Assignment 3: Word Count Wizard

Data Structures (CS-UH 1050) — Spring 2023

1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment.

Any documents and program code that you submit must be fully written by yourself. You can discuss your work with fellow students, as long as **these discussions are restricted to general solution techniques, without sharing the overall or specific algorithms.** Put differently, these discussions should not be about concrete code you are writing, nor about specific set of steps or results you wish to submit. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution, meaning you are also not allowed to possess solutions by someone from a different year or course, by someone from another university, or code from the Internet, etc. This also implies that **there is never a valid reason to share your code with fellow students, and that there is no valid reason to publish your code online in any form as long as you are a student at NYUAD**. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g., if the solution matches with high similarity score that of others), **the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi** (see https://students.nyuad.nyu.edu/campus-life/community-standards/policies/academic-integrity/) under the integrity review process.

2 Introduction

In this assignment, you are expected to design and implement a Word Count Wizard! The system should be able to count the number of unique words in the entire text provided by the user, return the frequency of any word in that given text, among other operations. The wizard should take the input text and organize it in a hash table. Further instructions are described in the coming section.

You are supposed to create the system utilizing object-oriented programming (OOP) principles and appropriate data structures discussed in class. **You should implement all the data structures manually, STL based containers are not allowed to be used.**

3 Implementation

Commands to implement:

Loading: When the Word Count Wizard starts, the user is greeted and asked to provide the full path of an input file.

Analytics: Next, the interface will display the count of collisions, the count of unique words and the total count of words using the default hash function, followed by a list of all possible actions to invoke by the user, such as: Finding the frequency of a given word, finding the word with the highest frequency (optional), and quitting the system. **Moreover, there is an optional choice of a hash function** (the user is presented with a list of five possible hash functions to choose from). If no choice is made (i.e., click enter directly), the default hash function will be used by the system.

Instructions:

1. The system should be able to read and extract words (strings) from the input file and load them into a hash table.
2. Exploit the **hash table** data structure in a way that allows you to identify and store the unique words and their frequencies in the Wizard system. ***You should determine the most suitable structure of an (key, value) entry within the hash table on your own; what a key should represent and what a value should represent.***
3. Use **separate chaining** to handle collisions.
4. Choose/design and implement **six different hash functions**, and determine which one should be **set as the default hash function** in the system. **You need to document both the design in pseudo code and the implementation in C++ of each of the six hash functions along with their experimental results in a pdf report as follows:**
    - You have to evaluate the performance of each hash function in terms of the number of collisions when applied on the same dataset files (use the supplementary files). The hash function causing the least number of collisions **on average** across all the files should be set as the default hash function to use by your system.
    - You have to document the evaluation results of each hash function in a report.
    - A user should also be able to choose one of the six implemented hash functions to be used by the system

Hints for text tokenization:

- Split each sentence by whitespace in order to extract words.
- Ignore punctuation marks (“.”, “,”, “\” , “!”, “?”, “;”) attached to the end of a word.
- Ignore cases (you may use the function **tolower()** for that purpose)
    - For example, "Token?" should be counted as an instance of "token".

Datasets:

- A large collection of stories (101) in a TXT format is included for you to test your work, and evaluate your six hash functions.

User Interface:

The application interacts with the user using a command line (terminal based) interface as shown below:

```
======================================================================
Welcome to the Word Count Wizard!
List of available Commands:
import <path>           :Import a TXT file
select_hashFun          :Choose a hash function
count_collisions        :Print the number of collisions
count_unique_words      :Print the number of unique words
count_words             :Print the the total number of words
find_freq <word>        :Search for a word and return its frequency
find_max                :Print the word with the highest frequency
exit                    :Exit the program
======================================================================

Please provide the path to the TXT file you wish to analyze:
>
```

The application, after performing the user's desired operation, presents the menu again for the next operation to be performed.

```
======================================================================
Welcome to the Word Count Wizard!
List of available Commands:
import <path>           :Import a TXT file
select_hashFun          :Choose a hash function
count_collisions        :Print the number of collisions
count_unique_words      :Print the number of unique words
count_words             :Print the the total number of words
find_freq <word>        :Search for a word and return its frequency
find_max                :Print the word with the highest frequency
exit                    :Exit the program
======================================================================

Please provide the path to the TXT file you wish to analyze:
> user/data/text.txt
Processing ...
The number of collisions is: 17
The number of unique words is: 180
The total number of words is: 320

======================================================================
List of available commands:
import <path>           :Import a TXT file
select_hashFun          :Choose a hash function
count_collisions        :Print the number of collisions
count_unique_words      :Print the number of unique words
count_words             :Print the the total number of words
find_freq <word>        :Search for a word and return its frequency
find_max                :Print the word with the highest frequency
exit                    :Exit the program
======================================================================
```

NOTE: Make sure to handle the cases of the input provided by the user, for example you can use the `tolower()` function.

Features to Implement:

- **import *<path>***
  - loads a txt file into the hash table
  - Then, displays the count of collisions, the count of unique words and the total count of words using the default hash function

```
> import /user/data/txt
Done!

The number of collisions is: 17
The number of unique words is: 180
The total number of words is: 320
```

- **count_collisions**
  - returns the number of collisions caused by a hash function.

```
> count_collisions
The number of collisions is: 17
```

- **count_unique_words**
  - returns the number of unique words in the entire text.

```
> count_unique_words
The number of unique words is: 180
```

- **count_words**
  - returns the total number of words in the text.

```
> count_words
The total number of words is: 320
```

- **select_hashFun**
  - allows the user to choose one of the six implemented hash functions to be used by the system to construct the hash table.
  - Then, displays the count of collisions, the count of unique words and the total count of words using the selected hash function (as in the import command)

```
> select_hashFun
Enter the hash function ID: [1, 2, 3, 4, 5, 6]
> 1
Hash function 1 has been chosen successfully!

The number of collisions is: 17
The number of unique words is: 180
The total number of words is: 320
```

```
> select_hashFun
Enter the hash function ID: [1, 2, 3, 4, 5, 6]
>
The default hash function has been chosen!

The number of collisions is: 17
The number of unique words is: 180
The total number of words is: 320
```

- **find_freq *<word>***
  - searches for a word and returns its frequency, if it is found. Otherwise, it will return zero.

```
> find_freq LIFE
The frequency of the word "life" is: 7
```

```
> find_freq VIOLATION
The frequency of the word "violation" is: 0
```

- **[Bonus Feature] find_max**
  - returns the word with the highest frequency. The running time of the method should be O(1) in the worst case scenario.
  - Note: Use the **Heap** data structure to represent the set of entries.

```
> find_max
The word with the highest frequency is: music
```

- **exit**
  - exits the system.

4 Submission and Grading

Description	**Score (/20)**
Quality in Code Organization & Modularity - Defining ALL the needed classes correctly, with their sets of attributes and methods (including the constructors and destructors), and creating objects from each class properly (2 pts) - Proper initiation and termination of the system (1 pt) - Properly dividing the program into [*.cpp, *.h, makefile], compressed then into a single zip file (1 pt) - STL based containers are not allowed to be used (-2 pts in case this is not met)	4
Clear and complete report (in pdf format) of the documentation and experimental results exploring the performance of the six hash functions when applied to the same supplementary TXT datasets: - in terms of the average number of collisions in overall per function to determine the default hash function to use by your system in case the user doesn't choose one (2 pts) - And documenting both the design (in pseudo code) and the implementation (in C++) of each of the six hash functions in the report (2 pts)	4
Correct design and implementation of the required methods: import (1 pt), select_hashFun (1 pt), count_unique_words (2 pts), count_words (2 pts), find_freq (2 pts), count_collisions (2 pts)	10
Proper error handling of missing and invalid input, etc.	1
Properly commenting the code (i.e., code documentation)	1
[Bonus Feature] implementing the **find_max()** method with O(1) running time, while using the **Heap** data structure to represent the set of entries	1

Extra points are used to pad your score up to the maximum score (i.e., 20 points).

You should solve and **work individually** on this assignment. The deadline of this assignment is **in 12 days of its release** on NYU Brightspace.

You should compress your **C++ source files** (*.cpp, *.h, makefile) in to **a single zip file** before uploading it directly to NYU Brightspace. The zip file should contain:

1. myList.h
2. myList.cpp
3. dochashtable.h
4. dochashtable.cpp

5. makefile
6. main.cpp
7. and the PDF report

NO SUBMISSIONS OR RESUBMISSIONS VIA EMAIL WILL BE ACCEPTED. Note that your program should be implemented in C++ and **must be runnable on the Linux, Unix or macOS operating system**. Late submissions will be accepted **only up to 2 days late**, afterwards you will receive zero points. For late submissions, 5% will be deducted from the homework grade per late day.

Make sure to **FOLLOW THE ASSIGNMENT SUBMISSION AND EXTENSION PROTOCOLS** that are shared as part of the syllabus, the course logistics and Lecture#1 on NYU Brightspace, in addition to **THE ACADEMIC INTEGRITY POLICY** shared via the code-of-conduct, syllabus, the course logistics, Lecture#1 and the discussions on NYU Brightspace.