- In order to receive full credit, your program must be correctly running, implements all required functions/data and organized.
- Maximum possible points 100

Assignment learning objectives: You are supposed to correctly define and write neat code that uses classes, functions, *dynamic arrays*, constructors, *destructors* and *overloaded operators*.

The assignment problem:

You have just started your new on-campus job at the front desk of the Swinney recreation center. Your job is to help students check-out and check-in gym items (towels, basket balls, rackets, bikes, etc). Traditionally, students who work this job record all items that students check out and in *manually* in two text files (checked out and checked in). At the end of the day, they update a third file (students.txt) that includes information about students and what items are currently checked out (if any). However, you believe that this process is tedious, time consuming and error prone. Therefore, you are thinking to *automate* this process by writing a C++ program that does all the work for you; because you are a great programmer who took 201L, and you can do it! See the following figure to better understand your program input and output.

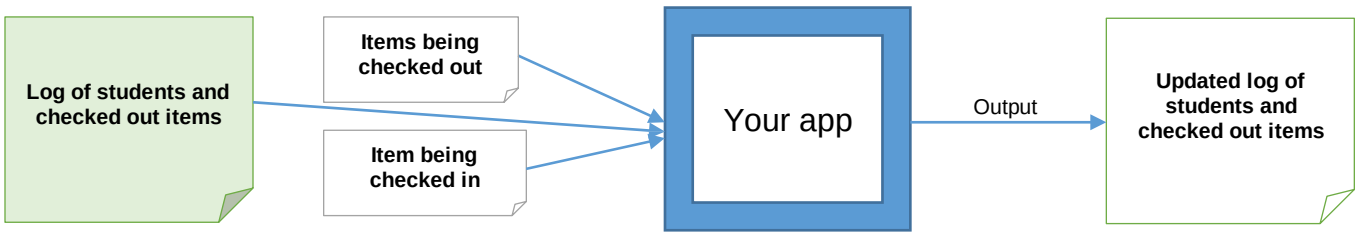

Figure1. Overview of the assignment's input and output data files

Hint: you know that not all students hit the gym frequently and, therefore, you realize that not all students check gym items all the time. At the same time, some students check several items simultaneously (e.g. towel, locker and a basketball). This means that you are not sure on how much memory to allocate for this data. Thus, you will write a *Student* class that has a dynamic array to hold the items that each Student has checked-out.

UMKC School of Computing and Engineering

Assignment 4 Due: October 23th

Your application will have a *Student* class with the following private data members:

- *string* for first and last name
- unsigned *int* for the ID number
- unsigned *int* for the number of items checked out and the array size
- string pointer (*string**) to manage the dynamic array

The Student class will have the following public methods:

Functions	Return type	Definition
Constructors	NA	Default constructor + constructor to allow default values of ID, first name and last name. The ID number defaults to 0 and the first and last names to empty strings
Getters	int, string	Getters to return ID, first name and last name. Getters should not change the object's data.
Setters	void	Setters for ID, first and last names (if required). The *setID* setter should check that the ID is between 1000 and 10,000. Both ends included.
CheckoutCount()	int	Returns the number of items a student has currently checked out. It should not change the object's data.
CheckOut(const string& item)	bool	Verifies that the item is not already checked out, and adds it to the dynamic array of checked-out items. If the item is already on the checked-out list, or for some reason the item cannot be added, it returns false; if adding the item was successful, this function returns true. If there is no dynamic array and an item is checked out, make the dynamic array of size 5.
CheckIn(const string& item)	bool	Verifies that the item is already on the checked-out list, and removes it. If this was the last item the student had checked out, the dynamic array should be deleted. If the check-in was successful, this function returns true, otherwise it returns false
HasCheckedOut(const string& item)	bool	Returns true if the item is on the Student's checked-out list, otherwise returns false. Should not change the object's data
Clear()	void	This method removes all of the object's data; the ID is set to 0, first and last names to empty strings and all checked-out data deleted.
Destructor	NA	the Student class needs a destructor to delete the dynamic memory

You application will also overload operators to be used on your defined type (Student class objects), including the following:

Operator	Definition
Stream extraction >>	`istream& operator>>(istream& in, Student& item)` Used to read the object's data from an *istream* (read data from the text files). It is also a friend function and takes a reference to an *istream* and a reference to a *Student* as its parameters. It begins by calling *Clear()* on the object, then by reading in the data in the same format as the insertion operator >> writes it to a stream. Whether this operator makes an array just large enough for the checked-out items, or calls CheckOut() repeatedly, is an implementation detail; handle it however you see fit.
Stream insertion <<	`ostream& operator<<(ostream& out, const Student& item)` Used to send the object's data out to an *ostream* (write data in text files). It is a friend function that takes a reference to an *ostream* and a const reference to a *Student* as its parameters. It writes the data to the stream in the following format: • The ID number, space, first name, space, last name, newline. • An integer with the number of items currently checked out, newline. • First item, space, second item, space, third item, space, etc… If the number of items checked out was 0, there is no third line of output.

Addition +	This is used as an alias for CheckOut. That is,
	Student Phelps;
	const int& item;
	Phelps = Phelps + item; // Phelps checks out an item ID.
	This function takes a Student on the left, const int& on the right, and returns a const Student. It will be slightly easier to implement as a member function.
	
	Note that this should return a new User;
	User2 = Phelps + item; // Phelps should not be modified by this.

Addition in place +=	Addition. Add in place: Another alias for CheckOut. Phelps += item; // another way to check something out This does not create a new object, it just checks it out for Phelps. This method does not return a value.
Comparison for equality == **and inequality !=**	Two objects are considered the same if they have the same ID number, otherwise they are unequal.

The *main()* function:

You have three **input** files:

- The first file, *students.txt*, is the roster of all students who come to the gym, including what items they have checked out (if any). Your program should use a dynamic array of Students class to store this data.
- The second file, Items*CheckedOUT.txt*, lists what several users have checked out. It is in the form of a series of (StudentID, ItemID) ordered pairs, with items separated by whitespace. Your program will process this file to check out the indicated items correspondingly.
- The third file, *ItemsCheckedIN.txt*, has several items that are being checked in. Search the data array until you find the student who has that item checked out first, and check it in using the class method.

Your application output:

Your application should produce a new text file, *Updated_Students.txt*, which is an updated file with all check-outs and check-ins processed, in the same format as the *students.txt* file.

Remember:

- Make sure you clearly understand the program and its requirements
- Always utilize the instructor's office hours
- Write a neat easy-to-follow code and organize it
- After you finish coding all the requirements of this program, think what can go wrong with the code. How can others mishandle it, and how should your program respond to bad input. Improve your code accordingly.
- **Zip your project and upload it to Canvas no later than midnight on October 23th.**

~BestOfLuck()