HW5
Out: October 18, Wednesday -- DUE: Oct 30, Monday, 12:00pm (noon)
EC327 Introduction to Software Engineering – Fall 2023

Total: 100 points

- *Comment your code and use a clean, easily understandable coding convention. A few coding style guides for your reference:*

http://geosoft.no/development/cppstyle.html
https://google.github.io/styleguide/cppguide.html

Submission:

1. Failure to follow submission guidelines (filenames, program I/O, and others) WILL result in a significant loss of points on the assignment as we use autograders.
2. Completed assignments **must** be submitted on GradeScope.
3. Submit the required files for each question:
   MyString.cpp MyString.h Q2.cpp Person.h Person.cpp Q3.cpp
4. 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.

Late Homework Submissions:
Recall the late submission policy and the fixed penalty of 20% for submitting up to two days after the deadline. Also recall that we only grade your latest submission (e.g., if you submit both on time and after the deadline, only your late submission is graded).

Please write C++ code for solving the following problems.

Q1. *Strings and Class Design.* **[40 points]**
The string class is provided in the C++ library. Provide your OWN implementation for the following functions in a new string class called MyString with additional features. You do not need to provide any other functions other than those listed below. The below prototypes should appear on your header file (**MyString.h**); and you should implement the functions in **MyString.cpp**.

```
MyString(char chars[], int size);           //constructor
MyString  append(int n, char ch) ;          //appends n copies of character ch to this string
MyString assign(MyString s, int n);         //assigns the first n number of characters in s into this string,
                                            starting from the first character in this string
int compare(MyString s) const;              //returns a value greater than 0, 0, or less than 0 if this string is
                                            greater than, equal to, or less than s, respectively
int compare(int index, int n, MyString s) const;  //compares this string with s(index,index+1,…,index+n-1)
char * data() const;                              //returns a pointer to a char array that contains the same
                                                  sequence of characters as in this string object
MyString commonSuffix(const MyString &s) const;       //returns the common suffix (*) of this string and
                                                  string s, if any.  Returns an empty string otherwise.
```

(*) Newton and Boston have the common suffix “ton”, ABC and abc do not have a common suffix (case sensitive), code and dude have the common suffix “de”.

Make sure to test your code, but you will be ONLY submitting your class declaration and implementation in the MyString.h and MyString.cpp files.

Q2. *Classes, pointers, dynamic memory allocation.* **[30 points]**
Create a class called Person. Use Person.h and Person.cpp files for declaring and implementing the class. Person class has the following private data fields: weight in pounds (double), height in inches (double), and name (C-string). Each data field should have the corresponding public *set* and *get* functions. Person class also has a public function, double getBMI(), that returns the Body Mass Index (i.e., weight (lbs) / height^2 (inches) x 703). Define a no-arg default constructor and another constructor that initializes an object by setting all the data fields to given input arguments. **You should use the following prototypes for your class:**

```
Person();
Person(double weight, double height, char name[]);
void setWeight(double weight);
double getWeight();
void setHeight(double height);
double getHeight();
void setName(char name[]);
char* getName();
double getBMI();
```

Write a program (in Q2.cpp, including the necessary .h files) that receives a number N from the user. The program should create N objects out of the Person class, and then have the user enter the necessary information to initialize all data fields in each object. Your program should then use the getBMI() function in the Person class to print the BMIs for all people on the screen in *increasing* order **using only two fractional digits**.

To ease testing your code, you should read input values from a file (instead of waiting for the user to enter the numbers one-by-one) using redirection:
./a.out < inputfilename

Put your class and function declarations in a file called *Person.h*, the implementation of the class in *Person.cpp*, and your main function, which reads the user input and prints the *getBMI()* output, in *Q2.cpp.*

Sample run:

Sample input file input.txt:
```
3
176 72.4 Jill
204 66.14 Elvis
165.6 77 Mike
```

Running your program with `./a.out < input.txt` should result with the following on the console:
```
Mike 19.64
Jill 23.60
Elvis 32.78
```

Important: Your code will go through an autograder so make sure to follow all given requirements above, e.g., your code should print only two fractional digits.

Q3. *Pointers and strings* **[30 points]**
Write a C++ function to read N words from a file called words.txt (which is in the same directory as your code). Then the function should find if a given expression exists in any of the words. The function writes the words that contain that expression in output.txt in case-insensitive alphabetical order. The function prototype is as follows:

```
void findExpression(int N, string & expr);
/* reads N words from words.txt, looks for the expression given in expr in each
word, and writes the words that include the expression in output.txt in case-
insensitive alphabetical order */
```

For example, assume words.txt contains the following words:

```
program
Boston
cheers
astonished
Stop
```

After function call:

```
string s1(“sto”);
findExpression(5, s1);
```

output.txt will include:

```
astonished
Boston
```

Make sure to test your function as needed, but you will not be submitting your test code. Please put ONLY your function implementation in Q3.cpp and include any necessary libraries. You may put the function header given above in Q3.h while testing, but you will not be submitting your .h file.