**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 workshop is divided into two coding parts and one non-coding part:

* *Part 1*: worth 0% of the workshop's total mark, is optional and designed to assist you in completing the second part.
* *Part 2*: worth 100% of the workshop's total mark, is due on **Sunday at 23:59:59** of the week of your scheduled lab. Submissions of *Part 2* that do not contain the *reflection* are not considered valid submissions and are ignored.
* *reflection*: non-coding part, to be submitted together with *Part 2*. The reflection does not have marks associated to it, but can incur a **penalty of max 40% of the whole workshop's mark** if your professor deems it insufficient (you make your marks from the code, but you can lose some on the reflection).

The workshop should contain ***only work done by you this term*** or provided by your professor. Work done in another term (by you or somebody else), or work done by somebody else and not **clearly identified/cited** is considered plagiarism, in violation of the Academic Integrity Policy.

Every file that you submit must contain (as a comment) at the top **your name**, **your Seneca email**, **Seneca Student ID** and the **date** when you completed the work.

* If the file contains only your work, or work provided to you by your professor, add the following message as a comment at the top of the file:

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

* If the file contains work that is not yours (you found it online or somebody provided it to you), **write exactly which parts of the assignment are given to you as help, who gave it to you, or which source you received it from.** By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

**Compiling and Testing Your Program**

All your code should be compiled using this command on matrix:

/usr/local/gcc/10.2.0/bin/g++ -Wall -std=c++17 -g -o ws file1.cpp file2.cpp ...

* -Wall: compiler will report all warnings
* -std=c++17: the code will be compiled using the C++17 standard
* -g: the executable file will contain debugging symbols, allowing *valgrind* to create better reports
* -o ws: the compiled application will be named ws

After compiling and testing your code, run your program as following to check for possible memory leaks (assuming your executable name is ws):

valgrind ws

To check the output, use a program that can compare text files. Search online for such a program for your platform, or use *diff* available on matrix.

**Part 1 (0%)**

The first 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 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.

**Test Your Code**

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

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/10.2.0/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 345\_w8\_p1

and follow the instructions.

***This part represents a milestone in completing the workshop and is not marked!***

**Part 2 (100%)**

The second part of this workshop upgrades your 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 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 and the result is suggested to be at least 150 words in length.**

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.

To avoid deductions, refer to code in your solution as examples to support your explanations.

**Submission**

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/10.2.0/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 345\_w8\_p2

and follow the instructions.