Lab 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

**SPECIFICATIONS**

This workshop merges a description list with a price list to create a user-friendly price list.  The workshop is in two parts:

1. Merging Data Sets
2. Smart Pointers

Files Provided

The three source files for this workshop are listed below

* **Element.h** (first version complete)
* **List.h** (first version complete)
* **W8.cpp** (incomplete)

The data files are at

* **Prices.dat**
* **Descriptions.dat**
* **BadPrices.dat**

Element Classes

The **Description** class holds a product code and a user-friendly description.  The **Price** class holds a product code and the current price.  The **Product** class holds a user-friendly description and the current price of a product.

List Template

The **List** template defines a class that retrieves a list of types 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.

Part 1 - Merging Data Sets

The **main()** program that uses the **Element** classes and the **List** template as seen in the provided w8.cpp file.  Your first task is to complete the coding of the **merge()** function.

merge Function

This function returns the user-friendly price list.  Each element in that list is of **Product** type.  Your function compares elements in the two received lists for common product codes and builds the user-friendly list from the matching pairs.  Your function allocates dynamic memory for each user-friendly element and adds that element to the new list by passing the object's raw pointer to the list's **+=** operator.  For this part of the workshop do not change any code in the header files provided.

Output

The output from a completed version of this program should look like:

Code Description

4662 tomatoes

4039 cucumbers

4056 brocolli

4067 lemons

4068 oranges

Code Price

4067 0.99

4068 0.67

4039 1.99

4056 2.49

Description Price

cucumbers 1.99

brocolli 2.49

lemons 0.99

oranges 0.67

Part 2 - Smart Pointers

Your second task is to introduce exception handling in the **Product** class.  **Comment out any code from the first Part** that is not needed and replace it with smart pointers as defined below.

Validate Price

Upgrade the **Product** class to include a **validate()** member function.  This function throws an exception if the stored price is a negative value.  The message thrown is shown in the output sample below.  The exception is caught by the **main()** function.

Finally, upgrade your **merge()** function to validate the product being added to the new list.  If the **Product** object throws an exception, your **merge()** function abandons building its new list and does not catch the exception.

To avoid memory leaks convert the raw pointer in your **merge()** function to a unique smart pointer.  Also, convert the raw pointer parameter in the **+=** operator in the **List** template to match.

Output

The output from a completed version of this program should look like:

Code Description

4662 tomatoes

4039 cucumbers

4056 brocolli

4067 lemons

4068 oranges

Code Price

4067 0.99

4068 0.67

4039 1.99

4056 -2.49

# \*\*\* Negative prices are invalid \*\*\*