1、 Qt 信号槽机制的实现
2、 QT bind()实现机制
3、 C++智能指针
4、 shared_ptr 是多线程安全的吗 自己如何实现一个 shared_ptr
5、 一个函数 void printNumber(int N)，在这个函数内创建两个线程，一个依次打印 0 到 N 之间的偶数 0，2，4，、、、、，一个依次打印 0 到 N 之间的奇数 1，3，5，、、、，现让两个线程交替打印，使输出 0、1、2、3、4、5、、、、

shared_ptr 实现

```
#ifndef __SHARED_PTR_
#define __SHARED_PTR_

template <typename T>
class shared_ptr {
public:
	shared_ptr(T* p) : count(new int(1)), _ptr(p) {}
	shared_ptr(shared_ptr<T>& other) : count(&(++*other.count)), _ptr(other._ptr) {}
	T* operator->() { return _ptr; }
	T& operator*() { return *_ptr; }
	shared_ptr<T>& operator=(shared_ptr<T>& other)
	{
		++*other.count;
		if (this->_ptr && 0 == --*this->count)
		{
			delete count;
			delete _ptr;
		}
		this->_ptr = other._ptr;
		this->count = other.count;
		return *this;
	}
	~shared_ptr()
	{
		if (--*count == 0)
		{
			delete count;
			delete _ptr;
		}
	}
	int getRef() { return *count; }
private:
	int* count;
	T* _ptr;
};
```

#endif
一般来说，智能指针的实现需要以下步骤：
1.一个模板指针 T* ptr，指向实际的对象。
2.一个引用次数(必须 new 出来的，不然会多个 shared_ptr 里面会有不同的引用次数而导致多次 delete)。
3.重载 operator*和 operator->，使得能像指针一样使用 shared_ptr。
4.重载 copy constructor，使其引用次数加一。
5.重载 operator=，如果原来的 shared_ptr 已经有对象，则让其引用次数减一并判断引用是否为零(是否调用 delete)。
　然后将新的对象引用次数加一。
6.重载析构函数，使引用次数减一并判断引用是否为零(是否调用 delete)。

两个线程交叉打印

```
#include <thread>
#include <iostream>
#include <mutex>
#include <condition_variable>

std::mutex data_mutex;
std::condition_variable data_var;
bool label = false;

void printodd()
{
	std::unique_lock<std::mutex> ulock(data_mutex) ;
	for(int odd = 1; odd <= 100; odd += 2 )
	{
		data_var.wait(ulock,[]{return label;});
		std::cout<< std::this_thread::get_id() << ": " << odd <<std::endl;
		label = false;
		data_var.notify_one();
	}
}

void printeven()
{
	std::unique_lock<std::mutex> ulock(data_mutex) ;
	for(int even = 0; even < 100; even += 2 )
	{
		std::cout<< std::this_thread::get_id() << ": " << even <<std::endl;
```

```
            data_var.notify_one();
            label = true;
            data_var.wait(ulock,[]{return !label;});
        }
}

int main()
{
        std::thread t1(printeven);
        std::thread t2(printodd);
        t1.join();
        t2.join();
        std::cout<<"end!"<<std::endl;
        return 0;
}
```