Claude's Custom Counters, Inc. – *Object Oriented Version*

Background

Having a menu of options and the capability to load bulk data from files are features of our system that were very well received. Now our corporate executives want to see an object-oriented version. There will not be much difference to actually see, but we are going to modify much of how the software works internally. The menu, file format, data validation, and most of the Project #3 features are unchanged. How these features are implemented is significantly changed.

Software Requirements Overview

We will be adding object-oriented features to the software. Specifically, we shall define a CounterTop class that encapsulates the data and operations applicable to one of Claude's custom countertops manufactured for delivery to a customer. The complete class declaration is provided on Canvas for download. In-line implementation code is provided for several member functions. You may use this code as your own. Your CounterTop class must have all data members shown in the class declaration. You must also include all of the member functions shown. You may add member functions if they enhance your software. Any such additions must not supersede the intent and/or functionality of required member functions. Data members may not be added. If desired, you may rename data members to match your naming conventions from Project #3. However, you may find it is actually easier not to do so. For class methods, you may use different parameter names, but the number of parameters, their data type(s), and their order in the parameter lists must not be changed. You may **not** change the identifiers of any class methods. You may **not** change the return type of any class methods.

The function to clear data from all of the vectors has been removed from the project. It was marginally useful in Project #3, but does not add much to this project. We shall keep the menu option to clear the vector, but just call our one vector's clear method from main if that option is selected. All other "stand-alone" functions from Project #3 are still required for this project. We shall replace the seven vectors from last project with a single vector. This greatly simplifies the parameter lists for our "stand-alone" functions. The function to display the menu should require no modifications. All of the other functions will need to be changed. Think carefully about the design of your changes before starting to write code. Some modifications may be significant, but with a reasonable amount of prior planning, most changes should be relatively modest or largely copy-and-paste from Project #3.

File Processing

The file format is unchanged from Project #3. We will not know how many total records are in the file. Your software shall continue reading and processing lines of data until reaching the end of the input file. After reading each line of data from the file, the software shall perform the same error checking as it did in Project #3. If errors exist, that row of data is simply skipped and none of the values shall be loaded into memory (appended to the vector). Selected values of rows having errors shall still be output to the terminal window followed by a brief notice explaining the error(s) and the file processing shall continue. If there are no errors, the software shall use the data from that row to set the value of each data member of a CounterTop object. Then that CounterTop object shall be appended to a vector of CounterTop objects. As was the case before, all of the values on each row of the data file must be read even if they are not used in calculations or output. After the detailed data for each row and the summary information have been output the menu shall again be displayed along with a prompt for the user to enter their next option.

Data Validation (no change from previous projects)

Calculations

The actual calculations are unchanged from Project #3. However, where and when these calculations are made is significantly different. In this project, we shall provide CounterTop objects the capability to calculate their own area, raw materials required for fabrication, and cost information. Depending upon your design for Project #3, you may have had duplicate code to make those calculations in multiple functions (I did). Now we shall remove this duplicate code from all of the "stand-alone" functions and encapsulate it within the CounterTop class declaration along with the data necessary for the calculations to be made.

Functions

As before, all code that does "real work" will be in user-defined functions and not part of function main. Function prototypes for required functions are shown below. The purpose of each Project #4 function is essentially the same as the corresponding function in Project #3. Beneath each function prototype is an abbreviated description of the function's purpose. Tasks that are unchanged from Project #3 are not repeated. Only changes, additions, and/or deletions from tasks the function must perform are highlighted. Functions that require data from the input files should not be called unless at least one file has been successfully loaded.

```
char displayMenu();
```

This function has no parameters and returns a single character. Its purpose is unchanged from Project #3.

```
void uploadFile(string fName, bool &loadSuccess, vector<CounterTop> &vCounterTops);
```

The parameters for this function are the input data file name and path, a boolean variable to store true if the file is successfully loaded, and a vector to store one CounterTop object for each error-free line of input data. You may use the parameter identifiers shown, or change them if you prefer other parameter names. You may NOT change the function identifier or add/delete parameters. The initial steps of file processing and data validation are unchanged from Project #3 except that data are no longer appended to parallel vectors. If a row of data passes all validation checks, then values from that row shall be used to initialize or update the data members of a CounterTop object. Then that CounterTop object shall be appended to the end of the vector parameter. Maintain counters to keep track of how many rows have errors and how many rows are error-free. Output those counts after all rows of data have been processed and displayed. No other calculations are done within this function. If data from the file are successfully loaded, then the boolean parameter shall be set to true. To decide if the file data were successfully loaded, you can simply test to see if the size of the vector has increased. This is not a very rigorous test, but it is sufficient for our purposes in this project.

```
    void allDetails(const vector<CounterTop> &vCounterTops);
```

The parameter for this function is a vector of CounterTop objects. You may use the parameter identifier shown, or change it if you prefer another parameter name. You may NOT change the function identifier or add/delete parameters. The purpose of this function is to display a detailed listing of file data and calculated data. The format and content of the output is the same as Project #3. Any code that calculated area, materials required, and costing data must be removed. Invoke the appropriate member function(s) of the CounterTop class for any calculated values that are output. Test for valid data are not required, because only valid rows of data are appended to the vector.

```
    void orderDetails(const vector<CounterTop> &vCounterTops);
```

The parameter for this function is a vector of CounterTop objects. You may use the parameter identifier shown, or change it if you prefer another parameter name. You may NOT change the function identifier or add/delete parameters. The purpose of this function is unchanged from Project #3. Any code that calculated area, materials required, and costing data must be removed. Invoke the appropriate member function(s) of the CounterTop class for any calculated values that are output. Test for valid data are not required, because only valid rows of data are appended to the vector.

```
    void summaryByStone(const vector<CounterTop> &vCounterTops);
```

The parameter for this function is a vector of CounterTop objects. You may use the parameter identifier shown, or change it if you prefer another parameter name. You may NOT change the function identifier or add/delete parameters. The purpose of this function is unchanged from Project #3. Any code that calculated area, materials required, and costing data must be removed. Invoke the appropriate member function(s) of the CounterTop class for any calculated values that are output. Test for valid data are not required, because only valid rows of data are appended to the vector.

```
    void summaryByRegion(const vector<CounterTop> &vCounterTops);
```

The parameter for this function is a vector of CounterTop objects. You may use the parameter identifier shown, or change it if you prefer another parameter name. You may NOT change the function identifier or add/delete parameters. The purpose of this function is unchanged from Project #3. Any code that calculated area, materials required, and costing data must be removed. Invoke the appropriate member function(s) of the CounterTop class for any calculated values that are output. Test for valid data are not required, because only valid rows of data are appended to the vector.

Class CounterTop declaration

The complete class declaration is provided on Canvas. You may (and should) copy the declaration exactly as it is and use the code as your own without attribution. To the right is a portion of the code from the class declaration and comments about some of the key class members.

```
class CounterTop
{
   //overloaded stream insertion and extraction operators
   friend ostream& operator<<( ostream &os, const CounterTop &rhsObj );
   friend istream& operator>>( istream &is, CounterTop &rhsObj );

private:
   //the data members below are required (you may change identifiers)
   //there is no need for additional data members
   int orderYear, orderMonth, orderDay;
   int deliveryYear, deliveryMonth, deliveryDay;
   char stoneCode;
   double length;
   double depth;
   double height;
   int lengthEdgesToFinish;
   int depthEdgesToFinish;
   string orderNumber;
   string region;
   string fipsStateCode;
   string customerNameAddress;

public:
   //overloaded assignment operator
   CounterTop&  operator=(const CounterTop &rhsObj);

   //all member functions for this class are public
   //see provided code for all other member function prototypes
   //you may NOT change member function
   //identifiers or add/delete parameters
}; //END declaration class CounterTop
```

The class has a default constructor, a constructor with parameters, and a copy constructor. The default constructor should set all data members to default values such as zero for numeric data members, an empty string for string data members, etc. The constructor with parameters shall initialize all data members to the value passed to the corresponding parameter. The copy constructor shall initialize all data members to the same values stored in the existing CounterTop object passed to its parameter.

Many of the member functions for this class are simple accessor and mutator functions that are implemented in-line. The in-line implementation code is provided as part of the class declaration.

Member functions that are more involved and/or will require more than one statement must be implemented outside of the class declaration. Below is a brief explanation of the purpose of those functions.

```
void setOrderDate(int yyyy, int mm, int dd);
```

This function updates all components of the order date.

```
void setDeliveryDate(int yyyy, int mm, int dd);
```

This function updates all components of the delivery date.

```
double getCounterTopArea() const;
```

This function calculates the area of the counter top to be manufactured. It makes use of the dimension values stored in the object's data members and returns the calculated value.

```
double getMaterialRequired() const;
```

This function calculates the material required to begin manufacturing. It should call the function above that calculates the counter top area and use that value along with the 26% wastage factor to calculate the quantity required. The final result is returned to the calling function.

```
double getStoneCost() const;
```

This function calculates the cost of stone for the counter top to be manufactured. It must use the value of the object's stone code data member to determine the cost per square foot. Then it must call the function that calculates material required to begin fabrication. Using those values the function calculates its final result and returns that value to the calling function.

```
double getFinishingCost() const;
```

The function above uses the object's data members that store length edges to be finished, depth edges to be finished, length, depth, and the finishing cost value to calculate the finishing cost for a counter top (the latter value (finishing cost) should be a global constant). That calculated value is returned to the calling function.

```
double getTotalCost() const;
```

This function calls the function that calculates stone cost, as well as, the function that calculates finishing cost. Those two values are added together and returned as the total cost.

```
friend ostream& operator<<( ostream &os, const CounterTop &rhsObj );
```

This function is the overloaded stream insertion operator for the CounterTop class. In lecture we will discuss the motivation for operator overloading in user defined classes. For this project, the overloaded stream insertion operator must be used in the allDetails function and in the orderDetails function. Passing a CounterTop object as the right hand side operand to the overloaded stream insertion operator results in its data members and/or calculated values (calculated via calls to its member functions) being output in the exact format as is currently displayed in those two functions.

```
friend istream& operator>>( istream &is, CounterTop &rhsObj );
```

This function is the overloaded stream extraction operator for the CounterTop class. In lecture we will discuss the motivation for operator overloading in user defined classes. Passing a CounterTop object as the right hand side operand to the overloaded stream extraction operator results in its data members being updated with values extracted from the stream object that is the left hand side operand. For this project, the overloaded stream extraction operator must be implemented and tested. However, you are not required to use the operator if it is not convenient for your specific project design.

You may **not** add data members to the class declaration. The data members available in the class should be sufficient for everything a CounterTop object requires. You may **not** change member function identifiers or add/delete parameters. You do have the option to add more member functions. However, no added member functions are allowed to replace functionality of the required member functions. If you add a member function it must remain consistent with the overall project intent and should enhance your program design. Below is an example of an optional function that I added to my program.

```
string getStoneName() const;
```

This function uses the object's stone code data member. The function "translates" that value into an actual stone name. In my program, I would call this function any time I needed to output a descriptive name instead of just the single character code for a CounterTop object.

Academic Integrity

This is an individual project and all work must be your own. Refer to the guidelines specified in the *Academic Honesty* section of this course syllabus or contact me if you have any questions.

Include the following comments at the start of your source code file:

```
/*
 *  main.cpp
 *
 *  COSC 051 Fall 2018
 *  Project #4
 *
 *  Due on: NOV 21, 2018
 *  Author: <your NetID>
 *
 *
 *  In accordance with the class policies and Georgetown's
 *  Honor Code, I certify that, with the exception of the
 *  class resources and those items noted below, I have neither
 *  given nor received any assistance on this project.
 *
 *  References not otherwise commented within the program source code.
 *  Note that you should not mention any help from the TAs, the professor,
 *  or any code taken from the class textbooks.
 */
```

These comments must appear **exactly** as shown above. The only difference will be values that you replace where there are "place holders" within angle brackets such as <your netID> which should be replaced by your own netID and the angle brackets removed.

Submission Details

Upload to Canvas a .cpp file containing your C++ program. In general, requests for extensions will not be considered. Late submissions will be penalized 2.5 points for each 15 minutes after the due date. Files submitted more than 10 hours late will receive a score of zero. Do **not** post your executable file. You should ensure that your source file compiles **on the cs-class server** and that the executable file runs and produces the correct output **on the cs-class server**. Use main.cpp as the name for your source code file.

Programming Skills

The programming skills required to complete this assignment include:

- Screen output (cout)
- Keyboard input (cin)
- Basic data validation
- Basic output formatting
- Basic calculations
- Control structures for repetition
- Advanced output formatting
- Control structures for repetition
- Advanced output formatting
- Tabulated output
- Advanced data validation
- Menu driven programs
- Vectors
- Functions
- **Object Oriented Programming**
- **Operator Overloading**
- **Copy Constructor**

How to approach this program

For this project several milestones are provided. You are NOT required to turn anything in or to meet these milestones. Make sure that your code compiles and runs prior to moving on to the next milestone.

Design and Test Plan Idea Sketch:

Milestone 1: Days 1 - 2

- Create an empty source code file; insert heading comments (copy and paste from Canvas, edit as applicable), add preprocessor directives, add using namespace std;
- Add a "skeleton" of function main()
- Add global constants that you plan to reuse from Project #3

Milestone 2 – Days 2 - 4

- Study the project description and bring questions to class
- Copy the provided CounterTop class declaration and paste it into your project
- Write function "stubs" for the class constructors
- Write function "stubs" for any other methods that are not implemented in-line

Milestone 3 – Days 4 - 6

- Copy your control structure for selection (switch or if / else if) from Project #3 and paste it into your function main()
- Delete (or comment out) the function calls since the functions are going to change and you will want to implement and test them one at a time.
- Copy your function to display the menu from Project #3 and past it into the new project
- Add code to call the display menu function and test to ensure it is still working

Milestone 4 – Days 6 - 8

- Write your implementation code for the CounterTop class default constructor
- Write your implementation code for the CounterTop class constructor with parameters
- Write your implementation code for the CounterTop class copy constructor
- Write your implementation code for the other CounterTop class member functions
- Write your implementation code for the CounterTop class non-member functions
- Perform unit testing
  - Instantiate a CounterTop object using the default constructor
  - Instantiate a CounterTop object using the constructor with parameters
  - Instantiate a CounterTop object using the copy constructor
  - Invoke each object's accessor functions to output data member values to the screen and verify that the values are correct
  - Invoke each object's mutator functions to update the data members and output the new values to the screen to verify that changes were made
  - Invoke each object's methods that calculate area, materials required, and costing data; verify that the calculated values are correct
  - Invoke each object's overloaded stream insertion operator and verify the output is correct

   - Invoke each object's overloaded stream extraction operator and verify data members are accurately updated

Milestone 5 – Days 10 - 11

- Modify the upload function
  - Remove code that used parallel vectors
  - Remove calculations (area, materials required, costing data) those should all be calculated in CounterTop class methods now
  - Add code to instantiate a CounterTop object for each valid row of file data, initialize the object's data members with that row of data, append the object to the vector of CounterTop objects passed in to the reference parameter

Milestone 6 – Days 12

- Modify the allDetails function
- Modify the orderDetails function
- Modify the summaryByStone function
- Modify the summaryByRegion function

Milestone 7 – Days 12 - 14

- Perform integration testing
- Copy to the server and verify that your code compiles and runs in that environment
- Submit your project early!

Grade Rubric

	Grade Standards - Missing: 0%, Poor: up to 50%, Fair: up to 67%, Good: up to 82%, Excellent: up to 99%, Perfect: 100%		
	Detailed Rubric (Code)	**100.00**	**<-- TOTAL**
1	**Code Quality and Formatting**	***14.00***	**<--sub total**
	proper indentation		
	good variable and constant names		
	good use of constants (no "magic numbers" in calculations)		
	good use of comments		
	good use of vertical white space to separate code		
	good use of horizontal white space to improve readability		
	line length less than 100 characters		
2	**User interface / data input / file i/o**	***16.00***	**<--sub total**
	menu is displayed with appropriate options		
	menu repeats until valid entry		
	if search is requested an no file is loaded. Error message is displayed		
	after processing, menu is re-displayed.		
	file open is tested.		
	if file does not open. Print error message and re-display menu		
	file is openned and closed correctly		
	each record is read and processed correctly		
	all prompts are neatly formatted		
3	**functions**	***20.00***	**<--sub total**
	function name, return type, and parameter list match the provided prototype		
	opens the file passed in as an argument to the function call and associates it with a stream prior to being used		
	if any input data fail validation error message(s) are displayed		
	will correctly process a file of any length (containing any number of data rows)		
	data are stored in appropriate objects		
	parameter lists are updated to contain objects		
4	**Classes and OOP**	**35.00**	**<--sub total**
	best practices for oop are observed		
	includes all members specified in the provided class declaration, all member functions are correctly implemented		
	default constructor initializes data members to reasonable values (e.g. empty string, 0, 0.0) as appropriate for the member's data type		
	constructor with parameters correctly initializes data members with values passed in as arguments		
	accessor functions for which implementation code has not been provided are correctly implemented, perform the required calculations, and return accura		
	copy constructor correctly initializes the data members of the CounterTop object being created to the same values of the previously existing object that is		
	the overloaded stream operators are correctly implemented and is used as instructed		
5	**display reportts**	**15.00**	**<--sub total**
	results are displayed correctly and neatly		
	clear options works correctly		
	Common Deductions (Code)		
	Program does not compile ON THE CLASS SERVER (deduction varies depending on how bad, value listed is max)	-100.00	
	Program compiles but has warnings ON THE CLASS SERVER (deduction varies depending on how bad, value listed is max)	-100.00	
	Program crashes during execution ON THE CLASS SERVER (deduction varies depending on how bad, value listed is max)	-100.00	
	Code uses any global variables	-40.00	
	Filename does not follow conventions specified	-20.00	
	Required comments and honor statement not included at start of file exactly as specified	-30.00	
	Late penalty for each 15 minutes late	-2.50	