Logan Leavitt

CS 202.1103

Christos Papachristos

Project X Documentation

The code for this project is split into files *DataType.h*, *DataType.cpp*, *SmartPtr.h* (all of which were provided), *SmartPtr.cpp*, *projX.cpp*, and a *makefile*. To compile the project, you can type "make" which produces an executable called *projX*. Typing "make clean" will remove the executable and all the object files. Upon running *projX*, you will notice several sections of output which demonstrate the functionality of the SmartPtr class. The first section of output demonstrates the SmartPtr default constructor. The default constructor allocates a new DataType object in memory as well a size_t variable kept track of through m_refcount. M_refcount is incremented to 1 as can be seen in the output of the constructor. The "->" and "*" operator overloads are also implicitly tested in this first section. "->" is used to access the DataType object pointed to by the SmartPtr, and the "*" is used to dereference and output the same object. The next section demonstrates the Copy Constructor and also changes the values of the DataType. Because the first SmartPtr was used to construct the second, no new memory was actually allocated and the two SmartPtr's both point to the same object. The third section tests the assignment operator. A default constructor is used to construct the third SmartPtr, and then the assignment operator is used to assign the first SmartPtr to the third. So even though memory is allocated in the default constructor, it is then successfully deallocated by the equality operator since it was the only SmartPtr of its kind after default construction. I then tested the SmartPtr parameterized constructor. I dynamically allocated a Datatype object, changed its values, and

then passed that pointer into the SmartPtr parameterized constructor. As can be seen, SmartPtr #4 prints different values because it points to a different object than the first three SmartPtr's. The subsequent sections test the same operations, but with NULL pointers instead. The last section of output shows all of the destructor calls as the program closes. These will deallocate memory if the current SmartPtr being destroyed is the last one of its kind. The last three refcounts shown countdown because they all point to the same DataType object, with sp1 actually deallocating the data because it is the last one to be destroyed. I ran the executable through valgrind and found no memory leaks, so all data is deallocated properly.