Project Overview

Our project took a slightly different approach then using linked links. Instead we utilized the STL and used different types of containers to keep track of and store data. Our program consists of 6 containers. Two vectors, two maps, and two lists. Before going into detail on the containers, it’s important to understand how we kept track of ‘bookings’ or ‘reservations’.

Our program has a single class called Reservation. Reservation has 3 data members, *customer*, *holiday* and *magician*. Each reservation has these 3 components. When a reservation is made, the reservation is either ‘active’ or placed onto a waiting list because the holiday is already booked. Both of these lists are of the same type, Reservation, however, waiting list has the *magician* member set to nothing because there are no magicians available on that date.

The two vectors are containers each holding a ‘list’ of the magicians or holidays in the program.

The maps are used to search for availability. If Bob wants to schedule a reservation for Easter, we have to know if there are magicians available on Easter. We used a map so we can ‘lookup’ magicians available on a specific date, instead of searching through our lists and meeting a set of conditions to check for availability. When we initialize these maps, we use information from the vectors above.

The two list containers are the meat of our program. These are substitutes to linked lists. Why did we use a ‘list’ container instead of ‘vector’ or a different container? Lists have faster performance than vectors for inserting, extracting and moving elements. When reservations are canceled, that instance could lie anywhere within the list. Therefore, we have to remove instances in random positions and using a ‘list’ container gives us the ability to do this. Our actual lists are in ordered with the most recent reservations at the end of the list. The only exception is on the waiting list. If a magician quits, and customer loses their actual reservation, then that customer is moved to the front of the waiting list via push\_front.

Filters can be applied to our list and map containers to view specific holidays, magician schedules, etc. Instead of sorting our actual list container in alphabetical order, we instead chose to add items to the end of the list. Otherwise every a time reservation was created, we would have to iterate through the list and compare customer names to determine where to insert it. We save processing time by just adding items to the end of our list and only sorting it when requested. So when a user wants view reservations by customer name, they enter the holiday and a new list (2D array) is dynamically allocated in said order, this list is then outputted to the user. The user has the opportunity to save the filter or report before it’s deleted. Saving these reports as well as the other containers made testing much more manageable.

Test Plan

The program was constantly tested as it was built. Each processing related function (i.e. Schedule, Cancel, etc.) took several hours to write and an additional several hours to test. The functions were unit tested as they were built, as well as unit tested after any additional updates. Eric spent a lot of time testing and debugging and the program runs really smooth because of it. Several test drivers were written for file I/O and then copied directly into the program. Eric created miniature sample projects in visual studio solely to create a directly and save a file to a specific path. It tested saving to a variable path, and after understanding how directories needed to be created prior to saving data in them, we we’re finally able to incorporate the driver into our program. The createDirectory, load and save functions are all replicas of test drivers.

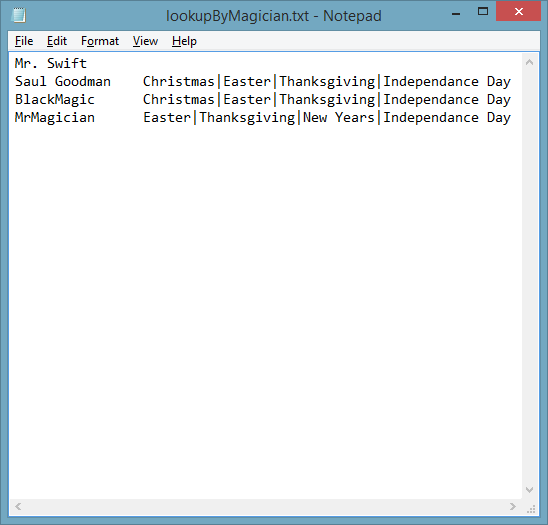
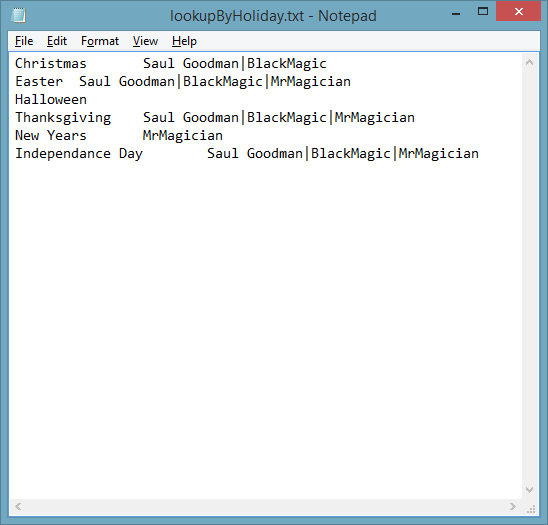
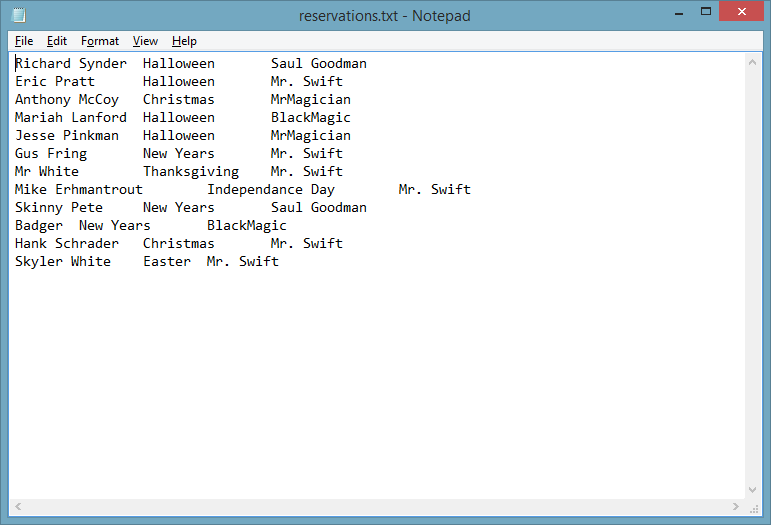
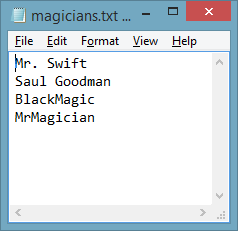
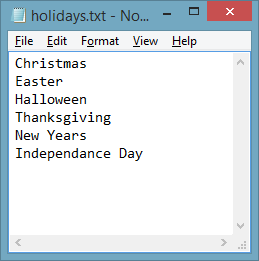
By outputting a variety of logs and being able to have them all correspond with one another, we we’re confident our project was running correctly. For example, if we saw magician Mr. Swift had 4 different reservations in reservations.txt, if we checked the lookupByMagician.txt file, we would see that Mr. Swift was missing 4 holidays from his availability. Furthermore, different statistics such as container.size() were included within functions so we could verify that sizes and magicians were moving as they we’re booked and removed from the program. We even took our testing a step further. After we saved our data to txt files, we wrote load functions to read the delimited files and add input the strings to their respective containers. We could load an existing project, and then verify our input was read correctly via Status(). The load functions were a great way to perform top-down testing. All the data was gathered at once so we had to work our way down to individual modules within our processing functions.

Extra Features

As someone who really enjoys learning programming, I (Eric) added some more features to make our project more realistic. Here’s some additional features that weren’t required but added anyway:

* Ability to create a new project; user enters their own holidays and magicians
* Ability to load existing projects
* Sample project embedded within the program
  + Having 15+ reservations already in the system saves a lot of typing
* Save temporary reports. When lists are filtered, the user can save that list to a txt file
* User friendly, catches bad input and allows user to try again or cancel and return home
* Saves to the actual C drive
  + Much easier to just run an exe than boot up an IDE and only save in local directory
* Reschedules people on waiting list when a new magician is added to the system
* Search for reservations by customer

Figure 1



In Figure 1, there are 6 holidays and 4 magicians. Since there are only 4 magicians, at most, that means there can be 4 reservations for a single holiday. In the lookupByHoliday.txt file, Halloween has no magicians available because all 4 magicians are booked on Halloween as shown in the reservations.txt file. Conversely, magician Mr. Swift is booked for all 6 holidays, therefore his availability will be nothing; he won’t be available any holidays. Elements are missing from the maps because they are scheduled, anything left on the map, means it hasn’t been booked yet.

Figure 2: Saved reports from sample data

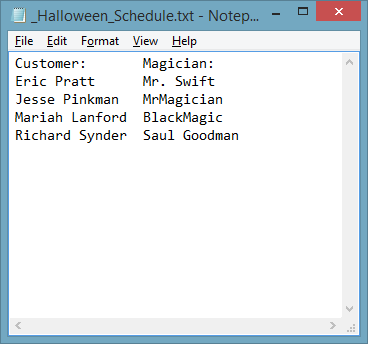
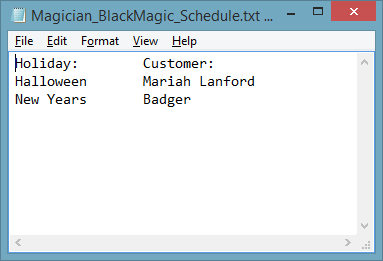


Figure 2 provides a sample report from a holiday and magician. These reports meet the ‘list’ requirements for the project. In other words, a list or report can be created for any holiday or magician and be sorted in alphabetical order by customer name or holiday.

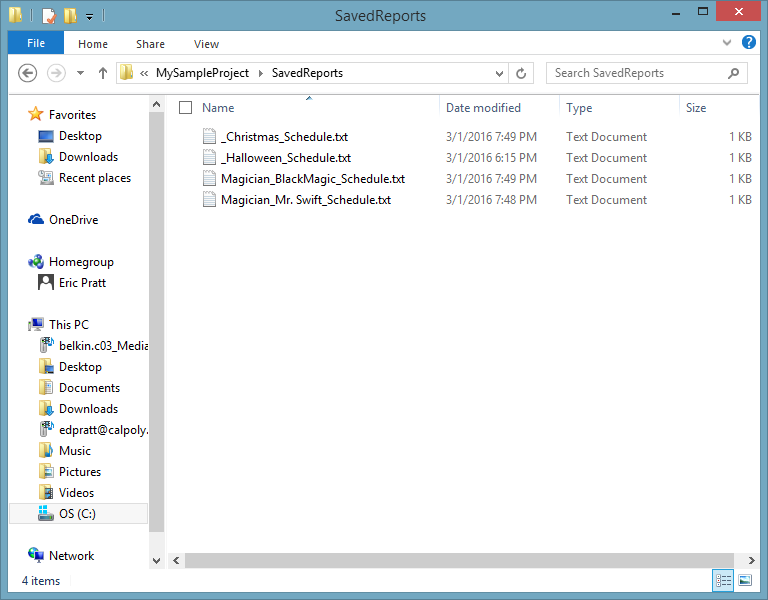
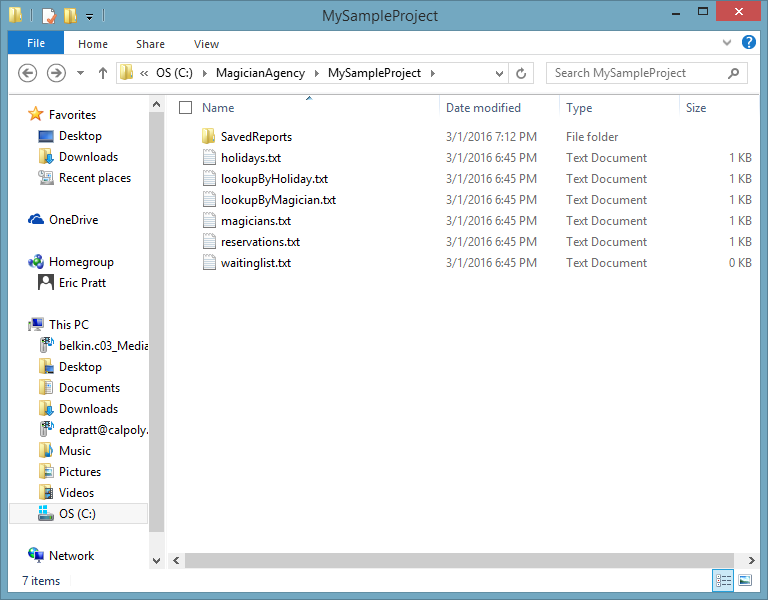
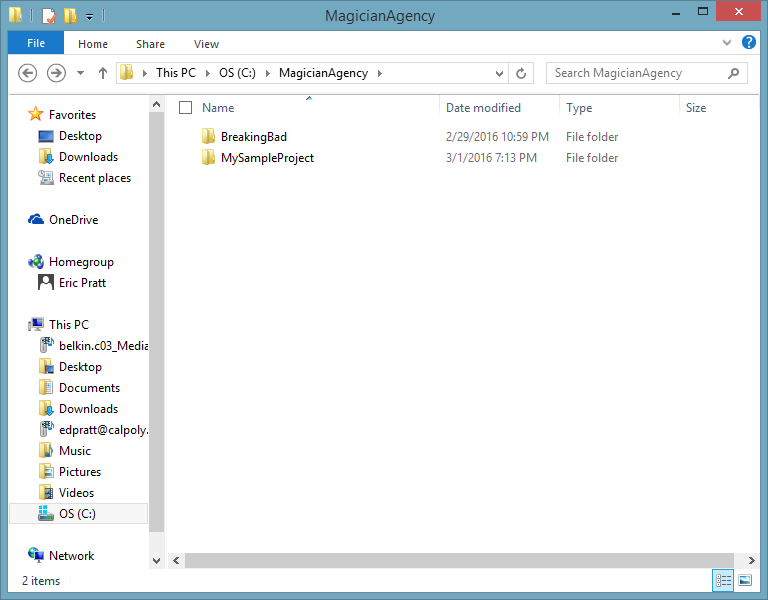


Figure 3 : Example folder hierarchy

Figure 3 shows folder hierarchy. The topmost folder contains the different projects. MySampleProject is the default project which is created during start up. However, users can create their own projects. In the top folder, a project, BreakingBad was created and saved. The middle folder shows each of the 6 containers outputted into their own txt file, with a folder for the saved reports the user creates within status of the current project. The last folder is a collection of all the saved reports.

|  |  |  |
| --- | --- | --- |
| Function | Expected Result | Actual Result |
| displayWelcomeScreen() | Display welcome screen to user | Works correctly |
| schedule() | Schedule a reservation for a magician | Prompts user to schedule a reservation for a magician |
| cancelReservation() | Cancel an existing reservation | Prompts user to enter information needed to cancel a reservation and deletes that reservation. |
| addMagician() | Add a magician to the list of magicians | Works correctly |
| removeMagician() | Remove a magician from the list of magicians | Works correctly |
| saveCurrentSettings() | Saved accurate directory | C:\MagicianAgency\MySampleProject\SavedReports |
| status() | Find out information for a specific magician or holiday | All functions corresponding with status work correctly (magician, holiday, waitlist, search) |
| addToReservations() | Add customer, holiday, and magician to the list | Works correctly |
| addToWaitingList() | Automatically shifts person on waiting list to corresponding holiday when available | Works correctly |
| viewReservationsOnWaitlist() | View Reservations that are on the waiting list | Works correctly |
| viewReservationsByMagician() | View reservations that are on the waiting list | Works correctly |
| viewReservationsByHoliday() | View reservations that are on the waiting list | Works correctly |
| Save functions | Upon selecting yes the project will save. Upon selecting no the project will not save | Works correctly |
| Load functions | Open an existing project | Works correctly |
| Create Functions | Create a new project | Works correctly |

Figure 4: Unit testing results