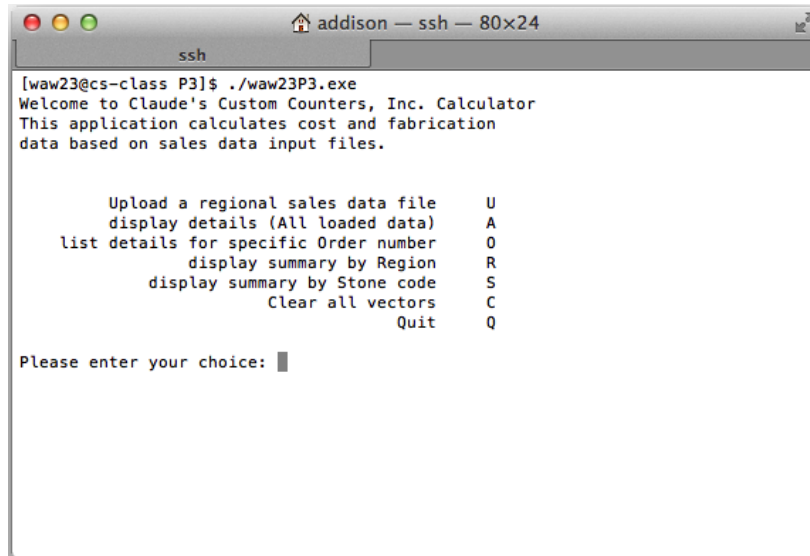


Claude's Custom Counters, Inc. - *Menu Driven Version*

Background

The bulk data version of your software was very well received. There have been some business process changes that will require software modifications. The headquarters will no longer consolidate regional sales files so we will need to load a separate file from each region. Additionally, Claude would like to have a menu-driven application that can perform multiple operations without exiting the program. Shown below is a screen capture of the required menu options.

A screenshot of a terminal window titled 'addison — ssh — 80x24'. The terminal shows the execution of './waw23P3.exe' which displays a welcome message and a menu of options. The menu options are: 'Upload a regional sales data file' (U), 'display details (All loaded data)' (A), 'list details for specific Order number' (O), 'display summary by Region' (R), 'display summary by Stone code' (S), 'Clear all vectors' (C), and 'Quit' (Q). The prompt 'Please enter your choice:' is followed by a cursor.

```
[waw23@cs-class P3]$ ./waw23P3.exe
Welcome to Claude's Custom Counters, Inc. Calculator
This application calculates cost and fabrication
data based on sales data input files.

      Upload a regional sales data file      U
      display details (All loaded data)     A
      list details for specific Order number O
      display summary by Region             R
      display summary by Stone code         S
      Clear all vectors                     C
      Quit                                  Q

Please enter your choice: █
```

Software Requirements Overview

When the application is executed, the menu of options above shall be displayed to the terminal window. The options must be in the order shown and the character corresponding to each option shall be exactly as shown. When the user types in a choice and presses enter, their entry must be tested to decide what action to take. If they entered a letter corresponding to one of the menu options then the appropriate function shall be called. If they did not enter a valid option, then a warning message should be displayed and the menu again presented. The function that is called must contain all code necessary to accomplish its requirements as described below. Once a function completes processing, control shall be returned to function main and the menu of options shall be displayed again. There must be a separate function for each menu option (except for Quit). A function prototype for each required function is provided below. You must use the function prototypes specified and each prototype must have the parameters shown in the order shown. You may add other functions to streamline your code or otherwise enhance your solution, but do not replace the functionality of any required function.

There is no design part for this project. You are strongly encouraged to make a design of your own. It does not need to be a formal design. There is a lot of opportunity to reuse code from Project #2. Plan carefully to determine what you can use from the last project and where you will place that code in the current project.

File Processing

If the user selects menu option 'U' (or 'u'), code in function main shall prompt the user to enter the path and name of the file to load. After the file name and path have been entered, a call to the load function shall be made. The

load function shall open the data file and process all rows of data. As always, the software must not attempt to process the file if it fails to open.

The first line of the file contains column headings. These are for anyone reading the file manually. We will not use these column headings and only need to read the entire line to "get it out of the way". The second line of the file is the first record of data that we will process. We will not know how many total records are in the file. We simply must continue reading and processing lines of data until reaching the end of the input file. Each line of the file contains the following data elements (**Note that there is a new string field for the sales region**):

Order Date	Date of order (string with format yyyy/mm/dd)
Delivery Date	Date of counter due to customer (string with format yyyy/mm/dd)
Stone Code	Stone code (a single character)
length	counter length (floating point number)
depth	counter depth (floating point number)
height	counter height (floating point number)
length edges finished	number of length edges polished/finished (an integer)
depth edges finished	number of depth edges polished/finished (an integer)
Order Number	Order Number (string, no spaces)
Region	<i>South, East, North, West, or Other</i> (a string, no spaces)
FIPS State Code	Federal Information Processing Standards State Code
	(string, no spaces)
Customer name & address	Customer's full name and address (string with spaces)

After reading each line of data from the file, the software shall perform the same error checking as it did for Project #2. If errors exist, that row of data is simply skipped and none of the values shall be loaded into memory (appended to the vectors). The values of that row shall still be output to the terminal window followed by a brief notice explaining the error and the file processing shall continue. If there are no errors, the software shall append data values from that row to the appropriate parallel vectors, as well as outputting the values to the terminal. You will need two vectors that store string values (*order number* and *region*). You will need one vector to store character values (*stone code*). You will need three vectors to store floating-point values (*length*, *depth*, and *height*). You will need two vectors to store integer values (*length edges to finish* and *depth edges to finish*). You may use any identifiers that you want for the vector objects. However, you must implement all eight vectors and use them to store file data and perform calculations. All other values in each row need to be read. Some are necessary for validation checks. However they will not be stored in vectors.

After the detailed data for each row and the summary information have been presented, 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

Calculations are essentially unchanged from Project #2. There are some new calculations required for the summary table by region, but they are analogous to the calculations for summary by stone code.

Functions

All code that does "real work" must be moved out of function main and placed in user-defined functions. Function prototypes for required functions are shown below. Also below is an explanation of what each function must accomplish.

```
char displayMenu();
```

This function has no parameters and returns a single character. The purpose of the function is to present the menu of options to the user, store the user's choice, and return that value to the calling function. The function shall continually display the menu of options until such time as the user enters a valid value. You may add a "maximum attempts" counter to prevent the possibility of endless incorrect user entries.

```
void uploadFile(string fName, bool &loadSuccess, vector<char> &sCode,
               vector<double> &len, vector<double> &dep, vector<double> &hei,
               vector<int> &lenF, vector<int> &depF,
               vector<string> &oNum, vector<string> &reg);
```

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 (false otherwise), and vectors to store selected values from the input data file. The identifiers above are abbreviated to save space (sCode – stone code, len – length, dep – depth, hei – height, lenF – length edges finished, depF – depth edges finished, oNum – order number, reg – region). You may use these identifiers or change them if you prefer other parameter names. This function has several important tasks. First it must open the input data file and test to ensure it opened. If the file opened successfully, then the function shall process the file contents. It must read and ignore the column headings. Then it must use a loop to read all data rows in the file. For each row, the function must validate the values according to the same validation rules from Project #2. The function shall output selected values from the data file and error information, as did the load routine in Project #2. If a row of data passes all validation checks, then values from that row shall be appended to the appropriate vector. 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

```
void allDetails(const vector<char> &sCode,
               const vector<double> &len, const vector<double> &dep, const vector<double> &hei,
               const vector<int> &lenF, const vector<int> &depF,
               const vector<string> &oNum, const vector<string> &reg);
```

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 was successfully loaded, you can simply test to see if the size of any vector has increased. This is not a very rigorous test, but it is sufficient for our purposes in this project.

The parameters for this function are the vectors that store data from all loaded files. The identifiers are abbreviated as described earlier. You may change the parameter identifiers if you wish. The purpose of this function is to display a detailed listing of file data and calculated data. This should look essentially the same as the detailed output when the file is being read (with calculated output added). There are no dates to output (since we do not store them in any vector). Your main program should only call this function if at least one file has been successfully loaded. Test for valid data are not required since only valid rows of data are appended to the vectors.

```
void orderDetails(const vector<char> &sCode,
                 const vector<double> &len, const vector<double> &dep, const vector<double> &hei,
                 const vector<int> &lenF, const vector<int> &depF,
                 const vector<string> &oNum, const vector<string> &reg);
```

The parameters for this function are the vectors that store data from all loaded files. The identifiers are abbreviated as described earlier. You may change the parameter identifiers if you wish. The purpose of this function is to display a detailed listing of file data and calculated data. The output is the same as the `allDetails` function. However, this function must first prompt the user for an order number. Only data for that specific order number shall be displayed. Your main program should only call this function if at least one file has been successfully loaded. Test for valid data are not required, because only valid rows of data are appended to the vectors. Output an appropriate message if no match is found.

The parameters for this function are the vectors that store data from all loaded files. The identifiers are

```
void summaryByStone(const vector<char> &sCode,
                   const vector<double> &len, const vector<double> &dep, const vector<double> &hei,
                   const vector<int> &lenF, const vector<int> &depF,
                   const vector<string> &oNum, const vector<string> &reg);
```

abbreviated as described earlier. You may change the parameter identifiers if you wish. The purpose of this

```
void summaryByRegion(const vector<char> &sCode,
                    const vector<double> &len, const vector<double> &dep, const vector<double> &hei,
                    const vector<int> &lenF, const vector<int> &depF,
                    const vector<string> &oNum, const vector<string> &reg);
```

function is to display a summary table by stone code. Your main program should only call this function if at least one file has been successfully loaded. Test for valid data are not required, because only valid rows of data are appended to the vectors.

The parameters for this function are the vectors that store data from all loaded files. The identifiers are abbreviated as described earlier. You may change the parameter identifiers if you wish. The purpose of this function is to display a summary table by region. This should basically look like the summary table that is organized by stone code, but the totals are calculated and displayed for each region. Your main program should only call this function if at least one file has been successfully loaded. Test for valid data are not required, because only valid rows of data are appended to the vectors. Values for region are South, West, East, North, and Other. All values in the data files begin with a capital letter. You do not need to account for any other capitalization or variations. Just match the five values shown above.

```
bool clearAllVectors(vector<char> &sCode,
                    vector<double> &len, vector<double> &dep, vector<double> &hei,
                    vector<int> &lenF, vector<int> &depF,
                    vector<string> &oNum, vector<string> &reg);
```

The parameters for this function are the vectors that store data from all loaded files (passed by reference). The identifiers are abbreviated as described earlier. You may change the parameter identifiers if you wish. The purpose of this function is to empty all the data that have been loaded in the vectors. It should return true if the operation is successful, otherwise it should return false.

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:

```
/*
 * <FileName>.<file extension>
 *
 *  COSC 051 <term year>
 *  Project #3
 *
 *  Due on: <Due Date>
 *  Author: <your name>
 *
 *
 *  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 <netID> which should be replaced by your own netID. For example, I would replace <netID>P3.cpp with waw23P3.cpp.

Submission Details

Upload (as instructed by your professor) a .cpp file containing your source code. Do **NOT** post your executable file. You should ensure that **your source file compiles on the server** and that the executable file runs and produces the correct output. Use the following file name for your file: <netID>P1.cpp. Late submissions will be penalized heavily – see rubric for details. If you are late you may turn in the project to receive feedback but the grade may be zero. In general, requests for extensions will not be considered.

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**

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	26.00	<--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 Data validation algorithms	16.00	<--sub total
all input data are validated to ensure they are valid and/or within limits		
no abnormal exits		
if any input data fail validation error message(s) are displayed		
for all "non-fatal" data validation errors, processing continues, but if a row contains one or more errors, then that row of data is eliiminated from calculations		
data are stored in appropriate vectors		
file is parsed correctly		
4 Search and Display	22.00	<--sub total
Search execute correctly		
displays are neat		
summary by region, material, etc are correct		
5 Functions	22.00	<--sub total
code is modularized into functions		
format, definition, prototypes, design, and syntax of functions are correct		
functionality of functions are correct		
correct return values		
correct parameter lists		
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	