陆初

脑子不太好用的普通人。

继承模板类的子类无法访问protected成员（坑

最近花了点时间把essential C++的第五章（讲继承）和第六章（讲模板）的部分给看完了。自己试着写一个栈来练习一下，以下是代码

```
template <typename type>
class my_stack {
public:
        virtual void push(type v) = 0;
        virtual void pop() = 0;
        virtual type top() const = 0;
        virtual unsigned int size() const = 0;
        virtual bool empty() const = 0;
        virtual bool full() const = 0;
        virtual ~my_stack() {}
};



template <typename type>
class LIFO_stack : public my_stack<type> {
        template <typename ano_type>
        friend ostream& operator<<(ostream& os, LIFO_stack&
<ano_type> s);
public:
        LIFO_stack() {}
        explicit LIFO_stack(vector vector) {
                for (auto v : vector) {
                        vec.push_back(v);
                }
        }
        void push(type v) {
                vec.push_back(v);
        }
        type top() const {
                return *(--vec.end());
```

```
        }
        void pop() {
                vec.pop_back();
        }
        unsigned int size() const {
                return vec.size();
        }
        bool empty() const {
                return !vec.size();
        }
        bool full() const {
                return vec.size() == vec.capacity();
        }
        virtual void print() const {
                cout << *this;
        }
        ~LIFO_stack() {}
protected:
        vector<type> vec;
};

template <typename type>
ostream& operator<<(ostream& os, LIFO_stack<type>& s) {
        auto it = --s.vec.end();
        for (; it != s.vec.begin(); --it) {
                os << *it << ' ';
        }
        os << *it << endl;
        return os;
}



template <typename type>
class Peekback_stack : public LIFO_stack<type> {
public:
        Peekback_stack() {}
        explicit Peekback_stack(vector vector) :
LIFO_stack(vector) {}
        type peek_elem(int pos) const {
                if (pos < vec.size()) {
                        return vec[vec.size() - 1 - pos];
                }
                else {
                        cerr << "invalid visit!";
                }
        }
};
```

我使用的是cgywin编译器，这段代码在编译的时候会出现这么一个错误：

```
/cygdrive/f/C_coding/MyCppProject/main.cpp: In member function 'type Peekback_stack<type>::peek_elem(int) const':
/cygdrive/f/C_coding/MyCppProject/main.cpp:88:18: error: 'vec' was not declared in this scope
         if(pos < vec.size()) {
                  ^~~
make[3]: *** [CMakeFiles/MyCppProject.dir/build.make:63: CMakeFiles/MyCppProject.dir/main.cpp.o] Error 1
Build failed in 5s 843ms
```

但是实际我已经在LIFO_stack这个类里面声明了vec为protected，作为子类的Peekback_stack应该继承了这个父类的成员才对……

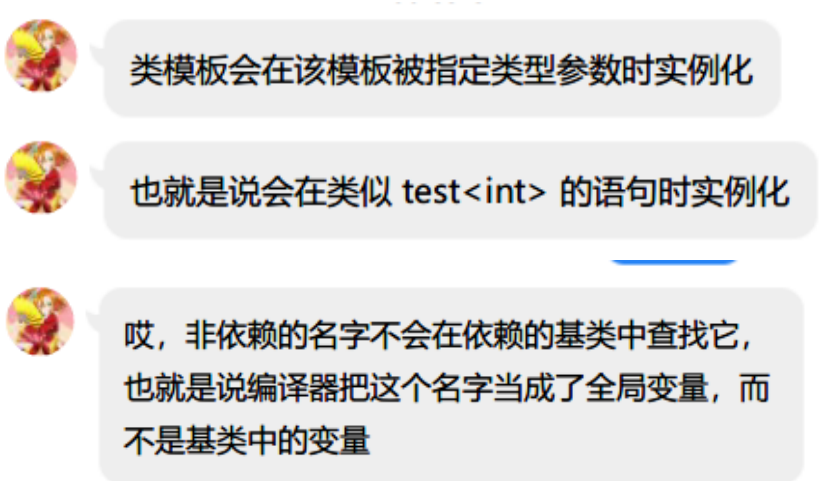
我百思不得其解，终于在yjp同学的提供下得到了这两个答案：

https://stackoverflow.com/questions/6592512/templates-parent-class-member-variables-not-visible-in-inherited-class

https://stackoverflow.com/questions/38639799/template-inheritance-and-a-base-member-variable

根本原因：

*This is because the template parent of a template class **is not instantiated during the compilation** pass that first examines the template. These names appear to be non-dependent on the particular template instantiation, and therefore the definitions need to be available.*



恍然大悟。

也就是说，在编译过程中，LIFO_stack还没有被实例化，其中的vec变量自然也还没有被实例化，而其子类Peekback_stack使用了「vec」这个变量名，由于编译过程中父类没有被实

例化，子类自然也没有继承到「vec」这个变量，所以最终编译器会在**全局**域内寻找vec这个变量。

而全局域是不存在「vec」这个变量的，所以编译器就会报错。

解决方法：

在vec前面加上this指针：「this->vec」表明vec是一个依赖的变量。

在vec前面表明使用域：「LIFO_stack::vec」表明vec被定义在LIFO_stack这个域内。

最终代码：

```
template <typename type>
class Peekback_stack : public LIFO_stack<type> {
public:
        Peekback_stack() {}
        explicit Peekback_stack(vector vector) :
LIFO_stack(vector) {}
        type peek_elem(int pos) const {
                if (pos < this->vec.size()) {
                        return this->vec[this->vec.size() - 1
- pos];
                }
                else {
                        cerr << "invalid visit!";
                }
        }
};
```

搞定。

顺带一提，在visual studio内就没有这个问题，所以这个问题并不是一定会出现，还是要看编译器具体怎么做。

2019/4/18 补充：

最近在看一个非常不错的C++ template入门教程：

https://github.com/wuye9036/CppTemplateTutorial

在这里面的2.3.2节中提到了模板中名字查找(name lookup)的一个过程：

在官方文档为：

> *14.6 名称解析（Name resolution）*
>
> *1) 模板定义中能够出现以下三类名称：*
>
> - *模板名称、或模板实现中所定义的名称；*
> - *和模板参数有关的名称；*
> - *模板定义所在的定义域内能看到的名称。*
>
> *…*
>
> *9) … 如果名字查找和模板参数有关，那么查找会延期到模板参数全都确定的时候。 …*
>
> *10) 如果（模板定义内出现的）名字和模板参数无关，那么在模板定义处，就应该找得到这个名字的声明。…*
>
> *14.6.2 依赖性名称（Dependent names）*
>
> *1) …（模板定义中的）表达式和类型可能会依赖于模板参数，并且模板参数会影响到名称查找的作用域 … 如果表达式中有操作数依赖于模板参数，那么整个表达式都依赖于模板参数，名称查找延期到模板实例化时进行。并且定义时和实例化时的上下文都会参与名称查找。（依赖性）表达式可以分为类型依赖（类型指模板参数的类型）或值依赖。*

> *14.6.2.2 类型依赖的表达式*

> *2) 如果成员函数所属的类型是和模板参数有关的，那么这个成员函数中的 `this` 就认为是类型依赖的。*

> *14.6.3 非依赖性名称（Non-dependent names）*

> *1) 非依赖性名称在模板定义时使用通常的名称查找规则进行名称查找。*

也就是说，模板上的名字查找实际分为两个阶段：

1. 模板定义阶段
2. 模板类实例化阶段

如果在模板定义中的名字(name)是「类型依赖」的，那么编译器会留到实例化时才进行名字查找。

如果该模板定义的名字(name)是「非依赖性名称」，那么编译器会在定义时就进行名字查找。

在这里面，基类中的vector<type> vec这一句表明了vec是一个「类型依赖」的名称，所以会留到实例化时进行名字查找。

而子类中直接使用了vec这个名字，没有任何迹象表明它是一个「类型依赖」的名字，所以编译器会在定义的时候在寻找vec，但是由于在函数内部还是全局中都没有「vec」这个名字，所以会报错。

所以为了解决这个问题，要在vec前面加「this->」，由于「this」是一个「类型依赖」的名字，所以this->vec也是一个「类型依赖」的名字，编译器会在实例化时进行vec这个名字的查找。

LIFO_stack::vec同理。因为LIFO_stack也是一个「类型依赖」的名字。

作者： 陆初

脑子不太好用的普通人。 顺带一提性格也有点古怪。 在老妈子和厌世肥宅中来回切换。
查看陆初的所有文章

陆初 / 2018-07-20 / C++ /

《继承模板类的子类无法访问protected成员（坑》有一个想法

to

2018-07-20 12:59

zaima

此站点使用Akismet来减少垃圾评论。了解我们如何处理您的评论数据。