Milestone 5 Addendum

# Purpose and Use of this document

The purpose of this document is to collect and list common issues that are:

* Issues caused by lack of clarity in the specifications
* Issues caused by the lack of the relevant information being in the specifications at all but are present in the tester

The document also proposes possible solutions to the issues and how they can be applied.

# Issues and Solutions

## Lack of an extra blank line

One issue you may encounter while working on milestone 5 is the lack of a newline before text “----Storage and loading test,…” near the end of the ms5\_tester.cpp main program. This is caused by the presence of a newline character in the input stream buffer that isn’t ignored. The most likely location of this is from the Product.cpp’s read function after the very last data is read from the input stream. To alleviate this issue, have a ignore() call at the end of your read() function.

In line 116 of the ms5\_tester.cpp, there’s a cin.ignore(1000, ‘\n’) call which would potentially have solved the same issue. It’s uncertain currently whether that’s an intentional choice or not in the design of the tester. For the sake of not modifying the tester, I would recommend the suggested fix above.

## Printing out Products when errorState is present

The ms5\_tester.cpp has expectations that you will print out the error message of the current product object if it’s in an errorState as determined by the relevant data member. The document for MS3 and MS5 did not explicitly state this as part of the write() function.

To address this, simply add a check against if the current object is in a errorState and if it is print out the error message instead of the product’s details.

## Product Type copying/carry over with Perishables

In milestone 5, we have the Perishable objects come into play. Their product type is ‘P’. This product type can be instantiated through the use of the base Product constructor. However we when perform a read() on a Perishable, it utilizes the Product’s read() function. In that function we create a temporary product to store our read values into and upon success we then assign this temporary product to the current object. The issue comes up in that the product type of this temporary product is ‘N’ as that’s the default constructor’s behavior for products. How then should we retain the type for the Perishable object to be ‘P’.

We can go about it a few different ways, the most straight forward strategy is to perhaps save the product type of the current object and use this saved type after the copy assignment of the temporary object to the current object to set the type. An alternative approach is to not have a strict specification of the copy assignment / copy constructor so that it simply doesn’t copy the type of the products. This will allow us to retain the type of the current object as it was originally created (‘P’ or ‘N’). Different approaches also exist but the implementation is in your hands.

## Product / Date errors carry over

In the read functions of both Product and Perishable the specifications state that we should be using temporary variables to store the values of those reads. If you aren’t doing so you may encounter errors due to having set the error of the current object rather than the temporary one.

Additionally, you may experience the carry over of prior errorStates in the tester. This is due to the objects in the tester are read into multiple times. Thus if you had an error in the previous read that error will carry over into the next read. You should be clearing the error states after a successful read or alternatively clearing the current error state at the beginning of each read to avoid this carry over.