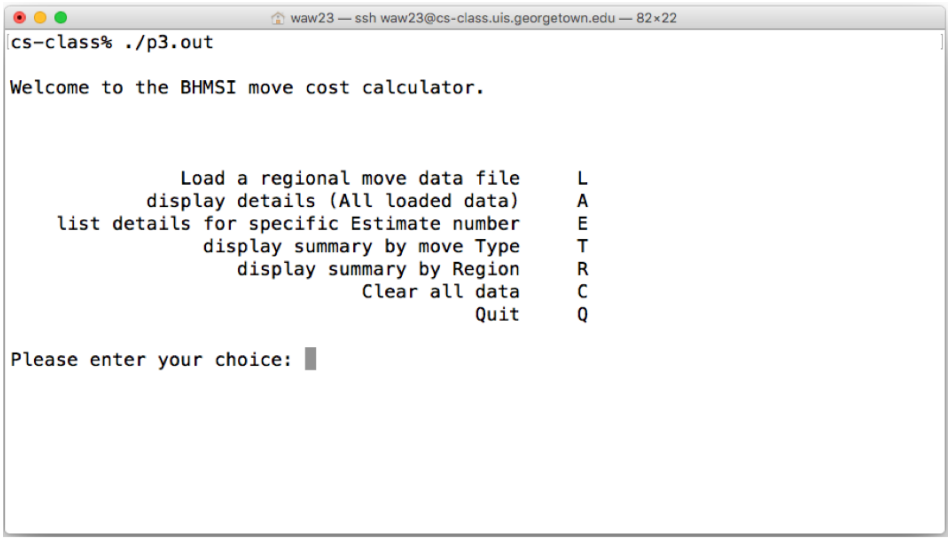
Bubba Hotep Moving and Storage, Inc. (BHMSI) - Menu Driven Version

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, Bubba would like to have a menu-driven application that can perform multiple operations without exiting the program. Shown below is what the required menu should look like.



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 shall 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 Clear and 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 '`L`' (or '`l`'), 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 are new string fields for the sales state and region**):

```
Estimate Number           Unique ID (not really a number, but a string, no spaces)
Estimate Date             Date customer entered data (string with format yyyy/mm/dd)
Move Date                 When customer wants to move (string with format yyyy/mm/dd)
Move Type                 One of 3 choices (a single character)
Distance                  Driving distance from origin to destination, in miles (integer)
Weight                    Weight of contents to be moved, in pounds (integer)
Pianos                    Count of pianos to be moved (integer)
Stairs at origin          More than 15 stairs at the origin (a single character)
Stairs at destination     More than 15 stairs at the origin (a single character)
State                     Two-character state abbreviation (string, no spaces)
Region                    Region of the country (string, no spaces)
Customer name & email     Customer's full name and email (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). Selected values of that row 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 append data values from that row to the appropriate parallel vectors, as well as outputting additional values to the terminal. You will need two vectors that store string values *(estimate number* and *region).* You will need three vectors to store character values *(*one for *move type* and two for the *stairs answers).* You will need three vectors to store integer values *(distance*, *weight* and *number of pianos).* 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 move type.

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 loadFile(string fName, bool &loadSuccess, vector<char> &vTyp,
                vector<int> &vDst, vector<int> &vWgt, vector<int> &vPno,
                vector<char> &vSO, vector<char> &vSD,
                vector<string> &vNum, vector<string> &vReg);
```

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 (`vTyp` – move type, `vDst` - distance, `vWgt` – weight of items being moved, `vPno` – number of pianos, `vSO` – stairs > 15 at origin, `vSD` – stairs > 15 at destination, `vNum` - estimate number, `vReg` - 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. If data from the file are successfully loaded, then the `boolean` parameter shall be set to `true`. To decide if the data were 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.

```
void allDetails(const vector<char> &vTyp,
         const vector<int> &vDst, const vector<int> &vWgt, const vector<int> &vPno,
         const vector<char> &vSO, const vector<char> &vSD,
         const vector<string> &vNum, const vector<string> &vReg);
```

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 estimateDetails(const vector<char> &vTyp,
         const vector<int> &vDst, const vector<int> &vWgt, const vector<int> &vPno,
         const vector<char> &vSO, const vector<char> &vSD,
         const vector<string> &vNum, const vector<string> &vReg);
```

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 estimate number. Only data for the move having that estimate 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.

```
void summaryByType(const vector<char> &vTyp,
         const vector<int> &vDst, const vector<int> &vWgt, const vector<int> &vPno,
         const vector<char> &vSO, const vector<char> &vSD,
         const vector<string> &vNum, const vector<string> &vReg);
```

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 type of move (similar to the summary table from Project #2). 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.

```
void summaryByRegion(const vector<char> &vTyp,
         const vector<int> &vDst, const vector<int> &vWgt, const vector<int> &vPno,
         const vector<char> &vSO, const vector<char> &vSD,
         const vector<string> &vNum, const vector<string> &vReg);
```

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 type, 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.

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 Spring 2019
 *  Project #3
 *
 *  Due on: March 21, 2019
 *  Author: <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 `<netID>` which should be replaced by your own `netID`. For example, I would replace `<netID>` on the "Author:" line with `waw23`.

Submission Details

Post to Canvas a `.zip` file containing your source code and the given Makefile. Locate the assignment `Project 3` on Canvas and attach/upload your file. 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: `submit.zip`. The code part of this project is due by end-of-day (11:59pm) on March 21st. Late submissions will be penalized 2.5% for each 15 minutes late. If you are over 10 hours late you may turn in the project to receive feedback but the grade will be zero. In general requests for extensions will not be considered. The value for this project is 100 points.

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
- File input/output
- Advanced output formatting
- Tabulated output
- Advanced data validation
- **Menu driven programs**
- **Vectors**
- **Functions**

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.

Milestone 1 – NLT March 1

- Create an empty source code file; insert heading comments (copy and paste from Blackboard, 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 #2

Milestone 2 – NLT March 1

- Add function prototypes
- Add definitions for the eight required parallel vectors
- Add function stubs for all required functions

Milestone 3 – NLT March 12

- Add implementation code for the function to display the menu, prompt for the user's choice, and return the value of that choice
- Add the loop to function `main` that repeatedly calls the display menu function and stores the value of the user's selection that is returned from that function, the loop repeats until the user elects to quit
- Add code to process the user's selection, since the menu function returns a `char`, this would be a good place to use a `switch` structure
- If your function stubs are in place from Milestone 2, then function calls can be put where they go in to the `switch` structure even though the function implementation code is not yet written

Milestone 4 – NLT March 14

- Implement function to load input data file(s) and store data for valid records in the parallel vectors

Milestone 5 – NLT March 16

- Implement function to display details for all orders
- Implement function to display details for a specific order number

Milestone 6 – NLT March 19

- Implement function to display summary by type
- Implement function to display summary by region
- Implement function to clear all vectors

Milestone 7 – NLT March 20

- Complete final testing and verify operations on the server
- Submit the project **prior** to the due date/time

Grade Rubric

A detailed grade rubric and list of common deductions will be published separately

Course Materials Notice

The materials used in Georgetown University courses ("Course Materials") generally represent the intellectual property of course instructors which may not be disseminated or reproduced in any form for public distribution (e.g., sale, exchange, etc.) without the written permission of the course instructor. Course Materials include all written or electronic documents and materials, including syllabi, current and past examination questions/answers, and presentations such as lectures, videos, PowerPoints, etc., provided by a course instructor. Course Materials may only be used by students enrolled in the course for academic (course-related) purposes.

Published course readings (book chapters, articles, reports, etc.) available in Canvas are copyrighted material. These works are made available to students through licensed databases or fair use. They are protected by copyright law, and may not be further disseminated or reproduced in any form for distribution (e.g., uploading to websites, sale, exchange, etc.) without permission of the copyright owner.

More information about intellectual property and copyright can be found here:

https://www.library.georgetown.edu/copyright.

More information about computer acceptable use policy and intellectual property can be found here:

https://security.georgetown.edu/it-policies-procedures/computer-systems-aup