
	void f() { cout<<"Executing f from B;"<<endl; }
};

class C : public B, virtual public A
{
};

void main()
{
	C obj;

	obj.f();
}
```

Пример 8. Множественный вызов методов.

```
class A
{
public:
	void f() { cout<<"Executing f from A;"<<endl; }
};

class B : virtual public A
{
public:
	void f()
	{
		A::f();
		cout<<"Executing f from B;"<<endl;
	}
};

class C : virtual public A
{
public:
	void f()
	{
		A::f();
		cout<<"Executing f from C;"<<endl;
```

```
	}
};

class D : virtual public C, virtual public B
{
public:
	void f()
	{
		C::f();
		B::f();
		cout<<"Executing f from D;"<<endl;
	}
};

void main()
{
	D obj;

	obj.f();
}
```

Пример 9. Решение проблемы множественного вызова методов.

```
class A
{
protected:
	void _f() { cout<<"Executing f from A;"<<endl; }
public:
	void f() { this->_f(); }
};

class B : virtual public A
{
protected:
	void _f() { cout<<"Executing f from B;"<<endl; }
public:
	void f()
	{
		A::_f();
		this->_f();
	}
};

class C : virtual public A
{
protected:
	void _f() { cout<<"Executing f from C;"<<endl; }
public:
	void f()
	{
		A::_f();
		this->_f();
	}
};

class D : virtual public C, virtual public B
{
protected:
	void _f() { cout<<"Executing f from D;"<<endl; }
public:
	void f()
	{
		A::_f(); C::_f(); B::_f();
		this->_f();
	}
};
```

```
void main()
{
	D obj;

	obj.f();
}
```

Пример 10. Неоднозначности при множественном наследовании.

```
class A
{
public:
	int a;
	int (*b)();
	int f();
	int f(int);
	int g();
};
```

```
class B
{
	int a;
	int b;
public:
	int f();
	int g;
	int h();
	int h(int);
};
```

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

```
class D
{
public:
	static void fun(C& obj)
	{
		obj.a = 1;	// Error!!!
		obj.b();	// Error!!!
		obj.f();	// Error!!!
		obj.f(1);	// Error!!!
		obj.g = 1;	// Error!!!
		obj.h(); obj.h(1); // Ok!
	}
};

void main()
{
	C obj;

	D::fun(obj);
}
```

Пример 7. Замена интерфейса.

```
class A
{
public:
	void f1() { cout<<"Executing f1 from A;"<<endl; }
	void f2() { cout<<"Executing f2 from A;"<<endl; }
};

class B
{
public:
	void f1() { cout<<"Executing f1 from B;"<<endl; }
	void f3() { cout<<"Executing f3 from B;"<<endl; }
};
```

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

class D
{
public:
	void g1(A& obj)
	{
		obj.f1(); obj.f2();
	}
	void g2(B& obj)
	{
		obj.f1(); obj.f3();
	}
};

void main()
{
	C obj;
	D d;

	// obj.f1();  Error!!! Неоднозначность
	// d.g1(obj); Error!!! Нет приведения к базовому классу при наследовании по схеме private
	d.g2(obj);
}
```

Пример 6. Объединение интерфейсов.

```
class A
{
public:
	void f1() { cout<<"Executing f1 from A;"<<endl; }
	void f2() { cout<<"Executing f2 from A;"<<endl; }
};

class B
{
public:
	void f1() { cout<<"Executing f1 from B;"<<endl; }
	void f3() { cout<<"Executing f3 from B;"<<endl; }
};

class C : public A, public B {};

class D
{
public:
	void g1(A& obj)
	{
		obj.f1(); obj.f2();
	}
	void g2(B& obj)
	{
		obj.f1(); obj.f3();
	}
};

void main()
{
	C obj;
	D d;

	d.g1(obj);
	d.g2(obj);
}
```

Пример 11. Виртуальные методы.

```
class A
{
public:
	virtual void f() { cout<<"Executing f from A;"<<endl; }
};

class B : public A
{
public:
	virtual void f() override { cout<<"Executing f from B;"<<endl; }
};

class C
{
public:
	static void g(A& obj) { obj.f(); }
};

void main()
{
	B obj;

	C::g(obj);
}
```

Пример 12. Абстрактный класс. Чисто виртуальные методы.

```
class A // abstract
{
public:
	virtual void f() = 0;
};

class B : public A
{
public:
	virtual void f() override { cout<<"Executing f from B;"<<endl; }
};

class C
{
public:
	static void g(A& obj) { obj.f(); }
};

void main()
{
	B obj;

	C::g(obj);
}
```

Пример 15. Виртуальный деструктор.

```
class A
{
public:
	virtual ~A() = 0;

};

A::~A() {}

class B : public A
```

```
{
public:
	virtual ~B() { cout<<"Class B destructor called;"<<endl; }
};

void main()
{
	A* pobj = new B();
	delete pobj;
}
```

Пример 13. Виртуальные методы и конструкторы и деструкторы.

```
class A
{
public:
	virtual ~A() { cout<<"Class A destructor called;"<<endl; }

	virtual void f() { cout<<"Executing f from A;"<<endl; }
};

class B : public A
{
public:
	B() { this->f(); }
	virtual ~B()
	{
		cout<<"Class B destructor called;"<<endl;
		this->f();
	}

	void g() { this->f(); }

};

class C : public B
{
public:
	virtual ~C() { cout<<"Class C destructor called;"<<endl; }

	virtual void f() override { cout<<"Executing f from C;"<<endl; }
};

void main()
{
	C obj;

	obj.g();
}
```

Пример 14. Дружба и наследование.

```
class C; // forward объявление

class A
{
private:
	void f1() { cout<<"Executing f1;"<<endl; }

	friend C;
};

class B : public A
{
private:
	void f2() { cout<<"Executing f2;"<<endl; }
```

```
};

class C
{
public:
	static void g1(A& obj) { obj.f1(); }
	static void g2(B& obj)
	{
		obj.f1();
//		obj.f2(); // Error!!! Имеет доступ только к членам A
	}
};

class D : public C
{
public:
//	static void g2(A& obj) ( obj.f1(); } // Error!!! Дружба не наследуется
};



void main()
{
	A aobj;

	C::g1(aobj);

	B bobj;

	C::g1(bobj);
	C::g2(bobj);
}
```

Пример 15. Дружба и виртуальные методы.

```
class C; // forward объявление

class A
{
protected:
	virtual void f() { cout<<"Executing f from A;"<<endl; }

	friend C;
};

class B : public A
{
protected:
	virtual void f() override { cout<<"Executing f from B;"<<endl; }
};

class C
{
public:
	static void g(A& obj) { obj.f(); }
};

void main()
{
	B bobj;

	C::g(bobj);
}
```