Nicholas Ang

CS 202 - 1101

Project X Documentation

Purpose: The purpose of this project is to utilize and allocate dynamic memory properly with the creation and use of Smart Pointers. Additionally, it uses aspects of previous knowledge such as pointers, string manipulation, iostream, along with other key aspects of C++. It also expands on overloading and conventions with dynamic memory to prevent memory leakage and segmentation faults.

Design: My project involves the use of cmake so I have a total of four folders for src, include, build, and devel. Inside the src folder is the projX.cpp source file along with two folders which respectively hold the DataType.cpp source file and the SmartPtr.cpp source file. In the include folder, there are two folders which respectively hold the DataType.h header file and the SmartPtr.h header file. Since the SmartPtr.h file, DataType.h, and DataType.cpp were given to us, I wrote the code for SmartPtr.cpp and the test driver projX.cpp. The test driver projX.cpp was coded in accordance to the instructions with minor to no change. The SmartPtr.cpp involves 3 constructors, a destructor, and multiple operator overloads. The default constructor dynamically allocates m_refcount and sets it to 1 while also setting m_ptr to a new DataType object. The parameterized constructor takes the values of the DataType *data unless data is a nullptr which causes it to set the refcount to 0 and the pointer to point at nullptr. The copy constructor copies the values of the other SmartPtr and sets them to the new object. If the other pointer is a nullptr, it sets refcount to 0 and the new object's pointer to nullptr. The destructor checks if the object is

the last of its kind and if it is, delete it. Otherwise, it decrements the refcount and prints a message. The operator overload for = checks if the object is the same as the right hand side. If it is not, it does something similar to the destructor and then copies the values of the right hand side to the expected object. If it is a nullptr, the object m_refcount is set to 0 and the pointer is pointed to nullptr. The operator overload for * returns the dereferenced pointer for m_ptr and the operator overload for -> returns the pointer m_ptr for member access.

Problems/Challenges: The project was fairly straightforward and easy to understand. However, I did have some problems with the refcount being some random value instead of being 0 for the nullptr part of the program. I solved it by fixing my destructor and my assignment operator to properly check for nullptr values and refcounts of 0. I also had a little trouble with segmentation faults but that was due to setting the object to wrong pointers or values.

Possible Changes: I would try to make my dynamic allocation simpler and make my code more simpler. I would try to make my code not as repetitive and clean it up. I would also try to experiment more with dynamic allocation and see how to use the delete and new commands more efficiently.

Testing SmartPtr Default ctor
SmartPtr Default Constructor for new allocation, RefCount = 1
Dereference Smart Pointer 1: {1,0.25}

Testing SmartPtr Copy ctor
SmartPtr Copy Constructor, RefCount = 2
Dereference Smart Pointer 1: {2,0.5}
Dereference Smart Pointer 2: {2,0.5}

Testing SmartPtr Assignment operator
SmartPtr Default Constructor for new allocation, RefCount = 1
SmartPtr Copy Assignment, RefCount = 3
Dereference Smart Pointer 1: {4,0}
Dereference Smart Pointer 2: {4,0}
Dereference Smart Pointer 3: {4,0}

Testing SmartPtr Parameterized ctor with nullptr
SmartPtr Parameterized Constructor from data pointer, RefCount = 0

Testing SmartPtr Copy ctor with nullptr SmartPtr
SmartPtr Copy Constructor, RefCount = 0

Testing SmartPTr Assignment with nullptr SmartPtr
SmartPtr Default Constructor for new allocation, RefCount = 1
SmartPtr Copy Assignment, RefCount = 0

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 = 2
SmartPtr Destructor, RefCount = 1
SmartPtr Destructor, RefCount = 0

Line By Line Analysis: The first two lines are messages which indicate a SmartPtr object was created with a refcount of 1. The third line uses the operator overload * to dereference the smartptr which was given values with an int of 1 and a double of 0.25 with functions in the DataType class. The fourth and fifth line show that the copy constructor is being used with copy

initialization and sets sp2 to sp1. The sixth line uses the operator overload * to dereference the smart pointer that was given an int of 2 and a double of 0.5 with functions from the DataType class. The seventh and eighth lines indicate that a new default constructed object was made and that it will be assigned something with the assignment operator =. The ninth line uses the assignment operator to set sp3 to sp1. Then it is given the int of 4 and the double of 0 using functions in the DataType class. The tenth, eleventh, and twelfth lines show that since all the objects sp1, sp2, and sp3 point to the same place, they have the same values and have the int value of 4 and the double value of 0. In the thirteenth and fourteenth lines, a smart pointer is created with nullptr and a refcount of 0. Then, in the fifteenth and sixteenth lines, the copy constructor is invoked and copies the values of the previously made parameterized constructed object. In the seventeenth, eighteenth, and nineteenth lines, a default constructed object is created and assigned using the overloaded assignment operator to the parameterized constructed nullptr object. The last 8 lines use the destructor in reverse order of creation to destroy the smartptr objects and free back the dynamically allocated memory.