# CSCI-1200 Data Structures — Spring 2021
# Homework 7 — Halloween Costumes

In this assignment you will design a data structure to maintain the inventory and customer data for a costume shop. The shop carries a variety of costumes and carries multiple copies of some of the more popular costumes. Customers come into the shop and rent the costumes; however, the customers are fickle and sometimes change their mind about what to be for Halloween. Each customer is only allowed to rent one costume at a time. Your task is to streamline the costume rental process and handle queries about the availability of costumes and "who's wearing what" to the next party. By using STL's associative container (`map`) we can make this system quite efficient and elegant. *Please carefully read the entire assignment before beginning your implementation.*

## Input/Output & Basic Functionality

Your program will read from `std::cin` and write to `std::cout`, but we expect you will *redirect* the input (& output) to trick your program into reading from & writing to files – see the course webpage: "Misc. C++ Programming Information". Each line begins with a character signaling which of four operations should be performed:

`a ghost 2` This operation will *add* a specified number of number of copies of a particular costume type into the costume shop inventory. The name of the costume is a single string with no spaces (underscores will be used if the costume name is multiple words, e.g., `Elvis_Presley`). If the costume is already in the shop, the number of copies is incremented. A useful message is printed to the screen following successful execution of this file. See the sample output.

`r Sally Smith princess` In this operation, a customer is attempting to *rent* a particular costume from the shop. The customer name is specified by two strings: their first and last names, in that order, and then the costume is specified by a single string, as above. If the shop does not carry that type of costume, if no copies are currently available, or if the person is currently renting a copy of this costume, then the request is declined with an appropriate message (see sample output). Each person is only allowed to rent one costume at a time; thus, if the person is currently renting a different costume, then that costume must first be returned, which will be done automatically by your code. Messages indicating the returned costume (if any) and rented costume are output.

`l gorilla` This operation *looks up* the specified costume and outputs the number of copies available for checkout and the names of the customers who currently have this type of costume rented. The customers are printed in chronological order, with the oldest rental first. If the shop does not carry this type of costume then an appropriate message is printed.

`p` This operation *prints* all customers who have ever attempted to rent from the shop (in alphabetical order by last name then first name) and the costume (if any) that they are currently renting.

Examples of the messages your program must output are available on the course website. To receive full credit on the assignment, please follow these examples exactly. To see if your program is performing perfectly, you may use the Unix library program, `diff` which takes two files as arguments and outputs the differences between them. `diff` is included with WSL. WinDiff is another option for Windows users. Please see a TA or the instructor in office hours if you have a question about these programs.

## Performance & Order Notation

You must carefully consider the performance of each of the costume shop operations and choose data structures to achieve efficient performance. Let $n$ be the number of different costumes in the shop, $m$ be the maximum number of copies of a given costume, and $c$ be the number of customers who visit the shop.

All of the operations should have sub-linear expected running time with respect to $n$. Furthermore, the `'a'`, `'r'`, and `'l'` commands should have sub-linear expected running time with respect to $c$. *Hint: That means you should use maps. In fact, you'll need at least two of them!* In your `README.txt` file include the order notation for each operation in terms of $n$, $m$, and $c$.

You are not explicitly required to create any new classes when completing this assignment, but please do so if it will improve your program design. We expect you to use `const` and pass by reference/alias as appropriate throughout your assignment. We have provided a partial implementation of the main program to get you started. You may use none, a little, or all of this, as you choose, but we strongly urge you to examine it carefully.

## Data Structure Diagram

As part of your electronic submission, you must include a neat diagram the overall data structures for this assignment. Follow the conventions from lecture for diagramming maps, lists, and vectors. Use this diagram to communicate design choices you made and how this data structure works. You can illustrate how a specific operation (just choose one) results in the insertion, modification, or removal of data in the structure. Consider diagramming the `small_input.txt` example, but you may change some of the details.

The accepted file formats for this diagram are .pdf, .png, or .jpg. Name your file `diagram.pdf` or `diagram.png` or `diagram.jpg` (depending on the filetype). You may draw this diagram with pen/pencil & paper and scan or take a photo with a camera or you may use electronic drawing/diagram software.

You will be graded on neatness, clarity, and amount of informative detail included in the picture. This diagram will be worth approximately 7 points of the total HW grade.

## Extra Credit

For extra credit, re-implement the functionality of your program without maps (use vectors and/or lists instead). Have your program take in an *optional* command line argument to specify the non-map version. How does the order notation for the expected performance of the program change? Test the two versions of the program with larger datasets to confirm your predictions of the performance differences. You can use the UNIX `time` command to measure the performance. Also, make up new larger test cases as necessary. Write up your analysis in your README.txt and submit both versions of the code – the map version should run by default (with no command line arguments).

## Submission

Use good coding style when you design and implement your program. Be sure to write your own new test cases and don't forget to comment your code! Use the provided template `README.txt` file for notes you want the grader to read. You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.