Output:

Testing SmartPtr Default ctor
SmartPtr Default Constructor for new allocation, RefCount = 1 //default constructor creates a new object and reference count, and sets the count to 1 because there is a pointer pointing to that object
Dereference Smart Pointer 1: {1, 0.25} //arrow operator was used on the smart pointer to access the setter methods of the object
//formatted output using the insertion operator

Testing SmartPtr Copy ctor
SmartPtr Copy Constructor, RefCount = 2 //the object made above was used to initialize this SmartPtr. Because the two pointers point to the same object, the reference count would be 2
Dereference Smart Pointer 1: {2, 0.5} //arrow operator used to change the values of the pointed object
Dereference Smart Pointer 2: {2, 0.5} //both pointers point to the same object, so any changes make will be reflected in the outputs of both pointers

Testing SmartPtr Assignment operator
SmartPtr Default Constructor for new allocation, RefCount = 1 //a default object was made first before it is assigned
SmartPtr Copy Assignment RefCount = 3 //assignment changes the object to which the left hand side points to. In this case this object now points to the same object as the first two smart pointers, so the reference count is now 3
Dereference Smart Pointer 1: {4, 0} //as discussed above, changes made to the object are reflected in all pointers
Dereference Smart Pointer 2: {4, 0}
Dereference Smart Pointer 3: {4, 0}

Testing SmartPtr Parameterized ctor with NULLdata
SmartPtr Parameterized constructor from data pointer, RefCount = 0 //initialization with a null pointer means that no object is there, and thus no references can be made

Testing SmartPtr Copy ctor with NULLdata SmartPtr
SmartPtr Copy Constructor, RefCount = 0 //because the data is null, no references can be made

Testing SmartPtr Assignment with NULLdata SmartPtr
SmartPtr Default Constructor for new allocation, RefCount = 1 //new default object was made before assignment
SmartPtr Copy Assignment RefCount = 0 //again, no references can be made to null data

End-of-Scope, Destructors called in reverse order of SmartPtr creation
(spNull_assign, spNull_cpy, spNull, sp3, sp2, sp1):

SmartPtr Destructor, RefCount = 0 //smart ptr destructors print the reference count *after* decrementing the reference count (if needed)
SmartPtr Destructor, RefCount = 0
SmartPtr Destructor, RefCount = 0
SmartPtr Destructor, RefCount = 2 //sp1, sp2, sp3 all point to the same object, so the reference count is 3 (2 after decrement)
SmartPtr Destructor, RefCount = 1
SmartPtr Destructor, RefCount = 0

This project was more or less straightforward, especially in terms of its functionality. Some of the logic was a little confusing, but I eventually got the hang of it.

A lot of trouble I had was making sure that null pointers were handled correctly and that the reference counts were correct. For the pointers, I mainly used a lot of if statements in my code to make sure each condition was handled correctly. For the most part, the main structure of the code was very simple (eg. if the pointer is valid, then deallocate/reallocate the data). I opted for using the (nothrow) new instead of using try-catch blocks mainly because I think it's a little better for debugging. Otherwise, using try-catch blocks is a little safer for use in programs.

One of the problems I had regarding the reference counts was decrementing them and destroying them in the destructor. Size_t is an unsigned type, so I had to be sure that it shouldn't equal -1 or else it'll overflow. In my destructor, I had to check that the reference count was a nonzero value before decrementing it. Other than that the usage of the reference count was also straightforward.