

Зависимые и независимые имена

Зависимые и независимые имена

- *Зависимое имя* - имя, которое зависит от параметра шаблона.
- *Независимое имя* - имя, которое не является зависимым.

```
template <class T>
struct A {
    static double x;
};

struct B {
    static int y;
};

template <class T>
void f() {
    B::y;           // независимое имя
    A<long>::x;      // независимое имя
    A<T>::x;         // зависимое имя
}
```

Зависимые имена типов

```
template <class T>
struct A {
    using X = T;
};

template <>
struct A<int> {
    const static int X = 0;
}

template <class T>
void f(T y) {
    A<T>::X* z;
}
```

Где беда?

Зависимые имена типов

```
template <class T>
struct A {
    using X = T;
};

template <>
struct A<int> {
    const static int X = 0;
};

template <class T>
void f(T y) {
    A<T>::X* z;    // Это умножение или указатель на тип?
}
```

error: dependent-name 'A<T>::X' is parsed as a non-type, but instantiation yields a type

Зависимые имена типов

Беда в том, что по умолчанию компилятор рассматривает такие имена как статические члены класса. И он очень расстраивается, если данное имя на самом деле оказывается типом.

```
template <class T>
void f(T y) {
    A<T>::X* z;    // Компилятор считает, что A<T>::X - это статическое поле
}
```

```
error: dependent-name 'A<T>::X' is parsed as a non-type, but instantiation yields a type
```

Благо в следующей строчке он подсказывает, что нужно сделать

```
note: say 'typename A<T>::X' if a type is meant
```

typename для зависимых имен типов

Чтобы зависимое имя воспринималось компилятором как тип, об этом нужно явно попросить

```
template <class T>
struct A {
    using X = T;
};

template <>
struct A<int> {
    const static int X = 0;
};

template <class T>
void f(T y) {
    typename A<T>::X* z; // typename говорит о том, что A<T>::X это тип
}
```

typename для зависимых имен типов

typename не ставится, если по контексту понятно, что это тип, а не член

```
template <class T>
struct B : A<T>::X { ... }; // наследование возможно только от типа
```

A<T>::X::type; // операция :: есть только у типов

```
template <class T>
class S {
    using X = int;

    void f() {
        S<T>::X* y; // текущее инстанцирование, неоднозначности нет
    }
}
```

Зависимые имена шаблонов

Зависимые имена шаблонов

```
template <class T>
struct S {
    template <class U>
    void f(int) {}
};

template <>
struct S<int> {
    int f = 0;
};

template <class T>
void g() {
    S<T> s;
    s.f<int>(0);
}
```

Что может пойти не так?

Зависимые имена шаблонов

```
template <class T>
struct S {
    template <class U>
    void f(int) {}
};
```

```
template <>
struct S<int> {
    int f = 0;
};
```

```
template <class T>
void g() {
    S<T> s;
    s.f<int>(0); // CE
}
```

error: expected primary-expression before 'int'

Зависимые имена шаблонов

```
template <class T>
void g() {
    S<T> s;
    s.f<int>(0); // "s.f" "<" "int" ">" "(0)"
}
```

```
error: expected primary-expression before 'int'
```

Дело в том, что компилятор не знает, что представляет из себя `S<T>` (какой тип `T` будет подставлен).

Он предполагает по умолчанию, что `s.f` - это поле класса, а `<` - это оператор "меньше".

template для зависимых имен шаблонов

Как и ранее, нужно дать подсказку компилятору.

В данном случае - дать указание, что имя является именем шаблона

```
template <class T>
void g() {
    S<T> s;
    s.template f<int>(0); // теперь компилятор знает, что f - имя шаблона
}
```

Зависимые базовые классы

Зависимые базовые классы

```
template <class T>
struct Base {
    int x;
};

template <class T>
struct Derived : Base<T> {
    void f() {
        x = 0;
    }
};
```

Но здесь-то все в порядке?

Зависимые базовые классы

```
template <class T>
struct Base {
    int x;    // Am I joke to you?
};

template <class T>
struct Derived : Base<T> {
    void f() {
        x = 0;    // как же так?
    }
};
```

error: 'x' was not declared in this scope

Зависимые базовые классы

```
template <class T>
struct Base {
    int x; // Am I joke to you?
};

template <class T>
struct Derived : Base<T> {
    void f() {
        x = 0; // как же так?
    }
};
```

error: 'x' was not declared in this scope

`Base<T>` - неизвестный тип, так как тип `T` не определен. Поэтому компилятор не знает, что в `Derived` есть поле `x` (Есть ли? Может есть специализация без него)

this для зависимых базовых классов

Снова натолкнем компилятор на истинный путь

```
template <class T>
struct Base {
    int x;
};

template <class T>
struct Derived : Base<T> {
    void f() {
        Base<T>::x = 0; // например, так
        this->x = 0;    // или так
    }
};
```

