Programming Project 3

EE312 Spring 2014

Due Feb 21st, 2014 before 5:00PM CST
FIVE POINTS
No late submissions accepted!

General: In this project you will write an inventory and customer relationship management (CRM) system for a hypothetical small company, *Ellie's Baby Emporium*. Your program will use the String abstract data type from class (you are given a fresh copy of this code for use in this project).

GRADING: you will be graded on producing the output precisely as documented. That includes all punctuation, capitalization, spacing, spelling errors, etc.

Input: Your system will read input from a file. The file contains a sequence of commands. Each command begins with a one of four keywords. The remainder of the command depends upon the keyword.

- **Inventory** <*type*> # – This command is used to record a new shipment from the factory into the store's inventory. To read this command you must read from the file the type of item received (Bottles, Diapers or Rattles) and the number of items in this shipment. You should then increase the store's inventory by that amount. For example: "Inventory Bottles 50" means that 50 bottles should be added to the store's inventory.
- **Purchase** <*name*> <*type*> # – This command is used to record a purchase by a customer. Each customer is known by their first name (it's a very friendly company), so <*name*> will be a one-word string for the customer's name. The <*type*> is once again Bottles, Diapers or Rattles and is used to indicate what type of item this customer purchased. Finally, the last part of the command is the number of items purchased. For example, "Purchase Craig Diapers 100" means that the customer "Craig" has purchased 100 Diapers.
- **Summarize** – This command requests that a summary be printed of the store's activity.
- **Quit** – This command terminates the input.

To get you started, *main.cpp* includes source code for reading the keyword at the start of each command and "understanding" the keyword. What *main.cpp* will do is invoke one of three functions depending upon what the keyword is.

- *processInventory()* – This function should read the item type and quantity from the input file and update the store's inventory of the indicated item type.
- *processPurchase()* – This function should read the customer's name, the item type and the quantity. The function should look up the customer in the customer database (creating a new customer record if this is a 1st-time customer) and increment the number of items purchased by this customer in their customer record. For example, if the customer record indicates that "Craig" had previously purchased 10 diapers and the current command indicates that "Craig" is purchasing 20 diapers, then the customer record should be set to indicate that 30

diapers have been purchased by Craig. Note that each customer should have their own customer record (so that Ellie can keep track of who her best customers are and offer incentives like coupons and things).

- *processSummarize()* – This command should print out a summary. The summary should first display the number of Bottles, Rattles and Diapers remaining in inventory at the time of the **Summarize** command. Next, the summary should display how many different customers have come to the store for purchases. Finally, the summary should report which customer purchased the most diapers (and how many diapers), who purchased the most bottles (and how many) and who purchased the most rattles (and how many). If a certain item has not been purchased by anybody, then the summary should indicate that.

You are provided with three input files. At the end of each file (after the **Quit** command) is a transcript of what the output should be from the **Summary** command. Please format your output exactly as shown in the file.

Error Handling: Note of course, that it is not possible to sell someone 50 bottles if there are only 20 in the inventory. If an error occurs when a purchase is attempted and the amount exceeds the amount currently in inventory, your *processPurchase* routine should print an error message, "Sorry, <name>, we only have %d <*type*>" (see the test files for the exact format the error message should take. In this case, you should not update either the inventory or the customer record (and you should not add any new customers to the database unless they actually buy something).

Your Mission: Edit the file Project3.cpp to complete this project. To read from the input file, you may use two functions *readNum* and *readString*. Both of these functions are inside *main.cpp* and you should not change these functions in any way. You should only call *readNum* when the next parcel of input in the input file is a number (you can always know what's coming because the commands are formatted in a specific order). When the next parcel in the input file is a string (like the customer's name, or the type of item purchased), use *readString*. Be sure to use *StringDestroy* appropriately.

The *Customer* struct is defined in *Invent.h*. You should use this struct to keep track of all the customers. For this project we'll assume that there will be a maximum of 1000 customers, so an array of 1000 *Customer* structs will be adequate, and most of the elements of the array will never actually be used. Each time you read a customer's name, you should search through this array to find a matching customer. Obviously, if there have only been three customer's so far, you should only search the first three entries in the array to find a match. Use the comparison function from the String abstract data type (StringIsEqualTo) to check to see if two Strings are the same. Note: you are required to use the Customer struct as defined in Invent.h AND you are required to use the String ADT example provided in String.h/String.cpp. PLEASE take a look at String.h! For this project you will need to call at least StringDestroy and StringIsEqualTo. The code in main.cpp illustrates how these functions can be used.

Your program must produce exactly the same output as specified in the test files, including the same punctuation, capitalization, spacing and typographical errors (if

any) present in those files. In addition, your program should produce the correct corresponding output for any other test file that we might create (with the equivalent formatting, punctuation, capitalization, typos, etc.)

Testing: We've supplied three test files. Please note that all three test files use "Summarize" as the last command in the file. Please do not assume that Summarize has to be the last command in a file, or that Summarize can appear only once. Any command can appear any number of times in any order. You might want to write your own test files (as always). You can edit the files using *WordPad* or any other text editor.