智能指针有没有内存泄漏的情况

答案： 有内存泄露的情况

1. **情况说明： 当两个对象同时使用一个shared_ptr成员变量指向对方，会造成循环引用，使引用计数失效，从而导致内存泄露。**
2. **如何解决： 为了解决循环引用导致的内存泄漏，引入了弱指针weak_ptr，weak_ptr的构造函数不会修改引用计数的值，从而不会对对象的内存进行管理，其类似一个普通指针，但是不会指向引用计数的共享内存，但是可以检测到所管理的对象是否已经被释放，从而避免非法访问。**

```
class Child;
class Parent{
private:
    std::shared_ptr<Child> ChildPtr;
public:
    void setChild(std::shared_ptr<Child> child) {
        this->ChildPtr = child;
    }

    void doSomething() {
        if (this->ChildPtr.use_count()) {

        }
    }

    ~Parent() {

    }
};

class Child{
private:
    std::shared_ptr<Parent> ParentPtr;
public:
    void setPartent(std::shared_ptr<Parent> parent) {
        this->ParentPtr = parent;
    }
    void doSomething() {
        if (this->ParentPtr.use_count()) {

        }
    }
    ~Child() {
    }
};

int main() {
    std::weak_ptr<Parent> wpp;
    std::weak_ptr<Child> wpc;
```

```
    {
        std::shared_ptr<Parent>    p(new Parent);
        std::shared_ptr<Child> c(new Child);
        p->setChild(c);
        c->setPartent(p);
        wpp = p;
        wpc = c;
        std::cout << p.use_count() << std::endl;
        std::cout << c.use_count() << std::endl;
    }
    std::cout << wpp.use_count() << std::endl;
    std::cout << wpc.use_count() << std::endl;
```

share_ptr 怎么知道跟它共享对象的指针释放了

同一个shared_ptr可指向同一个动态对象，并维护一个共享的引用计数器，记录了引用同一对象的shared_ptr实例的数量。当最后一个指向动态对象的shared_ptr销毁时，会自动销毁其所指对象(通过delete操作符)

weak_ptr 如何解决 shared_ptr 的循环引用问题?

为了解决循环引用导致的内存泄漏，引入了弱指针weak_ptr，weak_ptr的构造函数不会修改引用计数的值，从而不会对对象的内存进行管理，其类似一个普通指针，但是不会指向引用计数的共享内存，但是可以检测到所管理的对象是否已经被释放，从而避免非法访问

weak_ptr 能不能知道对象计数为 0，为什么?

不能。weak_ptr只可以从shared_ptr或者另一个weak_ptr对象构造，是配合shared_ptr而引入的一种智能指针，只是提供了一种访问对象的手段，而不会对对象的内存进行管理，因为对象计数是由share_ptr管理的，weak_ptr的构造和析构不会影响计数多少