**Workshop 8: Smart Pointers**

In this workshop, you merge two lists and use a smart pointer to ensure that memory is deallocated in the possible presence of an exception.

**Learning Outcomes**

Upon successful completion of this workshop, you will have demonstrated the abilities to:

* create a program component of quadratic complexity
* use a smart pointer to move an object

**Submission Policy**

The *in-lab* section is to be completed during your assigned lab section. It is to be completed and submitted by the end of the workshop period. If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the *in-lab* portion after the period. If you do not attend the workshop, you can submit the *in-lab* section along with your *at-home* section (see penalties below). **In order to get credit for the *in-lab* portion, you must be present in the lab for the entire duration of the lab.**

The *at-home* portion of the workshop is due on the day that is four days after your scheduled *in-lab* workshop (@ 23:59:59), **even if that day is a holiday**.

All your work (all the files you create or modify) must contain your name, Seneca email, student number and the date of completion (use the following template):

// Name:

// Seneca Student ID:

// Seneca email:

// Date of completion:

//

// I confirm that the content of this file is created by me,

// with the exception of the parts provided to me by my professor.

You are responsible to back up your work regularly.

**Late Submission Penalties**

The workshop can be submitted up to **1 (one) day** late (the day that is 5 days after the lab period); submissions received on this day are considered **late** and are subject to penalties:

* only *in-lab* portion submitted late (after the end of the lab period): 0 for *in-lab* portion, max 7/10 for the entire workshop.
* only *at-home* portion submitted late (more than 4 days after the lab period): max 4 for the *at-home* portion, max 7/10 for the entire workshop.
* both *in-lab* **and** *at-home* portions submitted late: max 4/10 for the entire workshop.
* when the submission closes, if the workshop is not complete, the mark for the entire workshop will be 0/10. The workshop is considered complete if there are two separate submissions (*in-lab* submission and *at-home* submission) containing the *in-lab code*, *at-home code* and *reflection*.

The submission is considered closed at the end of the day that is 5 (five) days after the lab period.

***In-Lab***

The in-lab portion of this workshop consists of modules:

* w8 (supplied)
* Element (incomplete)
* List (incomplete)
* Utilities (supplied header, incomplete implementation)

Your tasks for this part of the workshop are:

* Add a validate() function to the Element module
* Add an operator += overload to the List module using raw pointer syntax
* Complete the mergeRaw() function in the Utilities module using raw pointer syntax

**Element Module**

The Element module holds product code, price and description information.

* The Description structure holds product code and user-friendly description.
* The Price structure holds product code and current price.
* The Product structure holds user-friendly description and product price.

**Your Task:** Add a validate() function to the Product type. Your function reports an exception if the object’s price is negative. Your function receives nothing and returns nothing.

**List Module**

The List module defines a class that retrieves a list of records stored in a text file, holds the elements in an STL vector, provides access to them by index and displays them to an output stream.

**Your Task:** Add an operator += overload to the List template. Your function receives the address of an object to be stored in the List container and moves the object into that container. Your overload returns nothing.

**Utilities Module**

**Your Task:** Implement the mergeRaw() function. This function compares the elements in the two received lists for common product codes and builds the user-friendly list from the matching pairs. For each successful comparison, your function allocates dynamic memory of Product type using that description and the price. Your function then validates the Product object and adds it to its list. Your function returns a copy of this list.

**w8 Module (supplied)**

The tester module for the in-lab portion has been supplied. **Do not modify the existing code!**

When doing the workshop, you are encouraged to write your own tests, focusing on a single implemented feature at the time, until you get all functionality in place.

**Sample Output**

When the program is started with the command (the input files are provided):

w8.exe Descriptions.dat BadPrices.dat Prices.dat

the output should look like the one from the sample\_output.txt file.

**Submission (30%)**

To test and demonstrate execution of your program use the same data as shown in the output example.

Upload your source code to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/9.1.0/bin/g++) and make sure that everything works properly.

Then, run the following command from your account (replace profname.proflastname with your professor’s Seneca userid):

~profname.proflastname/submit 305XXX\_w8\_lab

and follow the instructions. Replace XXX with the section letter(s) specified by your instructor.

**⚠️Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

***At-Home***

The *at-home* part of this workshop upgrades your *in-lab* solution to use smart pointers.

Your tasks for this part of the workshop are:

* Add an operator += overload to the List module using smart pointer syntax
* Complete the mergeSmart() function in the Utilities module using smart pointer syntax

**w8 Module (supplied)**

The tester module for the in-lab portion has been supplied. **Do not modify the existing code!**

When doing the workshop, you are encouraged to write your own tests, focusing on a single implemented feature at the time, until you get all functionality in place.

**Sample Output**

When the program is started with the command (the input files are provided):

w8.exe Descriptions.dat BadPrices.dat Prices.dat

the output should look like the one from the sample\_output.txt file.

**Reflection**

Study your final solution, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time.**

Create a **text** file named reflect.txt that contains your detailed description of the topics that you have learned in completing this particular workshop and mention any issues that caused you difficulty and how you solved them. Include in your explanation—**but do not limit it to**—the following points:

* the advantage that smart pointer syntax provides. Use example from your code to support your explanation.
* the difference between raw and smart pointer syntax in your solution.

**Quiz Reflection**

Add a section to reflect.txt called **Quiz X Reflection**. Replace the **X** with the number of the last quiz that you received and list all questions that you answered incorrectly.

Then for each incorrectly answered question write your mistake and the correct answer to that question. If you have missed the last quiz, then write all the questions and their answers.

**Submission (30% for code, 40% for reflection)**

To test and demonstrate execution of your program use the same data as shown in the output example above.

Upload the source code and the reflection file to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/9.1.0/bin/g++) and make sure that everything works properly.

Then, run the following command from your account (replace profname.proflastname with your professor’s Seneca userid):

~profname.proflastname/submit 305XXX\_w8\_home

and follow the instructions. Replace XXX with the section letter(s) specified by your instructor.

**⚠️Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.