**Project 1B: Book Circulation**

Matt Miller & Laureen Nsoh-Awasom

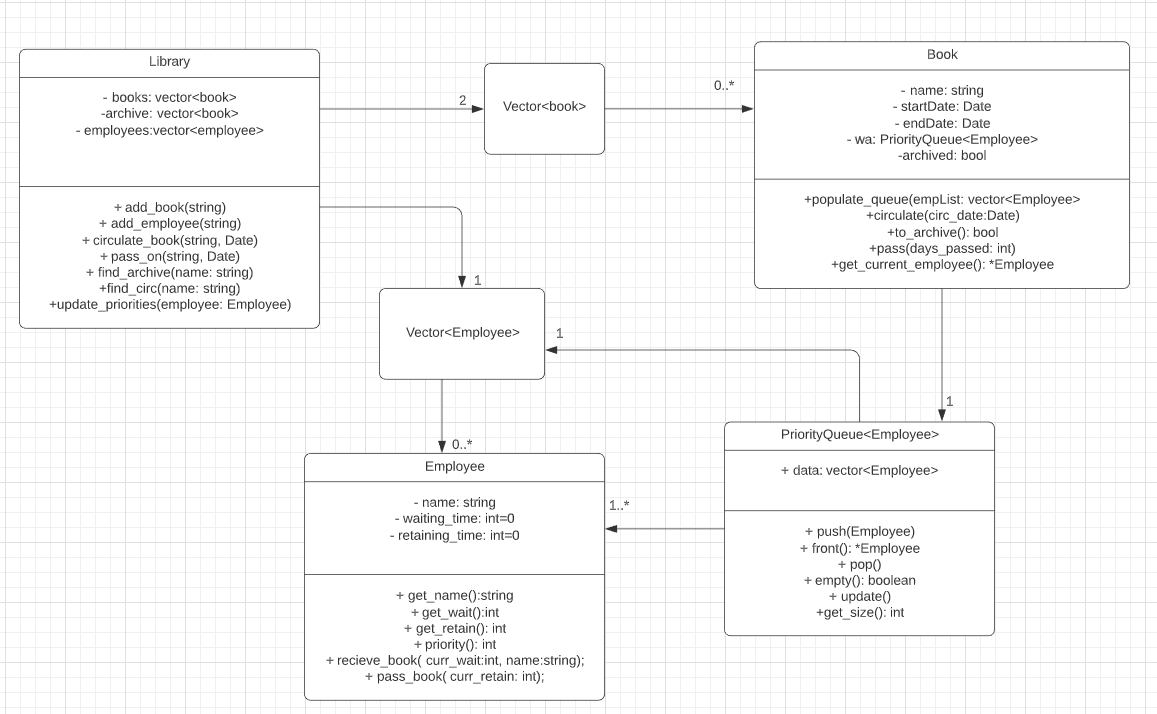
CS 303: Data Structures

Prof. Mayanka Chandra Shekar

July 7, 2020

GitHub: <https://github.com/MattMiller1989/Project1-b>

UML Diagram



When we were designing the project, the most important aspect was the priority of the employees which had to be constantly updated throughout the execution of the program. The project guidelines instructed us to use a priority queue without using the predifined priority queue class in the standard template library. One of the biggest design questions that we had was where to actually implement the priority queue. The first thought was to initialize and populate the queue within the library class. This would allow less complexity due to the fact that the priority of each of the employees would be updated more frequently. Our only concern with this first choice was mainly related to the legibility and sequencing factor. Because of these two concerns, we decided that it would be beneficial to have the priority queue initialized within the book class. The downside of this design choice is that a new priority queue needs to be initialized every time a book is put into circulation. The primary benefit of this design choice is legibility and simplicity. We were having trouble visualizing and planning the first choice so we ended up settling on the second one. A big part of the issues that we faced when trying to plan out the first design choice was trying to navigate the issue of circular dependencies. If I could go back and do the project over again, I would choose to do it in Java. This would allow me to implement the priority queue directly from the library class without having to worry about pointers and references.

Going into the project, we were first inspired by the phone directory. One of the aspects that really apealled to me about that project was the use of references and pointers when it came to the implementation of the methods which modified the data structure. Since C++ does not have automatic garbage collection, it is necessary to be deliberate when you are modifying the data. This was definitely the biggest challenge of the project. While this was helpful for some early inspiration, things become a little tricky when we realized that we had to make sure that the project would scale, since the book circulation project had many more moving parts. We did some googling at the start of the project, and we came across a repository for a priority queue on github: <https://github.com/yazanobeidi/priority-queue/blob/master/priority_queue.cpp> (Obeidi 2015). This program used binary heaps to organize the priority queues. At the time of discovery, we had not discussed heaps and were far more comfortable using vectors. This design choice to use vectors rather than heaps negatively impacted the complexity of our sorting functions with the tradeoff of it being more legible. Speaking of complexity, here are some of the relevant methods with their respective complexity.

Methods

Priorityqueue::update:

-This is the method that we use to sort the priority queue. We decided to use the selection sort primarily because of its simplicity and legibility. Time complexity of selection sort is O(n2) for best case and wort case (Singal 2019). Since we are dealing with rather small amounts of data, selection is an acceptable choice. If we were dealing with large sets of data, Quicksort or Heapsort would be a better choice. If we were to go with Heapsort, we would have to completely redesign the data structure and use heaps instead of vectors. This would give us a time complexity of O() (Morris 1998). If we still wanted to use vectors in the implementation of our priority queue, quicksort would a valid choice if we were dealing with large amounts of data.

Priorityqueue::find\_circ(string name):

-This is the method that we use to return the index of the book object from a vector of books. This vector represents all the books that are current in circulation. Once we have the index, we can use is to access the book object from the book circulation vector and pass it on to a different employee. We loop through the vector and return the index value of the object in the vector that has the specified name. We decided to do this because in the main function, the pass\_on method is passed the name of a book as an argument. The time complexity of this method is O(n).

Priorityqueue::find\_archive(string name):

-This is the method that we use to return the book object from a vector of books. This vector represents all the books that are currently in the archive. Once we have the index, we can use is to access the book object from the book archive vector and put it into circulation. We loop through the vector and return the index value of the object in the vector that has the specified name. We decided to do this because in the main function, the circulate method is passed the name of a book as an argument. The time complexity of this method is O(n).

Priorityqueue::update\_priorities(Employee& employee):

-This method is used to update the priority values of the employees. Within the priorityqueue, the priority of the employees changes every time the wait value or the retain value is updated. Since the employees in the priority queue are just copies of the employees that are added in the library class, we need to update the priority values of the original vector of the employees. This method loops through the original vector and compares the names of both employees. If the names match, the employee object in the original array is assigned to value of the employee object in the priority queue. This method was not added until the project had been finished, because we were having issues with making sure that the priority is properly updated after circulation begins for another book. The time complexity of the method is O(n).

Book::populate\_queue(vector<Employee>& empList):

-We use this method to populate the priority queue for the book object. The method receives, as argument, the vector of employees that work for the library. This method iterated through the vector of Employees and pushes each of them into the priority queue. Note that this method does not deal with the sorting of the priority queue. The time complexity of this method is O(n).

Works Cited

Morris, John. "Data Structures And Algorithms: Quick Vs Heap Sort". *Cs.Auckland.Ac.Nz*, 1998, https://www.cs.auckland.ac.nz/software/AlgAnim/qsort3.html. Accessed 8 July 2020.Obeidi,

Yazan. “Yazanobeidi/Priority-Queue.” *GitHub*, 1 July 2015, github.com/yazanobeidi/priority-queue/blob/master/priority\_queue.cpp.

Singhal, Akshay. “Akshay Singhal.” *Gate Vidyalay*, Publisher Name Gate VidyalayPublisher Logo, 21 Oct. 2019, www.gatevidyalay.com/selection-sort-selection-sort-algorithm/.