陆初
脑子不太好用的普通人。

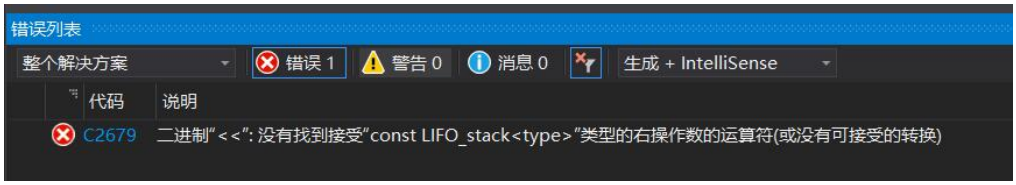
（续上篇）被自己蠢到的关于const关键字的坑

在我实现的LIFO_stack类里面，有一个print()函数，而这个函数是通过调用<<操作符实现的：

```
template <typename type>
class LIFO_stack : my_stack<type> {
virtual void print() const {
                cout << *this;
        }
}

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;
}
```

调用<<运算符输出LIFO_stack的话，编译器会报出以下错误：



我一开始还不太明白是什么原因……后来经过jp同学的提醒，才发现我的LIFO_stack参数没有加const修饰，而print()函数用了const修饰，不匹配，所以不能调用。

解决方法：

在operator<<的友元声明和定义中都给LIFO_stack加个const修饰就行了。

关于const关键字：

如果是在成员函数(menber function)后加的话，表明「这个函数不会破坏这个对象的常量性」，也就是「这个函数不会修改这个对象的任何变量（mutable变量除外）」。

例如

```
class A {
    void print() const ;
}
```

就表明print()这个函数不会改变A对象的任何成员变量（还是mutable变量除外）。

所以在这个函数内不允许有任何可能修改A对象的行为。

> 所以这个问题的答案就得到解答了。
>
> print()里面调用了<<运算符，而<<运算符里面的LIFO_stack不是const参数，有潜在的修改LIFO_stack对象的行为，所以不允许调用<<运算符。

如果const被放在某个变量之前：

```
const A a;
void test(const A a);
```

则表明这个变量是一个const变量。

const变量只能调用被声明为const的函数，因为没有被声明为const的函数无法保证其常量性不被破坏。

例如说，如果print()没有声明为const，那么上面这两个a变量都无法调用a.print()，即使print()里面没有修改a。

所以要牢记const变量只能调用被声明为const的成员函数！！！

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++ /

《（续上篇）被自己蠢到的关于const关键字的坑》有一个想法

超绝可爱唱歌超甜的17岁白丝jk

2018-07-20 19:06

不用想那么多，只需要记住后置const是用来修饰this的就行了，其他的一切都是const this的所导致的后果。