PIC 10A/2 - Winter 2022 Coding Task (20 marks)

Complete the task outlined below and submit your files, along with **Honesty.txt** (as described in the Final Exam instructions), to the Gradescope submission portal. Your code must compile and work on Visual Studio 2022.

At the end of this task, you should submit

- **CI.h** (must include the declarations of all free functions, constructors, and member functions);
- **CI.cpp** (must include the definition of all free functions, constructors, and member functions); and
- the honesty statement.

Scoring for this task is the same as for the homework. In particular, the **HW_Codes** items should be followed.

This problem is about analyzing a file and its words, without regard to uppercase or lowercase lettering. In this problem **a word is any sequence of non-whitespace characters**, so it includes things like "???#", "111.23", "!", etc.

First, write a **CI_String** class and associated functions. The class should

- have a member variable **str** of type **std::string** to store a string of all lowercase letters;
- have a constructor that can take an **std::string** (possibly with a mix of uppercase and lowercase letters) and only store the string with all lowercase letters in **str**;
- have a **get_string** accessor function to return **str**;
- have a **size** accessor function to return the size of **str**;
- have an overloaded **operator<** (as a free function) to compare the **std::string**s of two **Ci_String**s; and
- have an overloaded **operator<<** (as a free function) to print the **std::string** within a **CI_String**.

To help with this, you **must write and use**

- a function **make_lower** that can receive an **std::string** as an input and returns an **std::string** with all the letters lowercase.

As a reminder, you cannot use any libraries or functions not covered in this course on this exam. As a hint: if **c** is an uppercase **char** (so **'A'** <= **c** and **c** <= **'Z'**) then its lowercase version can be found as **c** + **'a'** - **'A'**. There are some subtleties here but they should not play a role in this task.

With this class, the file can be analyzed for its unique words, where we consider two words the same if they have the same letters in the same order, but ignoring capitalization, e.g., "APPLE" and "appLe" and "apple" are all considered the same. With the class and functions above, write the following free functions:

- **total_chars** that as an input receives *the name* of a file and returns the total number of characters (including whitespace) in the file. If the file is empty or does not exist, it should return 0.
- **unique_words** that as an input receives *the name* of a file and returns all the unique words of the file as an **std::set<CI_String>**. If the file is empty or does not exist, the returned set should be empty.
- **uniques_per_line** that as an input receives *the name* of a file and returns an **std::vector** whose contents are the number of *unique* words, *ignoring capitalization*, in each respective line. If the file is empty or does not exist, the returned set should be empty.

- **print_line_one_info** that as an input receives *the name* of a file and then prints each unique word of line 1 in lexicographic order with its multiplicity. Each word should appear on its own line and be of the format
  "[WORD] occurs 1 time." (if the word appears only once) or
  "[WORD] occurs [NUMBER] times." (if the word appears more than once)
  where [WORD] is the word (lowercase) and [NUMBER] is the number of times it does appear. If the file is empty or does not exist, there should be nothing printed.

As words are printed/displayed, only their all-lowercase representation should be displayed.

You **may assume:**

- any files to analyze are stored in the default storage directory for Visual Studio, i.e., the files are stored in the same directory as the .cpp/.h files.
- the files will not end in whitespace.

The following main routine should produce the output shown where **demo.txt** is the file provided and **no_file.txt** is a file that does not exist.

```
#include "CI.h"
#include <iostream>
#include <string>

int main()
{
    std::cout << make_lower("hELLo") << '\n';

    CI_String s("PIC 10A");
    std::cout << s << '\n';
    std::cout << s.size() << '\n';
    std::cout << ("CS" < s) << '\n';

    const std::string file = "demo.txt";
    auto char_count = total_chars(file);
    auto line_info = uniques_per_line(file);
    auto words = unique_words(file);

    std::cout << "In " << file << " there are " << char_count << " characters.\n";
    std::cout << "The unique words are:\n";
    for (const auto& w : words) {
        std::cout << w << '\n';
    }
    std::cout << "Line by line, the number of unique words are: ";
    for (auto s : line_info) {
        std::cout << s << ' ';
    }
    std::cout << '\n';
    std::cout << "Line 1 info:\n";
    print_line_one_info(file);

    // when file does not exist
    const std::string bad_file = "no_file.txt";
    auto char_count_bad = total_chars(bad_file);
    auto line_info_bad = uniques_per_line(bad_file);
    auto words_bad = unique_words(bad_file);

    std::cout << "In " << bad_file << " there are " << char_count_bad << " characters.\n";
    std::cout << "The unique words are:\n";
    for (const auto& w : words_bad) {
        std::cout << w << '\n';
    }
    std::cout << "Line by line, the number of unique words are: ";
    for (auto s : line_info_bad) {
        std::cout << s << ' ';
    }
    std::cout << '\n';
    std::cout << "Line 1 info:\n";
    print_line_one_info(bad_file);

    return 0;
}
```

Desired output:

```
hello
pic 10a
7
1
In demo.txt there are 89 characters.
The unique words are:
...!~#
3.14159
?
c@t
cat
coffee
dog
ic3
ice
puppy
tea
water
Line by line, the number of unique words are: 5 5 3 2
Line 1 info:
...!~# occurs 1 time.
c@t occurs 1 time.
cat occurs 2 times.
dog occurs 1 time.
puppy occurs 1 time.
In no_file.txt there are 0 characters.
The unique words are:
Line by line, the number of unique words are:
Line 1 info:
```