

## Claude's Custom Counters, Inc. – *Linked List Version*

### Background

The latest version of our software has been a great success. For now, we are free from the need to release another version. However, engineering is on notice that, among other enhancements, we must use a linked list, instead of a vector, to store the CounterTop objects. We will begin immediately to make that modification. This project does not need to be a complete system for release. Therefore, we will do away with some of the Project #4 features. This will enable us to focus more closely on the linked list and memory management. The menu and some other functions are omitted. The file format and data validation requirements are unchanged.

### Software Requirements Overview

With the exception of the menu and functions that are removed, most of the other software requirements are the same as for Project #3 and Project #4. Since there is no menu, the full path and name of the input data file shall be passed to the application as a **command line argument**. A command line argument is anything typed in the terminal window after typing the name of the executable file and before pressing the enter key. If you are developing your projects using an Integrated Development Environment (IDE), there should be a way to "simulate" passing command line arguments to the applicatio. Read the docs for your specific IDE if you prefer to test on your IDE as opposed to server. Your application must test to determine how many command line arguments were actually passed. If there are less than two (the executable name itself counts as one), output an error message and do not perform any other processing. Any command line argument (after the first) shall be assumed to be the full path and name of an input data file. Your program shall open and process the contents of the input data file passed as a command line argument. Processing means to open the input data file and associate it with a stream object. If the file opens successfully, then all rows shall be read, validated, and valid rows subsequently added to a linked list of CounterTop objects.

### File Processing

The file format is unchanged from Project #4. 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 Project #4. If errors exist, that row of data is simply skipped and none of the values shall be loaded into memory (added to the linked list). 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 instantiate a CounterTop object using the data from that row to initialize the data members of the object. A linked list data structure shall replace the vector that we used in Project #4. Therefore, memory to store each CounterTop object must be dynamically allocated. Then the CounterTop object shall be appended to the end of the linked list. Alternatively, the CounterTop object may be inserted at the front of the list. Appending objects to the end results in a linked list in the same order as the input data; inserting at the front results in the opposite order. Either option is acceptable for this project.

### Data Validation (no change from previous projects)

### Calculations (no change from previous projects)

## Functions

The following functions from Project #4 have been removed:

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

All other functions have been modified to work with a linked list instead of a vector. Otherwise, each function has the same purpose as it did in Project #4. The modified function prototypes are shown below along with a brief description of the function's purpose. There is also a new function listed below that was not part of Project #4.

```
void uploadFile(string fName, bool &loadSuccess, unsigned long &objCount, CounterTop* &llCtop);
```

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, an unsigned long to store the number of objects loaded, and a pointer to a CounterTop object. Function uploadFile will change the values stored in loadSuccess, objCount, and llCtop so each of those variables is passed by reference. Similar to Project #4, loadSuccess should be set to true if the number of objects on the linked list is larger when the function ends than it was when the function first began. The variable objCount shall be incremented each time another object is added to the linked list. The variable llCtop represents one of the fundamental differences between this project and Project #4. We are no longer storing objects on a vector. Instead, we shall use a linked list data structure to store our list of CounterTop objects. In the same way that vectors increase in size at runtime, our linked list must also increase in size, as necessary, while the program is running. The variable llCtop is a pointer to a CounterTop object. Specifically, llCtop points to the first CounterTop object on the linked list. Function uploadFile is largely unchanged from Project #4, except that it adds CounterTop objects to the linked list pointed to by llCtop (instead of pushing objects onto a vector). This requires dynamically allocating memory at runtime for each object that is added to the linked list. Your code may append objects to the end of the linked list or insert objects at the front of the list.

```
void clearLL( unsigned long &objCount, CounterTop* &llCtop );
```

This new function is critical to Project #5 as it has the task of deallocating all memory that was dynamically allocated for the linked list of CounterTop objects. Your implementation code should traverse the linked list pointed to by llCtop to deallocate memory for each object on the list. As each object is deleted, the objCount variable shall be decremented. When the function ends, assuming it executed completely and correctly, all objects that were on the linked list will have been deleted, llCtop will have been updated to point to NULL, and the objCount variable will have been decremented until its value is zero.

```
void summaryByStone( unsigned long objCount, CounterTop* llCtop );
void summaryByRegion( unsigned long objCount, CounterTop* llCtop );
```

The two summary report functions are unchanged from Project #4, except that they must traverse the linked list of CounterTop objects pointed to by llCtop instead of iterating through the elements of a vector.

## Demonstrating the Functionality of Your Code

To demonstrate the functionality of your program, add code to function main that calls the uploadFile function. Then call the summaryByStone function and the summaryByRegion function. Finally call the clearLL function. The first function call results in a linked list of CounterTop objects. The other functions will traverse the linked list and produce output that should match the sample program.

## 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 #5
 *
 *  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>P5.cpp with waw23P5.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.

## Grade Rubric

See supplemental 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
- Functions
- Object Oriented Programming
- Operator overloading
- **Self-referential classes**
- **Pointers**
- **Dynamic memory allocation**
- **Unit testing / integration testing**

## 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 – Days 1-3

- Study the project description and bring questions to next class meeting
- Register for the Project #5 forum
- 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 #4
- Copy and paste your CounterTop class declaration from Project #4, make additions and changes as specified in the project description
- Copy and paste your implementation code from Project #4, for the CounterTop class member functions, edit the heading of the constructor with parameters, there is one additional parameter for P5, otherwise, the code should compile (you do not need to implement the destructor or overloaded assignment at this time, **but you must add function stubs for these two functions**)
- Compile and run (nothing much should happen, this is just a check for compiler errors)
- Copy and paste the code for Milestone 1, Test Suite 1 into your function main(), run the test

### Milestone 2 – Days 3 - 4

- Modify your project to use command line arguments
- If you are using an IDE edit your project such that either OrdersP3All.dat or tOrdersP3All.dat is passed as a command line argument
- If you are using the server for development, then you actually type the file name on the command line after the executable file name and before pressing enter
- Copy and paste the code for Milestone 2, Test Suite 1 into your function main(), run the test

**Milestone 3– Days 5-6**

- Add implementation code for CounterTop class member functions getNext and setNext
- Add the pointer variable CounterTop \*llCounterTopList = NULL to function main (this pointer will keep track of the linked list that is replacing the vector from Project #4)
- Add an unsigned long variable to function main to store the size of the linked list (that is, the number of CounterTop objects that are on the linked list, this replaces the functionality of the size() function that was provided by the vector class in Project #4)
- Add the prototype for function clearLL and write your implementation code for this new function
- Add provided code for the class destructor
- Copy and paste the code for Milestone 3, Test Suite 1 into your function main(), run the test

**Milestone 4 – Days 7-8**

- Add your implementation code for the CounterTop class overloaded assignment operator
- Copy and paste the code for Milestone 4, Test Suite 1 into your function main(), run the test, if there are issues you may need to work some more on the overloaded stream insertion operator of the CounterTop class and/or the overloaded assignment operator of the CounterTop class
- Add the modified prototype for the uploadFile function
- Copy and paste your implementation code from Project #4 for the uploadFile function
- Modify the uploadFile function (this requires some serious thought and planning, **do not begin coding until you have a good plan of what to change**, below are a few general changes that are required)
  - change the heading to match the parameter list of the prototype
  - change the function implementation code to use a linked list instead of a vector
  - change the function implementation code to dynamically allocate memory for each CounterTop object that is added to the linked list
- For the next test, you may want to use the small test file, also turn off any output in the destructor
- Copy and paste the code for Milestone 4, Test Suite 2 into your function main(), run the test

**Milestone 5 – Days 9-10**

- Add the modified prototype for the summaryByStone function and the summaryByRegion function
- Copy and paste your implementation code from Project #4 for these functions
- Modify the functions as necessary to use the linked list instead of a vector
- Copy and paste the code for Milestone 5, Test Suite 1 into your function main(), run the test  
(Note: If this test successfully runs, then you have completed the project. You may submit your program with the test code as if it was your own without attribution.)

## Class CounterTop declaration

Below is a portion of the code from the class declaration and comments about some of the key class members. You should edit your class declaration to reflect the changes / additions shown.

```
class CounterTop
{
    //overloaded stream insertion operator
    friend ostream& operator<<( ostream &os, const 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;

    CounterTop *next; //new for P5

public:
    //all member functions for this class are public
    //only new or modified member functions are shown below
    //see provided code for additional details

    //modified for P5 (may not be in-line)
    CounterTop(int oYYYY, int oMM, int oDD, int dYYYY, int dMM, int dDD,
        char sCode, double len, double dep, double hei, int lenF, int depF,
        string oNum, string reg, string stCode, string custNameAdd,
        CounterTop *ctPtr = NULL);

    //new for P5 (may not be in-line)
    CounterTop operator=(const CounterTop &rhsObj);

    .....

    CounterTop *getNext(); //new for P5 (may be in-line)
    void setNext( CounterTop *cTopPtr ); //new for P5 (may be in-line)

}; //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 of 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.

Most member functions are unchanged from Project #4. Key concepts to understand regarding the new or modified member functions are discussed below.

```
CounterTop(int oYYYY, int oMM, int oDD, int dYYYY, int dMM, int dDD,
    char sCode, double len, double dep, double hei, int lenF, int depF,
    string oNum, string reg, string stCode, string custNameAdd,
    CounterTop *ctPtr = NULL);
```

The constructor with parameters has one additional parameter inserted at the end of the existing parameter list. The argument passed to this parameter shall be used to initialize the next data member. Note that the default value is NULL.

```
CounterTop operator=( const CounterTop &rhsObj );
```

This function is the class' overloaded assignment operator. It is very similar to the copy constructor, but does have a few differences. First it is a value returning function. Additionally, its implementation code must test for and avoid self-assignment.

```
~CounterTop();
```

For this project, the CounterTop class has a destructor. The destructor for this class is not strictly necessary but it can be informative to add a destructor anyway with a few output statements. After observing the output generated by doing this the output may be commented out or deleted. The implementation code for the class destructor is provided for download from Blackboard.

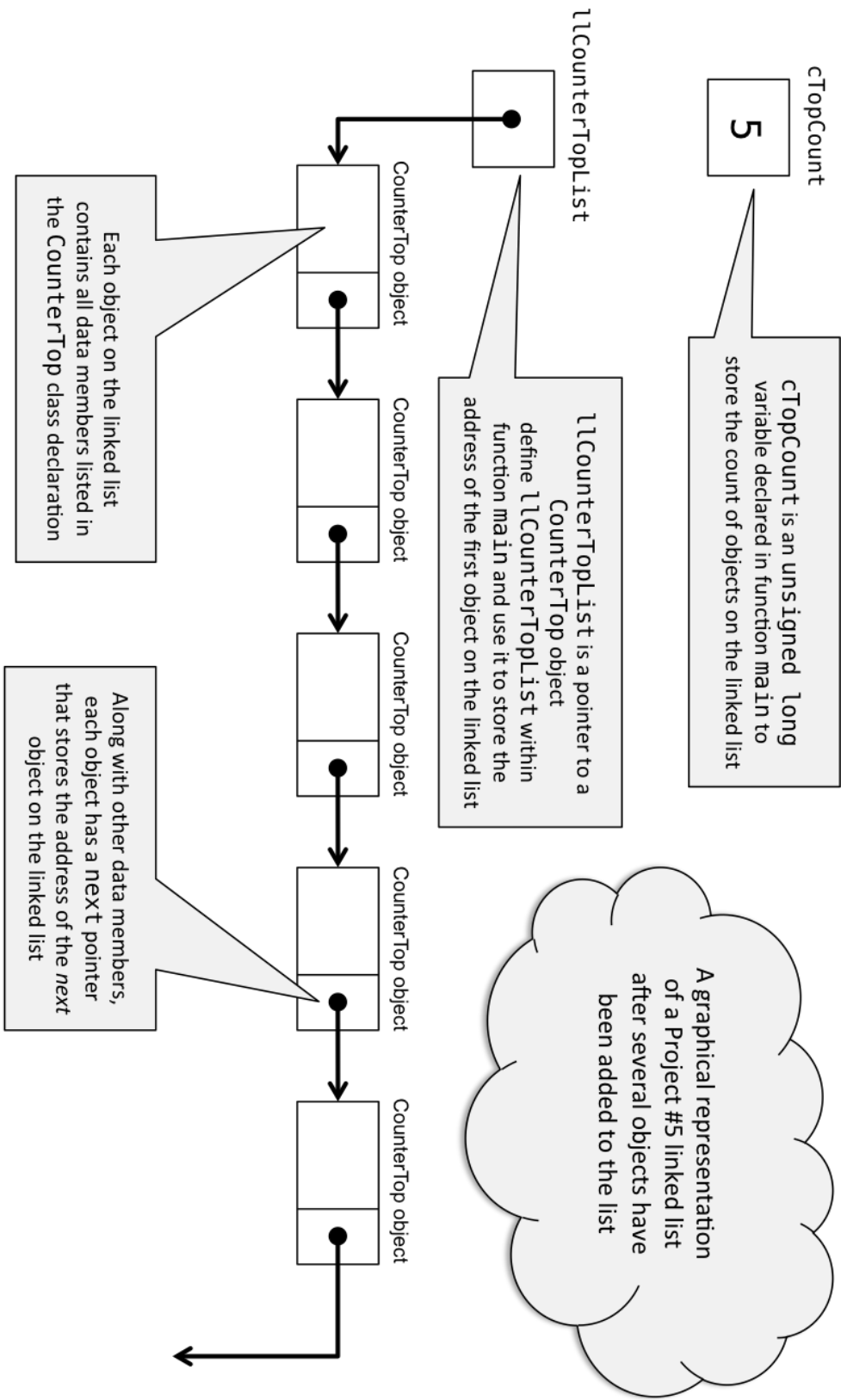
```
CounterTop *getNext();
```

This function is an accessor function of the class. It returns the value stored in the next data member.

```
void setNext( CounterTop *ctPtr );
```

This function is a mutator function of the class. It updates the next data member to the value passed as an argument to the ctPtr parameter.

Graphical Representation of a linked list of CounterTop objects





Grade Standards - Missing: 0%, Poor: up to 50%, Fair: up to 67%, Good: up to 82%, Excellent: up to 99%, Perfect: 100%		
	<b>Rubric</b>	<b>100.00 &lt;-- TOTAL</b>
<b>1</b>	<b>Code Quality and Formatting</b>	<b>20.00 &lt;--sub total</b>
	proper indentation, good use of vertical and horizontal white space, line length less than 100 characters	
	good variable and constant names	
	good use of constants / be comprehensive	
	good use of comments	
	menu is removed, instead the full path and name of the input data file shall be passed as a command line argument	
	functionality of the software shall be demonstrated by either running the milestone 5 test suite 1 or equivalent code in function main	
	Submission correctly, and in good faith, implements code to achieve the intent and requirements of the program as specified in the project description and clarified during in-class discussions and forum posts.	
<b>2</b>	<b>CounterTop class</b>	<b>15.00 &lt;--sub total</b>
	includes all members specified in the UML diagram	
	is correctly modified to be a self-referential class	
	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 and mutator functions for which implementation code has not been provided are correctly implemented, perform the required calculations, and return accurate results	
	All member functions listed in the class definition are included and fully implemented, function identifiers may be changed	
	Non-member functions (stream insertion operator) are correctly implemented	
	if any member functions are added to the class, they must enhance the class design and shall not replace or supersede any of the required functions	
<b>3</b>	<b>uploadFile function</b>	<b>25.00 &lt;--sub total</b>
	function name, return type, and parameter list match the provided prototype	
	the complete path and name of the input data file must be acquired from the first command line argument	
	input file is opened, and associated with a stream prior to being used	
	test is made to ensure file opened successfully	
	if file fails to open, the boolean parameter is set to false, the user is notified of a problem and the function ends (Optionally, you may have a loop that continues to prompt for a file name until the file successfully opens, so if that was the way your code worked before it does not need to be changed)	
	first line of input data is processed separately since it is not a row of data	
	program reads and processes the data in all lines of the input file(s)	
	program does not read past the last line of the input file	
	Input data are validated to ensure that the values are valid as shown in the project description	
	if any data fail validation, that row of data is not loaded, a brief error message is displayed on the terminal window describing the validation check that failed	
	if a row of from the file passes all validation checks, then those values are used to initialize/update the data members of a CounterTop object that object is then appended to the linked list of dynamically allocated CounterTop objects	
	will correctly process a file of any length (containing any number of data rows)	
	the input file is closed after all records have been read	
	records containing errors are not added to the linked list	
<b>4</b>	<b>clearLL function</b>	<b>20.00 &lt;--sub total</b>
	the clearLL function shall traverse the linked list and deallocate all memory that was dynamically allocated for the linked list	
	after the clearLL function executes the pointer to the front of the linked list shall be set to NULL, there shall be no memory leaks or dangling pointers	
	the count of loaded files and count of objects on the linked list shall be reset to zero	
<b>5</b>	<b>summary functions</b>	<b>20.00 &lt;--sub total</b>
	summary functions	
	an appropriate message is displayed and no data are presented if no files have been loaded (Note: if the user ever selects the "clear" option, all data should be removed from the vector, and the "state" of the system is then the same as if no files had been loaded)	
	any and all code in these functions that previously calculated area, material required, and cost values must be removed and replaced by calls to the corresponding CounterTop class member functions	

Note 1 Types of Errors. File-level error (file not found or file fails to open). Your program should output an error message, terminate the load function/process, and redisplay the menu.

Data-level error (invalid stone code, etc.). Your program should output an error message, not load that row of data into the vectors, continue processing the file and redisplay the menu when finished.

Data-type error (character value where number is expected, string value where char is expected, etc.) You are assured that this will not happen. There are no such errors in the test files provided. The TAs will not insert any such errors when grading.