EC327 - Spring 2025 - Lab 4

Background

This lab assignment will provide hands-on experience with fundamental object-oriented concepts, focusing on inheritance and polymorphism.

Important

Please make sure your code compiles and runs as intended on the engineering grid. Code that does not compile will NOT be graded and will receive a 0.

Submission Instructions

You will submit this assignment as a single .zip file with the following name:

<first-name>_<last-name>-lab4.zip

So for example:

ed_solovey-lab4.zip.

Header Files

If you don't want to compyu the headerfiles from the pdf you can clone them from the repo found here.

If you wanted to clone this to your device you can do:

```
git clone https://github.com/esolovey-bu/EC327-Spring2025
```

Note on Collaboration

You are welcome to talk to your classmates about the assignment and discuss high level ideas. However, all code that you submit must be your own. We will run code similarity tools against your submissions and will reach out with questions if anything is flagged as suspicious.

The closed-book exams for this class will mostly cover material very similar to what you do on the homework assignments. The best way to prepare for the exams is to do the assignments on your own and internalize the learnings from that process. Cheating on the assignments will very likely result in you not doing well on the exams.

Helpful Information

In this lab, you will be using vectors, and since we have not covered them explicitly in class yet here is a quick highlight:

Vectors are similar to arrays, but you don't have to declare a size for them when you initialize. Rather they are adaptive, a great data type if you don't know how much data may be stored in a give array. Here is a little example of how vectors can be used.

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main() {

   vector<int> v = {1, 2, 3, 4, 5}; //Can initialize vector with values

   for (int i = 0; i < v.size(); i++) { //.size() gives the length of the vector
       cout << v[i] << " "; //Can index in using [] brackets
   }
   cout << endl;

   vector<string> v2; //Initialize a vector without values
   v2.push_back("Hello"); //.push_back add the values to the back of the list
   v2.push_back("World");
   v2.push_back("!");
   v2.push_back("Goodbye");

   //remove the last element
   v2.erase(v2.begin() + 3); //.erase removes a values from the list

   for (int i = 0; i < v2.size(); i++) {
       cout << v2[i] << " ";
   }
   cout << endl;
   return 0;
}
```

If you want to learn more about vectors please visit this link here or here.

Problem 1 - Alphabet Counter (20 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **lab4** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-lab4-1.cpp

So for example, my file would be:

ed_solovey-lab4-1.cpp

Actual Problem

Create a header file called **lab4_problem1.h** with the following contents:

```
#ifndef LAB4_PROBLEM1_H
#define LAB4_PROBLEM1_H

#include <string>
using namespace std;

/**
* Reads in the contents of a file, and return the (case-insensitive)
* English-letter frequencies within the file.
* @param filename The name of the file to read.
* @return A new array (allocated on the *heap*) whose i-th entry contains
*    the number of times the i-th letter of the English alphabet appears
*    within the file, either as a capital or lower-case letter.
*
* @example
* If the file "example.txt" contains the following text:
* The quick brown fox jumped over the lazy dog.
*
* Then alphabetCounter("example.txt") will return a 26-integer
* array with, among others, the following values:
* - Its 0th entry will be 1 - because the letter 'a' appears just once in the
file.
* - Its 3rd entry will be 2 - there are two 'd' 's in the file.
* - It's 14th entry will be 4 - there are four 'o' 's in the file
*/
int *alphabetCounter(string filename);

#endif //LAB4_PROBLEM1_H
```

Your **<first-name>_<last-name>-lab4-1.cpp** should implement the above **alphabetCounter** function.

Hint: To determine the correct position in your frequency array, subtract the ASCII value of 'a' from the lowercase character to get its zero-based index in the alphabet.

You can test your implementation from a **main** function or from tests you set up otherwise. Your submission will be tested against multiple inputs, and you should convince yourself that your solution works for the general case. This goes for all the problems.

Problem 2 - Fields Counter (20 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **lab4** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-lab4-2.cpp

So for example, my file would be:

ed_solovey-lab4-2.cpp

Actual Problem

Create a header file called **lab4_problem2.h** with the following contents:

```
#ifndef LAB4_PROBLEM2_H
```

```cpp
#define LAB4_PROBLEM2_H

#include <string>
using namespace std;
/**
* @param filenameCSV A file in CSV (comma-separated values) format,
*   in which each row is a line of fields, separated by commas,
*   following these rules:
*   - Fields may be enclosed in double quotes, or not.
*   - Fields *must* be enclosed in quotes if they contain a comma or
*     double-quote mark.
*   - Double-quotation marks inside quoted fields are doubled.
* @return The maximum number of fields in any line of the file.
*
* @example 1
* Suppose the file "example.csv" contains the following, single line:
* 1729, San Francisco, "Hello, World", "He asked: ""Quo vadis?"""
*
* Then fieldCounter("example.csv") should return 4, corresponding to the
* four fields:
* - 1729
* - San Francisco
* - Hello, World
* - He asked: ""Quo vadis?""
*
* @example 2
* Suppose the file "example2.csv" contains the following lines:
* a,b,c
* "a", "b", "c", "d"
* 1, 2
*
* Then fieldCounter(example2.csv") should return 4, corresponding to the
* line with the most fields ("a", "b", "c", "d").
*/
int fieldCounter(string filenameCSV);

#endif //LAB4_PROBLEM2_H
```

Your **<first-name>_<last-name>-lab4-2.cpp** should implement the above **fieldCounter** function.

Problem 3 - Student Class (30 points)

Submission Instructions

Your solution to this problem should contribute a **cpp** file and its matching **header** file to your overall **lab4** zip.

The **cpp** file should follow the following format:

<first-name>_<last-name>-lab4-3.cpp

So for example, my file would be:

Ed_solovey-lab4-3.cpp

The **header** file should follow the following format:

lab4_problem3.h

Actual Problem

Create a header file called **lab4_problem3.h** based on the following contents:

```
#ifndef LAB4_PROBLEM3_H
#define LAB4_PROBLEM3_H

#include <string>
using namespace std;

/**
* This class represents a student with the following properties:
* - It has a name (string) and quiz score(s) (int).
* - It can be constructed with a name, or with a name and a quiz score.
* - It has an appropriate destructor.
* - Quiz scores can be added, one at a time, to it with a method addQuizScore.
* - It has a method getAverageScore() that returns the average
*       of all scores entered for the student.
*
* @example
* labFour_p3 student("Jack Daniels");
* student.addQuizScore(100);
* student.addQuizScore(70);
* student.addQuizScore(70);
* Then student.getAverageScore() should return 80, the average of 100, 70, and
70.
*/
// please provide the implementation in your cpp file
class Student {

};

#endif // LAB4_PROBLEM3_MIDDLE_H
```

Your **<first-name>*<last-name>-lab4-3.cpp*** *should implement the* ***Student*** *class that you header file* ***lab4_problem3.h*** *describes.*

For problem 3 please submit both your **<first-name>*<last-name>-lab4-3.cpp*** *file and your lab4_problem3.h file to the autograder when submitting.*

Problem 4 - Person Class (30 points)

Submission Instructions

Your solution to this problem should contribute a single **cpp** file to your overall **lab4** zip. The **cpp** file should follow the following format:

<first-name>_<last-name>-lab4-4.cpp

So for example, my file would be:

ed_solovey-lab4-4.cpp

Actual Problem

Create a header file called **lab4_problem4.h** with the following contents:

```
#ifndef LAB4_PROBLEM4_H
#define LAB4_PROBLEM4_H

#include <string>
#include <vector>

using namespace std;

/**
* This class represents a person with the following properties:
* - It has a constructor that creates a person with a given name
*       (as a string) and no friends.
* - It has member functions matching the signatures below.
*
* @example
* Person person1 = Person("Jim");
* Person person2 = Person("Jane");
* Person person3 = Person("Bob");
*
* person1.befriend(person2);
* person1.befriend(person3);
* person2.befriend(person1);
*
* person1.getFriendCount() - returns 2 (friends)
* person2.getFriendCount() - returns 1 (friend)
* person3.getFriendCount() - returns 0
*
* person1.getFriends() - returns "Jane, Bob"
* person2.getFriends() - returns "Jim"
*
* person1.getFriendsOfFriends() - returns "Jim"
* person2.getFriendsOfFriends() - returns "Jane, Bob"
*
*/
class Person {
```

```
private:
     string name;
     vector<Person*> friends; //ordered by when added first is index 0

public:
    Person(string name);

    /**
     * Adds thePerson as a friend of this person.
     * @param thePerson person to add as a friend.
     */
    void befriend(Person& thePerson);

    /**
     * Removes thePerson as a friend of this person.  If the
     * person was not a friend, then this does nothing.
     * @param thePerson The person to remove.
     */
    void unfriend(Person& thePerson);

    /**
  * @return The number of friends this person has.
  */
    int getFriendCount();

    /**
     * @return A comma-separated list of the names of all friends
     *    of this person, in any order.
     */
    string getFriends();

    /**
     *
     * @return A comma separated list of the names of all friends
     *     of the friends of this person, in any order.
     */
    string getFriendsOfFriends();
```

```
   };

#endif //LAB4_PROBLEM4_H
```

Your **<first-name>_<last-name>-lab4-4.cpp** should implement the above Person class.