Programming Assignment #4

In this project you will write an inventory and customer relationship management (CRM) system for a hypothetical small company, *The Green Dragon Game House and Inn*. Your program will use the String abstract data type from class.

Setup

Begin by downloading the starter file provided on Canvas and creating a new project on CLion. You will be completing the functions described below in `Project4.cpp`.

Task Summary

When `main.cpp` is run, it reads input commands from a text file with a specified format, i.e., a series of one-word commands and arguments for each command. The required read function is provided to you. As each command is read from the input file, it is parsed (code provided), and then each command is executed (your code). The command processing functions (and helper functions) that you write must interact with a database (which is represented as an array of structs) that is provided to you, but must be modified and read by you in your code.

Input Breakdown

Input: Your system will read input from a file. The file contains a sequence of commands. Each command begins with 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 (Books, Dice, Figures, Towers) and the number of items in this shipment. You should then increase the store's inventory by that amount. For example: "Inventory Books 50" means that 50 books 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 Books, Dice, Figures, Towers 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 Frank Dice 100" means that the customer "Frank" has purchased 100 dice.
- **Summarize** – This command requests that a summary be printed of the store's activity.
- **Quit** – This command terminates the input.

Functions

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 "Frank" had previously purchased 10 dice and the current command indicates that "Frank" is purchasing 20 dice, then the customer record should be set to indicate that 30 dice have been purchased by Frank. Note that each customer should have their own customer record (so that the innkeeper can keep track of who their 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 Books, Dice, Figures, Towers 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 dice (and how many dice), who purchased the most books (and how many), who purchased the most figures (and how many), and who purchased the most towers (and how many towers). 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 books 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 `Project4.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 to avoid memory leaks.

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 customers 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.

Format

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.).

Submission

You should submit `Project4.cpp` to the corresponding Gradescope assignment. Do **NOT** change the name of `Project4.cpp`.

FAQ

Q: Where do I store the data for the inventory?

A: The simplest way is to create a global `Customer` to keep the numbers in. You should reinitialize the values to 0 between calls and store data in those values as necessary.

Q: Do we have to optimize the runtime for this lab?

A: As long as it runs in a reasonable amount of time (less than 5 seconds or so), it is not necessary to do any more optimization.

Q: How do `readString` and readNum work?

A: They read through the input file similar to how you would read through a `char` array.
`readString` returns a `char*` that points to a character at the beginning of a word and ended with a null (`\0`). It saves the current location in the string so that every call will move to the next word. `readNum` does something similar, but it does the extra step of converting to a number for you. To use them, allocate space for a string or an int on the stack, like this:

```
String in;
int in2;
```

and then pass pointers to those variables to the `readString` or `readNum` functions.

```
readString(&in);
readNum(&in2);
```

Just allocating space for a pointer (with some thing like `String* in`) doesn't make space for the actual String anywhere, just a pointer. Therefore we need to make space on the stack and then pass a pointer to that space into the functions, as shown above.

Q: If two customers are tied for max purchases of an item, which one do we return?

A: Return the first customer added to the list of customers that purchased that number of the item.
Example:

```
Inventory Books 4
Purchase Sue Books 1
Purchase Dan Books 2
Summarize
Purchase Sue Books 1
Summarize
Quit
```

This would return **Sue** for max purchaser of books because she was encountered first.

Q: Do we include customers who bought 0 or negative items? What if they can't complete their purchase?

A: Do not add customers who bought 0 items to the database (and also disregard customers who try to buy negative items), and don't include them towards max purchaser. Do nots print/output anything in these cases.

Q: Are customers with different capitalization considered different customers?

A: Customers with different capitalization should be considered different customers.
Example:

```
Purchase Dan Books 2
Purchase dan Books 2
```

Would put two books under **Dan** and put two under **dan**.

Q: Printing using `printf` is causing weird errors.

A: The strings given by `readString` are not null terminated. Using the string print function in `String.h` will print them correctly. Remember that the `len` field is used to print the string properly, so if you modify the string inside, make sure to update `len`. If you don't, the string will not print as expected.

Q: Will there be multiple calls to `readInput` followed by a single call to `reset`?

A: There will only be one call to `readInput` for each call to `reset`.

Q: When I run my program in the linux server, i am getting the error: `malloc` was not declared in the scope. How can I fix this?

A: Add `#include <cstdlib>`. The equivalent in C is stdlib.h.

Q: How do I detect memory leaks?

A: First, create your own `testx.txt` that has only one or two commands, that shows a leak. `Run valgrind ./proj4 -leak-check=full`. Compare the numbers of allocs and frees. Also, modify `StringCreate` and `StringDestroy` to print out the string that is being created or destroyed, and step through the debugger to isolate which created `String` is not being destroyed. This will find some common sources of leaks.

Note that `reset()` will be called right before `main` returns. This might not matter for your implementation, but if you allocate some of your data on the heap, it means that you can free it before returning.

Assume properly formatted input.

CHECKLIST – Did you remember to:

- ☐ Re-read the requirements after you finished your program to ensure that you meet all of them?
- ☐ Make sure that your program passes all our testcases?
- ☐ Check for memory leaks?
- ☐ Make up your own testcases?
- ☐ Upload your solution to Canvas?

..