Joshua Insorio

11.4.2018

CS 202

Papachristos

The extra credit project for CS 202 is meant to review and strengthen our ability to create and use dynamic memory wrapped in classes. I feel that this is project just adds onto what we already knew in regards to classes. So I instead built off my prior knowledge from the previous project and implemented that here.

Building upon the last project, my solution for this one was taken from the last one with a few components removed and a focus on a few of them. These components being the constructors alongside the assignment and pointer operators. So from the header code that was provided to us, which initializes the functions and methods of the class that we are intended to use, I had to create the cpp file which holds the body of the previously initialized functions and methods. At first I was unfamiliar with smart pointers and how they work, eventually found out that this is because all of the smart pointers point to the same piece of memory. Therefore, each smart pointer acts as a "doorway" to that specific piece of data.

This project did not present any hardships in particular, as it serves to strengthen what we already are familiar with. This project is not tedious nor is it difficult as dynamic memory allocation is straight forward. The main thing I had to keep an eye out for was my management of pointers and data that was being allocated. Any difficulties I had in this project was just tracking down the pointers and making sure enough space was allocated. Given more time with this project, I would explore more areas of smart pointers and see what else they can accomplish.

Testing SmartPtr Dflt-ctor

SmartPtr Default Constructor from data pointer, RefCount = 1

Dereference Smart Pointer 1: {1,0.25}

* The program is calling for the Smartptr class, and in which the default constructor is called. Because there was no parameter in the calling of the vehicle class, “SmartPtr sp1”, the default constructor was used. Upon successfully calling the constructor sp1’s intval and doubleval were initialized with values 1 and 0.25.

Testing SmartPtr Copy-ctor

SmartPtr Copy Constructor, RefCount = 2

Dereference Smart Pointer 1: {2,0.5}

Dereference Smart Pointer 2: {2,0.5}

* The copy constructor was called here with the intent to create a new SmartPtr object and initialize them with values from the sp1 object. The values were copied over because of of the operator= overload. Prior to initializing sp2 with new values, it originally had sp1 values.

Testing SmartPtr Assignment Operator

SmartPtr Default Constructor from data pointer, RefCount = 1

SmartPtr Assignment, RefCount = 3

Dereference Smart Pointer 1: {4,0}

Dereference Smart Pointer 2: {4,0}

Dereference Smart Pointer 3: {4,0}

* Similar to the copy constructor, the assignment operator was called to assign sp3, which is the new object created, with the value in sp1. Then new values for intval and double val are initialized into sp3.

Testing SmartPtr Parametrized ctor with NULL data

SmartPtr parameterized Constructor for new allocation, RefCount = 0

* At this point, the SmartPtr spNull object is created with the parameter of “NULL”. From here, a pointer to Null is kept and then directly assigned to spNull’s m\_ptr. And because m\_ptr was assigned as “NULL”, sp\_Null’s refcount pointer was assigned to 0.

Testing SmartPtr Copy ctor with NULL data SmartPtr

SmartPtr Copy Constructor, RefCount = 0

* Similar to the parameterized ctor, the copy ctor was called upon the SmartPtr spNull\_cpy with the parameter of the address of spNull. From here spNull’s members, which are m\_ptr and m\_refcount, are then copied over into spNull\_cpy’s members.

Testing SmartPtr Assignment with NULL data SmartPtr

SmartPtr Default Constructor from data pointer, RefCount = 1

SmartPtr Assignment, RefCount = 0

* The assignment operator test is assigning sp\_Null into the SmartPtr spNull\_assign object. Then spNull’s members, which are m\_ptr and m\_refcount, are then copied over into spNull\_cpy’s members.

End-of-Scope, Destructors called in reverse order of SmartPtr creation

(spNull\_assign, spNull\_cpy, spNull, sp3, sp2, sp1):

SmartPtr Destructor, RefCount = 0

SmartPtr Destructor, RefCount = 0

SmartPtr Destructor, RefCount = 0

SmartPtr Destructor, RefCount = 3

SmartPtr Destructor, RefCount = 2

SmartPtr Destructor, RefCount = 1

* All the destructors are posted here because destructors are ran at the end of the program. It follows the sequence of the constructors from the base first, then the derived class, then the destructors and finally the derived destructor.