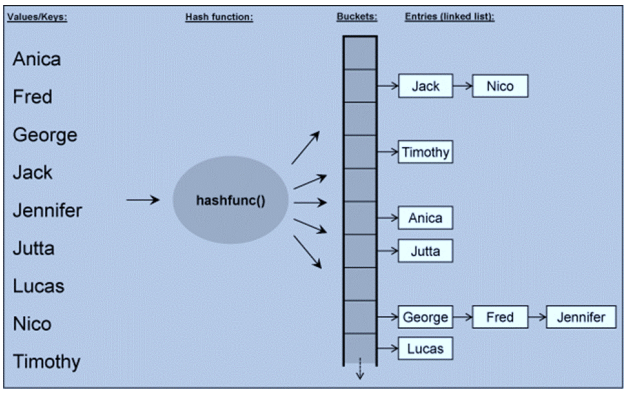
CPSC 131, Data Structures – Spring 2022
Word Counting: Unordered Containers Homework

Learning Goals:

- Familiarization and practice with key/value association data structure usage and hash table concepts
- Familiarization and practice using the STL's unordered_map container interface
- Reinforce the similarities and differences between sequence and associative containers
- Reinforce reading persistent data from disk files and storing in memory resident data structures
- Reinforce modern C++ object-oriented programming techniques



Description:

This project analyzes words in English text using a hash table to store words. The main task is to count the occurrences of each word in a book - called its word frequency. Text analysis using word frequencies is used in linguistics. In this project, ExtendedBook inherits and extends our previous project's Book class. You are given the public interface of ExtendedBook. Your task is to complete the class implementation. You are provided with starter code that forms your point of departure to complete this assignment:

1. **main.cpp** – This file is provided, requires no modifications, and will be overwritten during the grading process. Function main() orchestrates the flow of execution and tests your intermediate results along the way.

2. **ExtendedBook.hpp/ExtendedBook.cpp** – The class has a single member attribute of type std::unordered_map, which is the C++ Standard Library's implementation of a hash table, to store the association of words (key) to the number of times that word occurs (value).

   a. *ExtendedBook* – This (default) constructor takes the same arguments as its base class Book. This function is to

      i. Open a disk file containing this book's content as an input file stream. The name of the file, if it exists, is *ISBN.bok* where ISBN is the book's 10 or 13 digit ISBN. For example, 9789999275842.bok contains the text of book with an ISBN of "9789999275842" titled "Billboard Music Week (1st edition)" by "Joel Whitburn".

      ii. Read a single word at a time from the input stream until end of file. Words are delimited by whitespace as defined in standard C++.

      iii. For each word read, accumulate the number of times that sanitized word has appeared in the text as the word's frequency.

   Avoid setting the base class's attributes within the body of this function. Instead, be sure to use a Constructor Member Initialization List to construct the base class passing the proper values.

Constraint: Only "sanitized" words shall be added to the hash table. For example, leading and trailing punctuation, parentheses, brackets, etc. should be removed, but intra-word punctuation should remain. A working sanitize function has been provided.

b. *numberOfWords* – This function takes no arguments and returns the number of unique words.

c. *wordCount* – This function takes a constant reference to a standard string as a parameter and returns the frequency of occurrence of that sanitized word, or zero if the word is not found in the hash table.

d. *mostFrequentWord* – This function takes no arguments and returns the most frequent word, or the empty string if the hash table is empty.

e. *maxBucketSize* – This function takes no arguments and returns the size of the largest bucket in the hash table. See the unordered_map's bucket interface at https://en.cppreference.com/w/cpp/container/unordered_map

Rules and Constraints:

1. You are to modify only designated TO-DO sections. The grading process will detect and discard any changes made outside the designated TO-DO sections, including spacing and formatting. Designated TO-DO sections are identified with the following comments:

```
//////////////////////// TO-DO (X) ////////////////////////////////
...
//////////////////////// END-TO-DO (X) ////////////////////////////
```

Keep and do not alter these comments. Insert your code between them. In this assignment, there are 6 such sections of code you are being asked to complete. 1 of them is in ExtendedBook.hpp and 5 are in ExtendedBook.cpp.

Reminders:

- The C++ using directive `using namespace std;` is **never allowed** in any header or source file in any deliverable product. Being new to C++, you may have used this in the past. If you haven't done so already, it's now time to shed this crutch and fully decorate your identifiers.
- It is far better to deliver a marginally incomplete product that compiles error and warning free than to deliver a lot of work that does not compile. A delivery that does not compile clean may get filtered away before ever reaching the instructor for grading. It doesn't matter how pretty the vase was, if it's broken nobody will buy it.
- Use Build.sh on Tuffix to compile and link your program. The grading tools use it, so if you want to know if you compile error and warning free (a prerequisite to earn credit) than you too should use it.
- Filenames are case sensitive on Linux operating systems, like Tuffix.
- You may redirect standard input from a text file, and you must redirect standard output to a text file named output.txt. Failure to include output.txt in your delivery indicates you were not able to execute your program and will be scored accordingly. A screenshot of your terminal window is not acceptable. See How to build and run your programs. Also see How to use command redirection under Linux if you are unfamiliar with command line redirection.

Deliverable Artifacts:

Provided files	Files to deliver	Comments
main.cpp Book.hpp	1. main.cpp 2. Book.hpp	You shall not modify these files. The grading process will overwrite whatever you deliver with the one provided with this assignment. It is important that you deliver complete solutions, so don't omit these files.
Book.cpp	3. Book.cpp	Replace with your (potentially updated) file from the previous assignment.
ExtendedBook.hpp ExtendedBook.cpp	4. ExtendedBook.hpp 5. ExtendedBook.cpp	Start with the files provided. Make your changes in the designated TO-DO sections (only). The grading process will detect and discard all other changes.
sample_output.txt	6. output.txt	Capture your program's output to this text file using command line redirection. See command redirection. Failure to deliver this file indicates you could not get your program to execute. Screenshots or terminal window log files are not permitted.
	readme.*	Optional. Use it to communicate your thoughts to the grader
9789998302938.bok 9789998819450.bok		Text files to be used as program input. Do not modify these files. They're big and unchanged, so don't include them in your delivery.
CheckResults.hpp BookTests.cpp		When you're far enough along and ready to have your classes tested, then place these files in your working directory. These tests will be added to your delivery and executed during the grading process.