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 handling 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). The *at-home* portion of the lab 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 and student number.

You are responsible to back up your work regularly.

Late Submission Penalties:

- *In-lab* portion submitted late, with *at-home* portion: **0** for *in-lab*. Maximum of 7/10 for the entire workshop.
- If any of *in-lab*, *at-home* or *reflection* portions is missing, the mark for the workshop will be **0**/10.

GENERAL SPECIFICATIONS

This workshop merges a description list with a price list to create a user-friendly price list.

The modules that constitute this workshop are listed below along with their file names

1. **Element** module**: Element.h** (incomplete)
2. **List** module**: List.h** (incomplete)
3. **Utilities** module**:**
   Utilities.h (supplied)
   Utilities.cpp (incomplete)
4. **Main** module**: w8.cpp** (supplied)

The data files are at

- **Prices.dat**

```
067 0.99
4068 0.67
4039 1.99
4056 2.49
```

- **Descriptions.dat**

```
4662 tomatoes
4039 cucumbers
4056 brocoli
4067 lemons
4068 oranges
```

- **BadPrices.dat**

```
4067 0.99
4068 0.67
4039 1.99
4056 -2.49
```

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.

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.

SPECIFICATIONS – IN LAB

This part of the workshop is a raw pointer solution.

The output from your executable running Visual Studio with the following command line argument should look like

```
Command Line : C:\Users\...\Debug\in_lab.exe Descriptions.dat BadPrices.dat
Prices.dat

 Code Description
 4662    tomatoes
 4039   cucumbers
 4056     brocoli
 4067      lemons
 4068     oranges

********************************************
* Good Prices
********************************************
 Code   Price
 4067    0.99
 4068    0.67
 4039    1.99
 4056    2.49

********************************************
* Bad Prices
********************************************
 Code   Price
 4067    0.99
 4068    0.67
 4039    1.99
 4056   -2.49

********************************************
* Merging bad prices using Raw Pointers
********************************************
     C [1]
    CC [2] from [1]
    ~D [1]
     C [3]
    ~D [2]
ERROR: *** Negative prices are invalid ***
NOTE: An exception occurred while creating the list of products!
```

```
      You should notice that not all objects are deleted.

********************************************
* Merging good prices using Raw Pointers
********************************************
 Description   Price
   cucumbers    1.99
     brocoli    2.49
      lemons    0.99
     oranges    0.67
```

The input for testing your solution is stored in the three user-prepared files listed above.

Your tasks for this part of the workshop are:

1. Add a validate function to the **Element** module
2. Add an operator += overload to the **List** module using raw pointer syntax
3. Complete the merge function in the **Utilities** module using raw pointer syntax

validate Function

Add a validate function to your **Product** type. Your function reports an exception if the object's price is negative. Your function receives nothing and returns nothing.

Operator += overload

Add an operator += overload to your **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.

mergeRaw Function

Your 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.

In-Lab Submission (30%)

To test and demonstrate 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 gcc compiler 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 345XXX_w8_lab<ENTER>
```

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

SPECIFICATIONS – AT HOME

The output from your executable running Visual Studio with the following command line argument should look like

```
Command Line : C:\Users\...\Debug\at_home.exe Descriptions.dat BadPrices.dat
Prices.dat

 Code Description
 4662    tomatoes
 4039   cucumbers
 4056     brocoli
 4067      lemons
 4068     oranges

********************************************
* Good Prices
********************************************
 Code   Price
 4067    0.99
 4068    0.67
 4039    1.99
 4056    2.49

********************************************
* Bad Prices
********************************************
 Code   Price
 4067    0.99
 4068    0.67
 4039    1.99
 4056   -2.49

********************************************
* Merging bad prices using Raw Pointers
********************************************
     C [1]
    CC [2] from [1]
    ~D [1]
     C [3]
    ~D [2]
ERROR: *** Negative prices are invalid ***
NOTE: An exception occurred while creating the list of products!
      You should notice that not all objects are deleted.

********************************************
* Merging bad prices using Smart Pointers
********************************************
     C [4]
    CC [5] from [4]
    ~D [4]
     C [6]
    ~D [6]
    ~D [5]
ERROR: *** Negative prices are invalid ***
NOTE: An exception occurred while creating the list of products!
      You should notice that ALL objects are deleted.


********************************************
* Merging good prices using Raw Pointers
********************************************
 Description   Price
   cucumbers    1.99
     brocoli    2.49
      lemons    0.99
     oranges    0.67

********************************************
* Merging good prices using Smart Pointers
********************************************
 Description   Price
   cucumbers    1.99
     brocoli    2.49
      lemons    0.99
     oranges    0.67
```

The input for testing your solution is stored in the three user-prepared files listed above.

Your tasks for this part of the workshop are:

1. Add an operator += overload to the **List** module using smart pointer syntax
2. Complete the merge function in the **Utilities** module using smart pointer syntax

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. Take your time. Explain in your own words what you have learned in completing this workshop. Include in your explanation but do not limit it to the following points (40%):

- The advantage that smart pointer syntax provides.
- List 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.

Include all corrections to the Quiz(zes) you have received (30%).

At-Home Submission (70%)

To test and demonstrate 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 gcc compiler 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 345XXX_w8_home<ENTER>
```

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