Problem 3 : Smart Pointers

1 Introduction

What are smart pointers? Smart pointers are objects which behave like pointers but do more than a pointer. These objects are flexible as pointers and have the advantage of being an object (like constructor and destructors called automatically). A smart pointer is designed to handle the problems caused by using normal pointers.

1.1 Problems with pointers

What are the common problems we face in C++ programs while using pointers? The answer is memory management. Have a look at the following code:

```
char* pName  = new char[1024];
SetName(pName);

if(null != pName)
    delete[] pName;
```

How many times have we found a bug which was caused because we forgot to delete `pName`. It would be great if someone could take care of releasing the memory when the pointer is not useful. What if the pointer itself takes care of that? Yes, thats exactly what smart pointers are intended to do. Let us write a smart pointer and see how we can handle a pointer better.

2 Problem statement

Given the following interface for smart pointers:

```
template <class T> class SmartPointer{
    public :
        SmartPointer();
        SmartPointer(T *ptr);
        ~SmartPointer();

        SmartPointer<T> & operator=(SmartPointer<T> &sptr);
        unsigned get_count();
        T* operator->();
        T& operator*();

    protected:
        void remove();
        T *ref;
        unsigned *ref_count;
};
```

Protected members:

- `ref` represents the ordinary pointer that points to the object which the current smart pointer points to.

- `ref_count` records how many smart pointers point to the object that the current smart pointer points to.
- Function `remove()` performs as follows:
  - Decrease `ref_count` by 1.
  - If there is no other smart pointers that point to the object which the current smart pointer points to, deallocate the memory of the object.

Public member:

- Function `operator=(SmartPointer<T> &sptr)` overlaods the assignmant operator when assign to another smart pointer object

You are going to implement function `remove` and `operator=`. You are NOT allowed to modify the code outside the two functions.

2.1 Example

Given the following client of the `SmartPointer` class:

```
int main(){
    SmartPointer<Person> p1(new Person("Amy"));
    SmartPointer<Person> p2(new Person("Tom"));
    SmartPointer<Person> p3;

    cout << p1->Display() <<" : "<<p1.get_count() <<endl;
    cout << p2->Display() <<" : "<<p2.get_count() <<endl;

    p3=p1;
    cout << p3->Display() <<" : "<<p3.get_count() <<endl;

    p1=p3;
    cout << p1->Display() <<" : "<<p1.get_count() <<endl;

    p2=p1;
    cout << p2->Display() <<" : "<<p2.get_count() <<endl;
}
```

The output should be:

```
Amy : 1
Tom : 1
Amy : 2
Amy : 2
Tom is removed
Amy : 3
Amy is removed
```

3 Input/Output

There is no input file. Test cases are given in the `main` function.
You are NOT allowed to modify the code outside the `remove` and `operator=` functions; otherwise, you will receive zero credit.