1 Assignment 7 Introduction

This assignment is due *before* 11:59pm on the listed date. Which means, submissions made *on or after* 11:59pm will be counted as a late submissions. Late submissions *before* 11:59pm on the following day will receive a 25% point deduction as penalty. Submissions made *on or after* 11:59pm on the same day will *not* be graded. We strongly recommend completing the assignment before then.

It is imperative that you meticulously follow the submission process outlined at the end of this assignment. Incorrectly structured submissions will receive a 10% point deduction as a penalty.

Assignments due on Mondays generally involve materials covered in lectures from the previous week, whereas assignments due on Thursdays involve materials covered in lectures from the running week. So be sure to watch the lectures and go over the reading materials before attempting the assignments.

Good luck!

2 Baby Vectors (40 points)

Write a class named `BabyVector`, that generally acts like a regular vector, and supports the `size`, `at`, and `push_back` methods. This class only needs to support handling integers.

Unlike regular vectors however, a `BabyVector` ignores the first time it encounters a value, and only stores them when the said value is repeated (i.e. provided again).

Example usage:

```
BabyVector waah;

waah.push_back(1);
waah.push_back(2);
waah.push_back(3);
waah.push_back(2);

// Currently only has [2] stored, as it forgot the first occurrences of 1, 2, and 3.

assert(waah.size() == 1);

// The above should yield true as its size is currently 1.

waah.push_back(1);
assert(waah.at(0) == 2);
assert(waah.at(1) == 1);

// The above should yield true in both cases, as it now has [2, 1] stored due to
   seeing 1 for the second time.
```

3 Censored Copy (60 points)

Write a function named `ReplacementCensor` in a `censor` implementation and header file, that takes 3 function arguments. The first argument is an `istream` of text to be processed. The second argument is an `ostream` to where the processed text should be copied to. The last argument is a `map` of strings to strings, where the key is a string that needs to be replaced with the value in the text of the first argument's contents. This function should return a `set` containing the words that were replaced. The keys should be matched in a case insensitive manner, and the text to replace may not be white space delimited, meaning they can appear in longer words.

Example `istream` (First argument):

```
note: this is a line with multiple WORds that should be rePLACEd.
 all instances of word eveninlargerWordsshould be repLAced.
```

Example `map` (Third argument):

```
{"word", "Grouped-Letter-Unit"},
{"be", "wasp"},
{"not found", "not appearing"},
{"PlaCe", "LoCation"}
```

Expected `ostream` (Second argument) after being processed:

```
note: this is a line with multiple Grouped-Letter-Units that should wasp
    reLoCationd.
 all instances of Grouped-Letter-Unit eveninlargerGrouped-Letter-Unitsshould wasp
     reLoCationd.
```

Expected `set` returned by `ReplacementCensor`:

```
{"PLACE", "WORd", "Word", "be", "pLAce", "word"}
```

A tester for this particular example is provided below:

```
#include "censor.h"
#include <map>
#include <set>
#include <string>
#include <iostream>
#include <sstream>
#include <cassert>

int main() {
  std::map < std::string, std::string > bad_to_good = {
    {"word", "Grouped-Letter-Unit"},
    {"be", "wasp"},
    {"not found", "not appearing"},
    {"PlaCe", "LoCation"}
  };
  std::istringstream iss("note: this is a line with multiple WORds that should be
      rePLACEd.\n all instances of word eveninlargerWordsshould be repLAced.");
  std::ostringstream oss;
  std::set < std::string > result = ReplacementCensor(iss, oss, bad_to_good);
  std::set < std::string > expected_return = {
    "PLACE",
    "WORd",
    "Word",
    "be",
    "pLAce",
    "word"
  };
  assert(result == expected_return);
  assert(oss.str() == "note: this is a line with multiple Grouped-Letter-Units that
      should wasp reLoCationd.\n all instances of Grouped-Letter-Unit
      eveninlargerGrouped-Letter-Unitsshould wasp reLoCationd.");
}
```

4 Assignment 7 Submission Process

- Create a folder, name it `your_msu_id7`. For example, if your MSU email is `johndoe@msu.edu`, then you should name the folder `johndoe7`.

- For each programming task, create a sub-folder inside your `your_msu_id7` folder, and name it as the number that corresponds to the programming task number. For this assignment, there should be two sub-folders named '2' and '3'.

- Inside each sub-folder, put the `main.cpp` (along with any necessary header files) for the appropriate solution. While you were not explicitely asked to create a `main.cpp` file in this assignment, you should still create them that use your implementation and header files. When grading, we will be using our own `main.cpp` files that include your implementation and header files.

- Compress/Zip `your_msu_id7` folder and name it `your_msu_id7.zip`. For example, if the name of your folder is `johndoe7`, then you need to create a zip file named `johndoe7.zip`. Zip file guide: `https://copyrightservice.co.uk/reg/creating-zip-files`.

- Submit the zip file through D2L.