智能指针

请实现一个智能指针类`SmartPointer`，要求如下：

1、其是一个模板类，具有类型参数T；

2、具有一个`T* ptr`和`int* ref_count`两个数据属性；

3、实现空构造函数、常规构造函数（参数为指针类型）、拷贝构造函数；

4、实现赋值运算符重载，包括同类型以及指针类型的赋值；

5、重载*取内容运算符，重载`->`调用运算符（重载原型为`MyClass* operator->(){ return ptr; }`），用于类似指针的调用；

6、实现一个析构函数，用于指针内存的释放：当引用计数`==1`时，动态释放内存。

为了测试这个智能指针，请再构建一个简单的测试类型`TestClass`,要求如下：

1、`TestClass`只有一个智能指针对象数据成员`SmartPointer<T> sp`；

2、其有一个常规构造函数，接受一个智能指针对象作为参数，用于初始化`sp`；

3、其有一个拷贝构造函数和一个赋值运算符重载函数，均使用浅拷贝直接赋值`sp`；

测试输入输出请参考以下示例代码：

```cpp
#include <iostream>
using namespace std;

template<typename T>
class SmartPointer {
private:
    T* ptr;              // 指向动态分配的对象的指针
    int* ref_count;     // 引用计数指针

public:
    // 空构造函数
    SmartPointer() {
        cout<<"SmartPointer Constructor"<<endl;
        cout<<"create a new smart pointer"<<endl;
        // Your code here
    }

    // 常规构造函数
    explicit SmartPointer(T* p) {
        cout<<"SmartPointer Constructor"<<endl;
        cout<<"create a new smart pointer"<<endl;
        // Your code here
    }

    // 拷贝构造函数
    SmartPointer(const SmartPointer<T>& other){
        cout<<"SmartPointer Constructor"<<endl;
        // Your code here
    }

    // 赋值运算符重载
    SmartPointer<T>& operator=(const SmartPointer<T>& other) {
        cout<<"Assignment Operator"<<endl;
        // Your code here



        return *this;
    }

    // 赋值运算符重载（指针类型）
    SmartPointer<T>& operator=(T* newPtr) {
        cout<<"Assignment Operator"<<endl;
```

```
        // Your code here


        return *this;
    }

    // 重载*运算符
    T& operator*() const { return *ptr; }
    T& operator[](int i) const { return ptr[i]; }

    // 析构函数
    ~SmartPointer() {
        cout<<"Deconstructor"<<endl;
        if (ptr) {
            cout<<"decrease ref_count ["<<*ref_count<<"] by 1"<<endl;
            (*ref_count) --;
            // Your code here


        }
    }
};

template<typename T>
class TestClass {
public:
    SmartPointer<T> sp;  // 智能指针数据成员
    // 常规构造函数
    TestClass(SmartPointer<T> p) : sp(p) {}

    // 拷贝构造函数
    TestClass(const TestClass& other) : sp(other.sp) {}

    // 赋值运算符重载
    TestClass& operator=(const TestClass& other) {
        if (this != &other) {
            sp = other.sp;
        }
        return *this;
    }
};

int main() {
```

```
    int n;
    cin>>n;
    SmartPointer<int> p = SmartPointer<int>(new int[n]);
    for (int i = 0; i < n; ++i) {
        int m;
        std::cin>>m;
        p[i] = m;
    }

    TestClass<int>* TCPtr[10];
    for (int i = 0; i < n; ++i) {
        TCPtr[i] = new TestClass<int>(p);
        TCPtr[i]->sp[i] *= i;
    }
    for (int i = 1; i < n; ++i) {
        // check here
        TCPtr[i] = TCPtr[i-1];
        TCPtr[i]->sp[i] *= i*(i-1);
    }
    for (int i = 0; i < n; ++i) {
        delete TCPtr[i];
        TCPtr[i] = nullptr;
    }
    return 0;
}
```