Kristian Snyder  
Data Structures CS2021 Section 001  
Prof. Charles Zimmer, Prof. Karthik Vadambacheri  
Lab 08

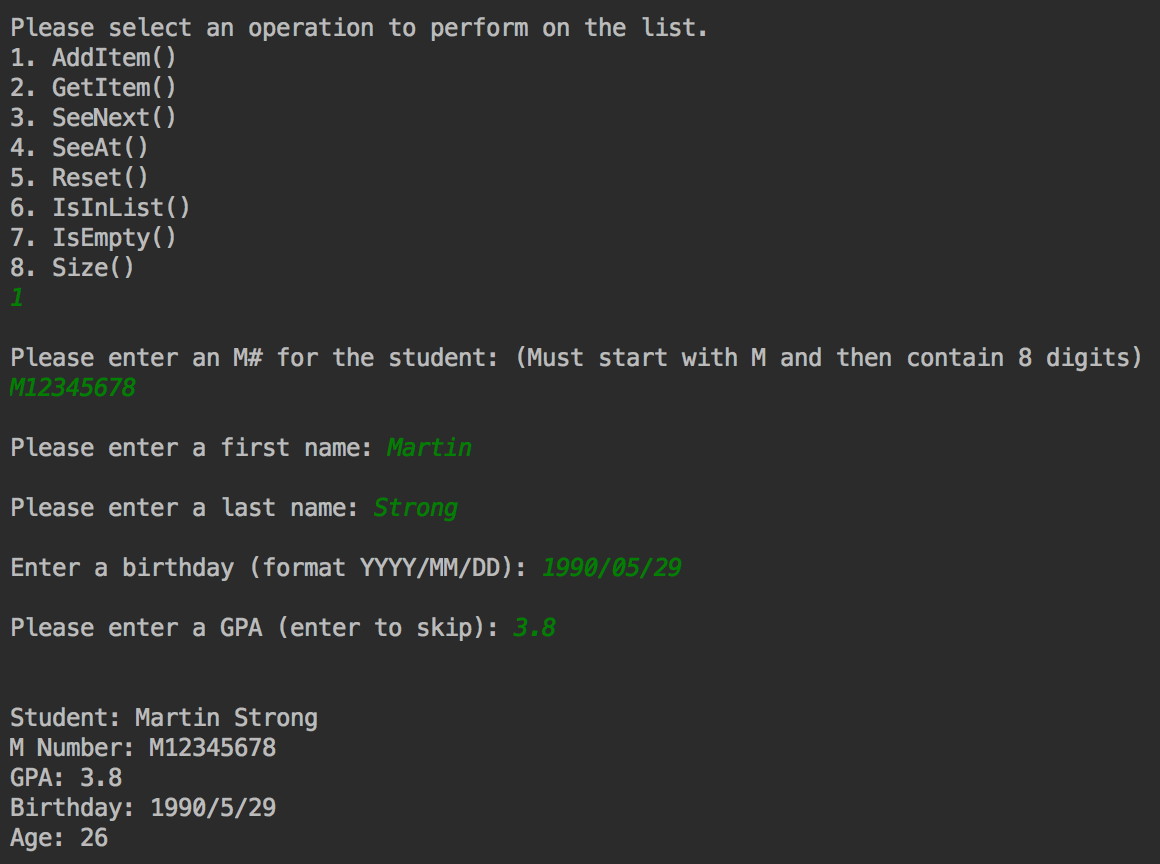
This lab focused on implementing a linked list primarily, but also handled design and structuring of classes resembling real-world ideas and how to handle data types with custom constraints. Consideration of memory management, efficient insertion and deletion for the list and how to compare objects for removal were important portions of creating the sorted list. For Students, ensuring that data would be in the correct format and how to handle creation of instances were the most applicable challenges to future projects and work.

While actually having to implement a linked list is a rare task in almost all modern programming languages, understanding their structure and, therefore, how and when to use them is crucial to creating efficient code. A singly linked sorted list using pointers has many advantages and disadvantages compared to other data structures and even other ways to implement the same concept. Having experience implementing many data structures provides improved instincts about which is best in a specific situation, and can be very helpful when dealing with complicated problems where the most efficient solution isn't necessarily easy to determine or look up.

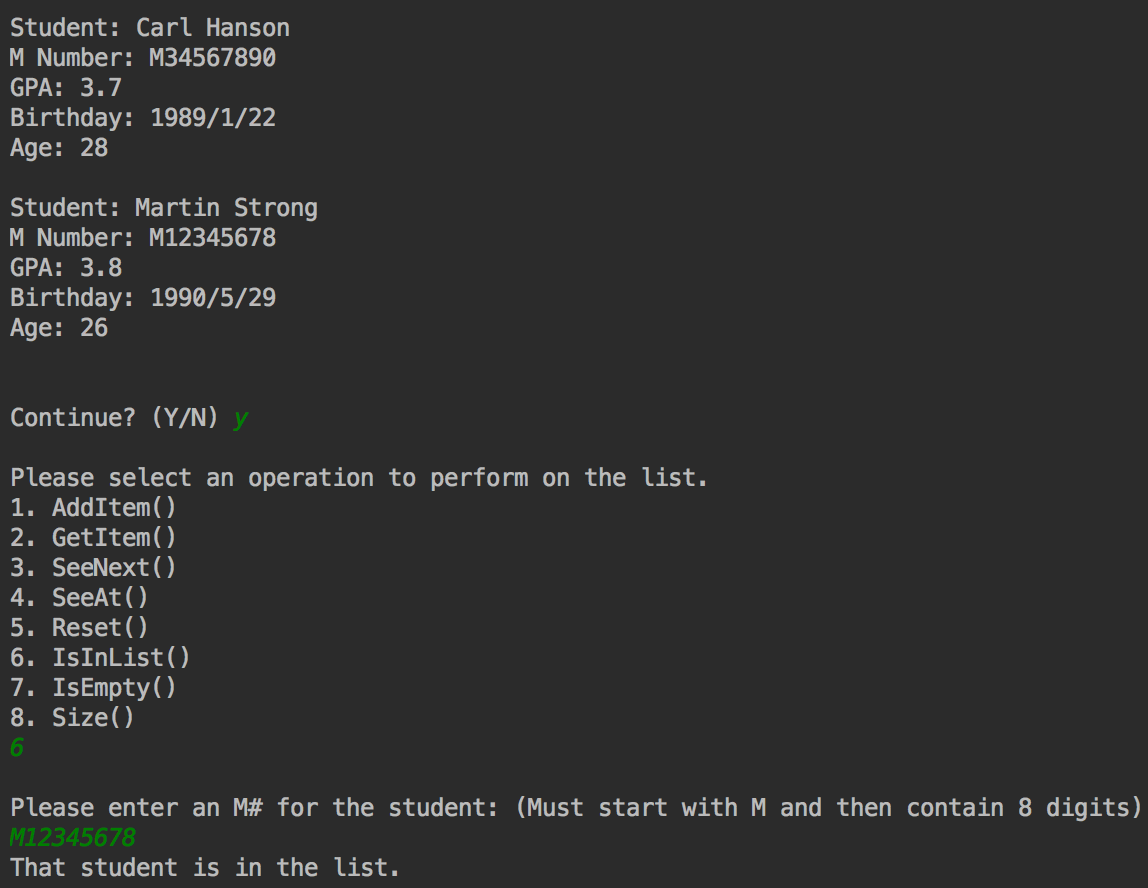
Additionally, having to tackle implementation of a linked list in C++ forces careful consideration of how to manage memory, as typically insertions and deletions from a linked list are so common as to result in large memory leaks without correct implementation. It can be very easy to forget to delete all data and clean up correctly, so doing this correctly in a linked list is a good primer for managing memory in a larger application in the future.

Creating the Student class was a useful exercise in creating objects that have constraints superseding basic types or structs in C++, such as creating an M# that has to begin with M and contain 8 digits exactly. This is where experience in application development is extremely useful in any project with business rules; making decisions on how to create and maintain the underlying datatype commonly has murky benefits at best for various options. Creating an M# can be implemented using an integer with a defined maximum and simply printing the integer with a prepended M and leading zeroes when requested, but it could also be performed with a string that has the entire format enforced on insertion. Decisions like this are some of the most common that a developer has to make in any field, so more experience in them is always welcome.

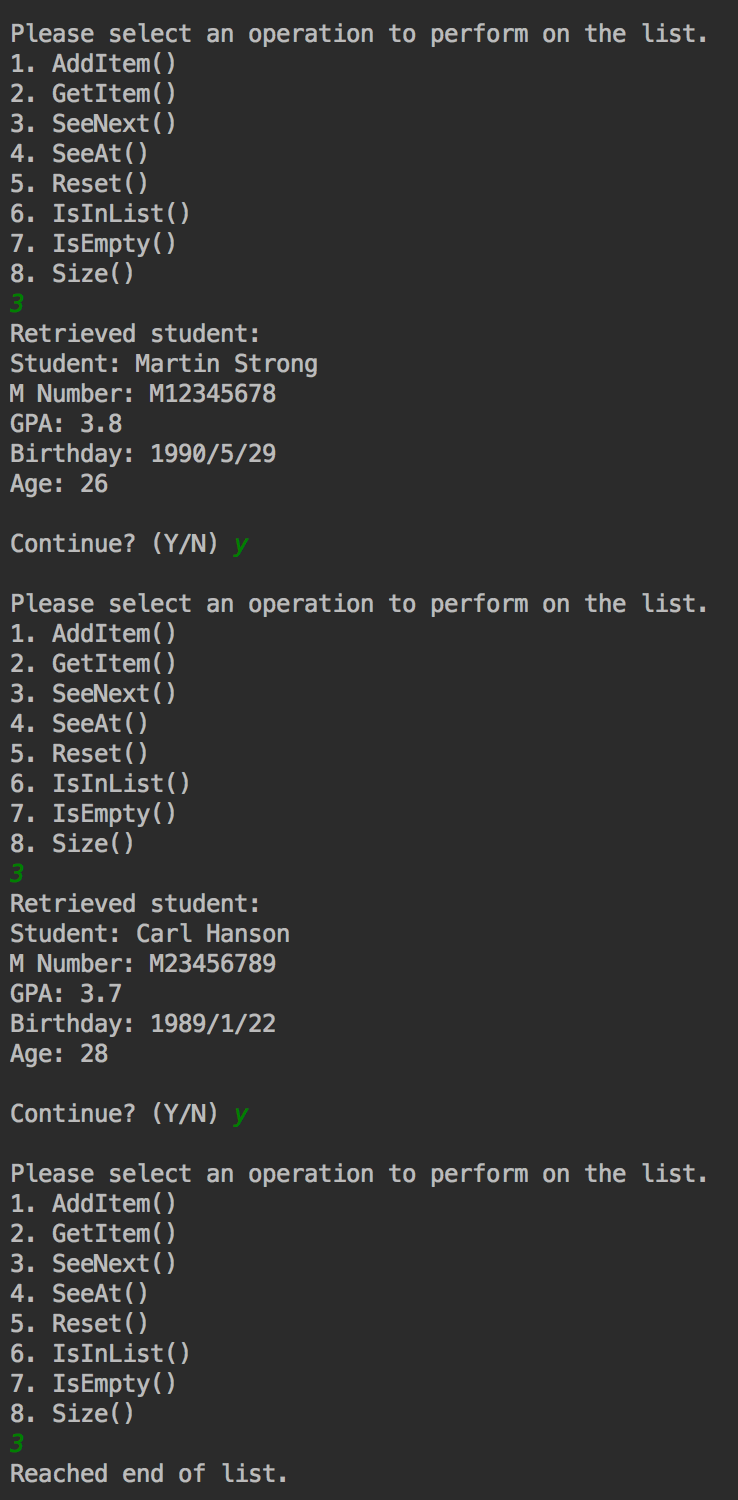
## Task 3



Adding a student

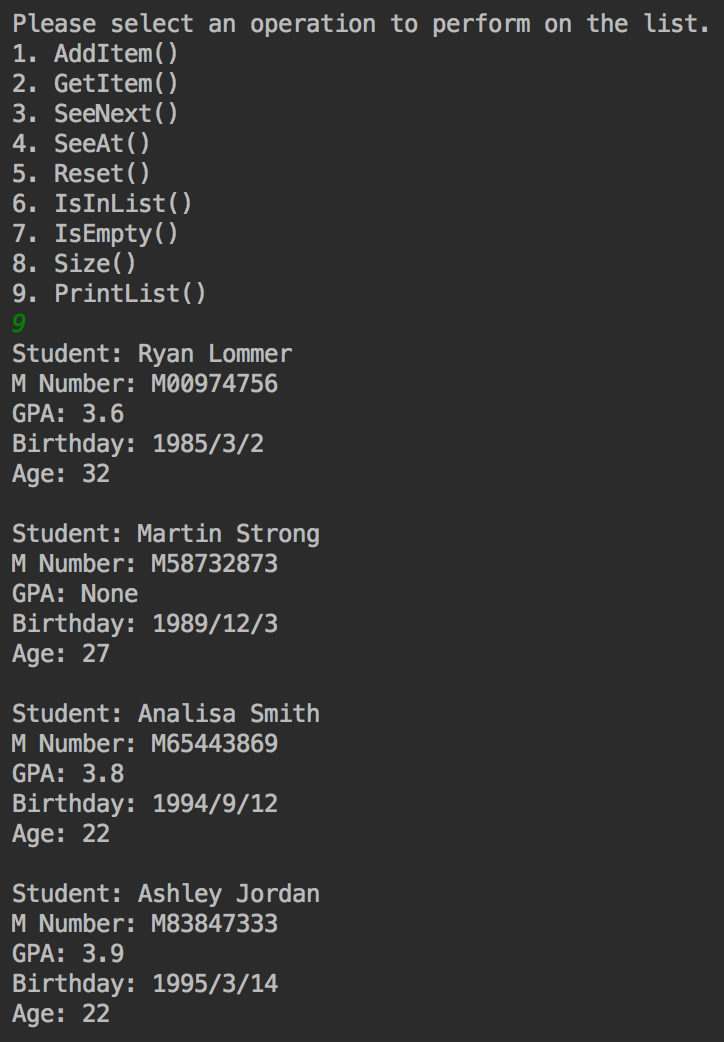


Checking if a student is in the list



Doing SeeNext() until the end of the list

## Task 4



Printing the contents of the list

All getters that were created in previous tasks were reused in order to display the contents of the list; no actual rewriting was necessary. However, in order to access the birthday and GPA, I implemented the display of a Student as an override of the << operator and a friend of the class, so I could access the underlying data. For safety toward the instance, the object is passed const so that no data can be overwritten. I leaned toward displaying a complete set of data for each Student when possible, so all information available for each instance is displayed.