Abraham Meza

Christos Papachristos

CS 202 – 1001

April 10, 2019

Project X Documentation/Output Explanation

**Description:**

For this project you will create your own **SmartPtr (Smart Pointer)** class. A Smart Pointer serves the purpose of wrapping a set of useful behaviors around a common Raw Pointer, such as:

1. Automatically handle allocation of Dynamic Memory if necessary, when a SmartPointer object is created.
2. Automatically handle deallocation of Dynamic Memory if appropriate, when a SmartPointer object lifetime ends.
3. Provide access to the Dynamic Memory it encapsulates (via the actual Raw Pointer) using the same notation (the same operators) as a Raw Pointer, so that it is exactly as easy to use.
4. Automatically handle cases such as a) when a Smart Pointer is used to point to the data already allocated by another SmartPointer, and avoid re-allocation, or b) when a SmartPointer’s lifetime ends but there also exists another SmartPointer pointing to the same data, and avoid deallocating early (understand when the last SmartPointer corresponding to that memory is destroyed, and only then delete the data).

Continuing through Computer Science II (202), our instructor assigned us an extra credit project that allows students to challenge themselves to utilize smart pointers through class manipulation and implementation. Similar to the previous projects, the instructor provided us with pieces of the overall code in order to lead us to the desire direction. By providing students a given header file that illustrates the structure of the class skeleton, our instructor wants students to demonstrate their abilities to implement that given format and to be able to test it through a functional test driver. The purpose of the program is to efficiently allocate and deallocate memory of a given pointer. When an object is referenced, the program should adjust its parameters to create an efficient and minimalistic environment for the object. As students continue to polish their knowledge and different implementations with classes, they are also being evaluated to be able to prove the functionality of their program through individual testing.

For my design, I referred to the header file provided, and analyzed the different functions and methods our instructor wanted. Luckily, the professor had addressed the time consuming methods and functions within his own programming files. The only header file we had to implement was the “SmartPtr”, which had minimal implementation as it addressed only constructors and overload operators. Compared to the previous projects, the instructions were very similar in the way students had to analyze and manage memory. As a result of the excessive exposure to class skeletons, I could understand the functions of each constructor well. I simply followed the instructions given and implemented the dynamic memory accordingly through the parameters within the constructor arguments. Similar to the previous project (proj7), I made sure to add additional confirmation statements throughout the code in order to trace it for future reference and debugging. As I continued to implement the constructors, I realized that each dynamic memory execution had to rely on a count variable in order to analyze the minimal space for the pointed object. In doing so, I had to monitor the memory of each object to ensure that the memory did not go below the value of 0. If the memory becomes a negative value, the system will crash, and the code will fail. As a result, I created conditional statements that would analyze the counts in order to properly delete the memory when needed.

Once I finished implementing the constructors, I moved onto the overload operations. Similar to the previous projects, the implementation of the overload operators was self-explanatory as it involved copying data of the pointed object to another. In doing so, memory would have to be modified (deallocated/reallocated) in order to have an efficient environment for the modified object. In addition, similar to the constructors, the count had to be monitored in order to prevent a misuse of the memory. As I followed the instructions given, I was able to easily implement the operators.

After finishing the “SmartPtr” program file, I moved onto working on my test driver. Similar to the previous project, I looked onto previous assignments and samples in order to construct the test driver. In doing so, I was able to structure my main file correctly, and tested my implementations with pre-determined objects. I implemented additional confirmation statements and organizers in order to clearly read the output and track the execution on the terminal screen, then went back and deleted the placeholders after the debugging process.

Subsequently after a few tests, I was satisfied with my results. Though the instructor did not provide a sample output, he was very clear to what the functionality was supposed to be for each method. By comparing the driver to the previous projects, I was able to map out and correct any possible mistakes. In doing so, I was able to reliably test my code and modify it accordingly. In addition, my confirmation statements allowed me to make sure that every specification was completed and functional.

All in all, the project was very efficient in teaching students about dynamic memory in terms of pointing to specific objects, instead of predetermined data, while allowing them to further master class manipulation and implementation through the use of their own test drivers. As a result of this project, students are now able to see the benefits of practicing memory properties and how to precisely debug their code. Overall, the project was a success and was a good example to learn from.

Project X Output and Explanation

|  |  |
| --- | --- |
| //////////////////////////////// | \*\* cout labeling design |
| ///// Constructor Tests ///// | \*\* cout labeling design (Constructor Tests) |
| //////////////////////////////// | \*\* cout labeling design |
|  |  |
| SmartPtr Default Constructor for new allocation, RefCount = 1 | \*\* cout confirmation for new smart pointer (1) (default constructor; memory allocation) |
| SmartPtr Parametrized Constructor from data pointer, RefCount = 1 | \*\* cout confirmation for new smart pointer (2) (parameterized constructor; memory allocation; parametrized by DataType (1)) |
| SmartPtr Copy Constructor, RefCount = 2 | \*\* cout confirmation for new smart pointer (4) (copy constructor; memory allocation; parameterized by SmartPtr (2); copying SmartPtr (2) data to SmartPtr (4)) |
|  |  |
| /////////////////////////////// | \*\* cout labeling design |
| ///// Assignment Tests ///// | \*\* cout labeling design (Assignment Operator Tests) |
| /////////////////////////////// | \*\* cout labeling design |
|  |  |
| SmartPtr Default Constructor for new allocation, RefCount = 1 | \*\* cout confirmation for new smart pointer (5) (default constructor; memory allocation) |
| SmartPtr Copy Assignment, RefCount = 3 | \*\* assignment argument that triggers Copy Constructor to pass SmartPtr (2) data to new SmartPtr (5) |
|  |  |
| /////////////////////////////// | \*\* cout labeling design |
| ///// Star/Arrow Tests ///// | \*\* cout labeling design (Operator [Star/Arrow] Tests) |
| /////////////////////////////// | \*\* cout labeling design |
|  |  |
| SmartPtr Parametrized Constructor from data pointer, RefCount = 1 | \*\* cout confirmation for new smart pointer (6) (parameterized constructor; memory allocation; parameterized by DataType (2)) |
| {5,10} | \*\* cout position of DataType (2) @ 5 |
| {3,10} | \*\* cout position of DataType (2) @ 3 |
|  |  |
| //////////////////////////////// | \*\* cout labeling design |
| ///// Destructor Tests ////// | \*\* cout labeling design (Destructor Tests) |
| //////////////////////////////// | \*\* cout labeling design |
|  |  |
| End-of-Scope, Destructors called in order of SmartPtr creation. | \*\* system cout that illustrates to user which order the destructors will execute |
|  |  |
| SmartPtr Destructor, RefCount = 0 | \*\* system cout confirmation that destructor was a success and that all recorded memory and data has been cleared from the program [cout count (0)] |
| SmartPtr Destructor, RefCount = 2 | \*\* system cout confirmation that destructor was a success and that all recorded memory and data has been cleared from the program [cout count (2)] |
| SmartPtr Destructor, RefCount = 1 | \*\* system cout confirmation that destructor was a success and that all recorded memory and data has been cleared from the program [cout count (1)] |
| SmartPtr Destructor, RefCount = 0 | \*\* system cout confirmation that destructor was a success and that all recorded memory and data has been cleared from the program [cout count (0)] |
| SmartPtr Destructor, RefCount = 0 | \*\* system cout confirmation that destructor was a success and that all recorded memory and data has been cleared from the program [cout count (0)] |