“ **In the name of God** ”

Amirkabir university of technology
HomeWork 4

Advanced Programming C++
Dr.Jahanshahi

Writer : Sajad Ghadiri
Student Number : 9723067

Teaching assistant : Mr K.Behzad

In this homework I implemented 2 classes according to the structure of the README.md which was uploaded on Mr Behzad's github .
At first I imported his repo and then I cloned and i start to write basics of first class which is called Unique_Ptr.
I wrote unique_ptr.h like below :

```
template<typename T>
class UniquePtr
{

public:
    UniquePtr(T* N); // Constructor
    UniquePtr(); // Default constructor
    UniquePtr(const UniquePtr& ptr) =  delete ; // Mopy constructor
    UniquePtr(UniquePtr&& ptr); // Move constructor
    ~UniquePtr(); // Destructor

    T& operator=(const UniquePtr& ptr) = delete ; // Assign operator
    T& operator*(); // * operator
    T* operator->(); //
    operator bool(); // Bool operator

    T* get(); // Get raw pointer stored in the class
    void reset(); // Reset the pointer to nullptr
    T* reset(T* inp);// Reset the pointer & make new pointer by input value
    T* release();

private:

T* _p ;

};

#include "unique_ptr.hpp"
```

As you wanted us, I wrote all methods and operands by the " template " concept.
I wrote some comments which can guide to understand what did i do but I'll mention most important challenges in all codes and files :

1- In unigue_ptr class, defining " Copy Constructor " and " = operand " is meaningless. So by searching i found a way to write them and let the compiler understand to give me error at the right time .

2- I defined outputs of methods and operands T* and T& both according to the concept of them and example hints on the README.md file.

3- I defined bool operator to convert to bool variable and using in if condition in the unit_tests.cpp .
Also I defined make_unique function out of class body which is clear that makes a new pointer.

```cpp
////////////////////////////bool operator ////////////////
template<typename T>
UniquePtr<T>::operator bool()
{
    if(_p == nullptr)
    {
        return false ;
    }
    else
    {
        return true ;
    }
}
//////////////////////////// make_unique ////////////////
template<typename T>
T* make_unique(T p)
{
    return new T{p} ;
}
////////////////////////////////////////////////////////////
```

* For SharedPtr class every steps are similar until implementing use_count() method. After implementing this method which count number of instances pointing to a same Place,we must change some part of code like Constructors ,reset ,operand = and …

4- Most difference from uniqueptr is = operand which its output is SharedPtr<T>&
Not T& .
Also a new member variable which is int* number to use in use_count() method.

```
template<typename T>
class SharedPtr
{

public:

    SharedPtr(T* N); // Constructor
    SharedPtr(); // Default constructor
    ~SharedPtr(); // Destructor
    SharedPtr(const SharedPtr& ptr) ; // Copy constructor

    T& operator*(); // * operator
    T* operator->(); // -> operator
    SharedPtr<T>& operator=(const SharedPtr& ptr) ; // Assign operator

    T* get(); // Get raw pointer stored in the class
    void reset(); // Reset the pointer to nullptr
    T* reset(T* inp) ; // Reset the pointer and make new pointer by input value
    int use_count();   // Get the number of shared pointers
    explicit operator bool(); // Bool operator

private:
T* _p ;
int* number ;

};
```

5- To destruct all copy of a pointer, everytime I mines one time from “ number ” variable
And finally at the end i delete it’s memory and equal it to nullptr.

```
}
////////////////////// Destructor ////////////////////////////////
template<typename T>
SharedPtr<T>::~SharedPtr()
{
    *number = *number - 1 ;

    if(*number == 0)
    {
        delete _p ;
        _p = nullptr ;
    }
}
```

6- I should initialize " number " variable in Constructor and Default Constructor and then I must consider to increase number of pointers when I use Copy Constructor so :

```
////////////////////////// Constructor ////////////////////
template<typename T>
SharedPtr<T>::SharedPtr(T* N) : _p{N}
{
    number = new int(1) ;
}

////////////////////////// default constructor ////////////
template<typename T>
SharedPtr<T>::SharedPtr()
{
    _p = nullptr ;
    number = new int(0)  ;
}
////////////////////// copy constructor ////////////////
template<typename T>
SharedPtr<T>::SharedPtr(const SharedPtr& ptr)
{
    _p = ptr._p ;
    number = ptr.number  ;
    *number = *number + 1 ;
}
```

Final Result :))

```
[       OK ] HW4Test.TEST16 (0 ms)
[ RUN      ] HW4Test.TEST17
[       OK ] HW4Test.TEST17 (0 ms)
[ RUN      ] HW4Test.TEST18
[       OK ] HW4Test.TEST18 (0 ms)
[ RUN      ] HW4Test.TEST19
[       OK ] HW4Test.TEST19 (0 ms)
[ RUN      ] HW4Test.TEST20
[       OK ] HW4Test.TEST20 (0 ms)
[ RUN      ] HW4Test.TEST21
[       OK ] HW4Test.TEST21 (0 ms)
[----------] 21 tests from HW4Test (2 ms total)

[----------] Global test environment tear-down
[==========] 21 tests from 1 test suite ran. (2 ms total)
[  PASSED  ] 21 tests.
<<<SUCCESS>>>
ubuntu@0d9bb1c40db7:/usr/src/app/build$
```