

Assignment 3: Multilingual Dictionary “Translator”

Data Structures (CS-UH 1050) — Spring 2025

1 Code of Conduct

All assignments are graded and we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we 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 details. 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. Discussions with others should not lead to you possessing the complete or partial solution of others in any form (paper or digital), regardless of who made the solution. 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. 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

The objective of this assignment is to develop a multilingual dictionary called “translator”. In this assignment you will implement a dictionary using hash tables, offering efficient storage and retrieval of word translations in different languages. This dictionary program utilizes hash functions for indexing, allowing fast access to entries and handling collisions through linear or quadratic probing. The dictionary offers a user-friendly interface for importing, adding, deleting, searching, and exporting dictionary entries. Additionally, the

program implements lazy deletion to optimize memory management during removal operations. With its comprehensive functionality and efficient data structure, this dictionary program must provide a robust solution for managing multilingual word translations. **You should not use STL hash-table hash-map classes, instead, you should implement the hash table in C++.** You may however use the **STL vector** class in this assignment. The program should be **case-insensitive** i.e. the commands **find** or **Find** should be treated the same. “find hi” and “Find HI” or “FIND HI” must return the same answers. The words and their translations should also be case-insensitive. The words “hello”, “Hello”, or “HELLO” etc. should be considered as the same word.

3 Implementation

The program consists of the following components:

Class HashTable

The **HashTable** class is the backbone of the dictionary implementation. The class implements a hash table for storing words and their translations in different languages. The HashTable class provides essential operations for managing dictionary entries efficiently, including insertion, deletion, search, and export. It uses a hash function to map words to indices in the array of buckets. Your hash function may use techniques, such as polynomial hashing, cyclic shift hashing, division and MAD methods. You are required to explore and document the performance of at least 3 different hash functions in terms of the average number of collisions when applied to the provided sample files to choose the optimal hash function for your system. Collisions should be handled using **linear** or **quadratic** probing. The class must also ensure memory management by deallocating memory in the destructor, preventing memory leaks.

The following detailed explanation should give you a comprehensive understanding of how the **HashTable** class functions within the dictionary implementation.

Private Member Variables:

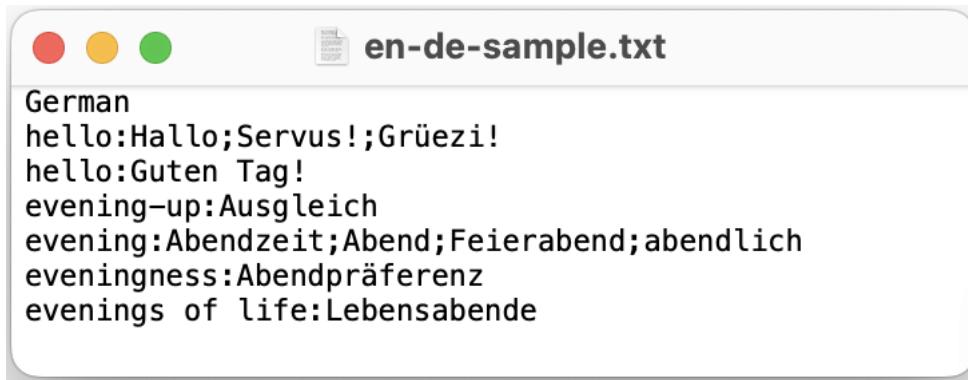
- **buckets:** An array of pointers to **Entry** objects representing the buckets in the hash table.
- **capacity:** The maximum number of buckets in the hash table.
- **size:** The current number of elements stored in the hash table.
- **collisions:** The number of collisions that have occurred during insertion.

Constructor:

- **HashTable(int capacity):** Initializes the hash table with a given capacity. It should dynamically allocate an array of pointers to **Entry** objects, where each pointer represents a bucket in the hash table. Initially, all buckets should be set to **nullptr**. The constructor should also initialize other member variables like size, capacity and, and collisions.

Public Member Functions:

1. **getSize():** Returns the number of elements currently stored in the hash table.
2. **getCollisions():** Returns the number of collisions that have occurred during insertion.
3. **hashCode(string word):** Computes the hash code for a given word using some hashing technique (e.g. polynomial hashing, cyclic shift etc.), depending on the implemented hash function, it should also ensure that the hash code is within the bounds of the table's capacity.
4. **import(string path):** Reads dictionary entries from a text file specified by the given path and inserts them into the hash table. It expects the file format to have entries in the form of "word:meaning1;meaning2;meaning3;...", where the word is an English language word (or a sentence) and meaning(s) is/are its translation in another language. There can be one or multiple meanings of the same word in a single line separated by semicolon(;) or possibly also in different lines. Following is an example of a dictionary file.



The first line of the file contains the name of the language. The words or sentences in the Dictionary file are not necessarily sorted.

The import method can be invoked from the import command of the user interface.

```

|bash-3.2$ ./translator
820324 German words have been imported successfully.
=====
Size of HashTable          = 812746
Total Number of Collisions = 936025
Avg. Number of Collisions/Entry = 1.2
=====
find <word>                : Search a word and its meanings in the dictionary.
import <path>               : Import a dictionary file.
add <word:meaning(s):language> : Add a word and/or its meanings (separated by ;) to the dictionary.
delTranslation <word:language> : Delete a specific translation of a word from the dictionary.
delMeaning <word:meaning:language> : Delete only a specific meaning of a word from the dictionary.
delWord <word>               : Delete a word and its all translations from the dictionary.
export <language:filename>   : Export a given language dictionary to a file.
exit                         : Exit the program
>import en-fr.txt
110323 French words have been imported successfully.
>import en-es.txt
31410 Spanish words have been imported successfully.
>|

```

You will be provided with the following three sample translation files:

- en-de.txt: English to German translation of around 820323 words.
 - This file is automatically imported upon startup of the program
- en-fr.txt: English to French translation of around 110322 words.
- en-es.txt: English to Spanish translation of around 31409 words.

Disclaimer: The dictionary files may contain wrong translations.

5. **insert(string word, string meanings, string language):** Inserts a new word along with its translation/meanings in a specific language into the hash table. It gets the hash code for the word, handles collisions using linear or quadratic probing, and inserts the entry into the appropriate bucket. If the word already exists in the dictionary then new translations or meanings must be added to it. The method should also accept multiple meanings separated by semicolons. If a meaning already exists then it should not be added again. The insert method can be invoked using **add** command from from the user interface. The method can also be reused/called from other methods e.g. import.

```

>add UAE:Vereinigte Arabische Emirate:German
>add UAE:Émirats arabes unis;UAE:French
>find UAE
UAE found in the Dictionary after 12 comparisons.
German    : Vereinigte Arabische Emirate
French    : Émirats arabes unis; UAE
>|

```

6. **find(string word):** Searches for a word in the hash table and prints its all translations if found. The find method should also print how many comparisons have been made

in the hash table to find the word. The find method can be invoked from the user interface using the following **find** command.

```
>find Success
Success found in the Dictionary after 2 comparisons.
German : Erfolg; Gelingen; Sukzess; Sukzeß; Triumph; Ausgang; Folge;
French : réussite; succès
Spanish : éxito
>find nyuad
nyuad not found in the Dictionary.
```

7. **delWord(string word)**: This method should delete a word and its all translations from the dictionary (hash table) using the “**lazy deletion**” approach. Lazy deletion is a technique used in hash tables to efficiently handle the removal of elements while minimizing the overhead of reorganizing the table. Rather than immediately removing an element from the table when requested, lazy deletion involves marking the element as “deleted” and postponing the actual removal until a later time. Entries marked as deleted are skipped during searches, ensuring that they do not affect the correctness of dictionary operations.

```
>delWord UAE
UAE has been successfully deleted from the Dictionary.
>find UAE
UAE not found in the Dictionary.
>delWord UAE
UAE not found in the Dictionary.
>
```

8. **delTranslation(string word, string language)**: Deletes an entire translation of a word in a specific language from the hash table

```
>delTranslation Success:Spanish
Translation has been successfully deleted from the Dictionary.
>find Success
Success found in the Dictionary after 2 comparisons.
German : Erfolg; Gelingen; Sukzess; Sukzeß; Triumph; Ausgang; Folge; Gedeihen;
French : réussite; succès
>
```

9. **delMeaning(string word, string meaning, string language)**: Deletes a specific meaning of a word in a specific language from the hash table. If the only meaning of a word in a particular language has been deleted, then the entire translations in that language must also be deleted.

```
>delMeaning success:Triumph:German
Meaning has been successfully deleted from the Translation
>find success
success found in the Dictionary after 2 comparisons.
German   : Erfolg; Gelingen; Sukzess; Sukzeß; Ausgang; Folge; Gedeihen; guter Ausgang; Erfolge
French   : réussite; succès
>
```

10. **exportData(string language, string filePath)**: Exports dictionary entries in a specific language to a file specified by the file path. Make sure that the exported file should be in the same format as the input files so that it can be imported by the program if/when needed. This method can be invoked using the **export** command from the user interface.

```
>export German:en-2-de.txt
812747 records have been successfully exported to en-2-de.txt
>
```

11. **~HashTable()**: The Destructor deallocates memory used by the hash table, including all entry objects and the array of buckets.

Class Entry

The Entry class in this project represents an individual entry in the dictionary hash table. The Entry class encapsulates a word along with its translations in different languages. It supports adding translations dynamically using the addTranslation method, which handles both adding new translations and appending meanings to existing translations. The Entry class provides a container for storing dictionary entries, allowing for efficient management and retrieval of words and their translations within the hash table.

Entry class has the following structure and functionality.

Attributes:

- **word**: A string representing the English word in the dictionary.
- **translations**: A list of Translation objects, each containing translations of the word in a different language.
- **deleted**: A boolean flag indicating whether the entry has been marked as deleted (used for lazy deletion).

Constructor:

- **Entry(string word, string meanings, string language)**: Initializes an Entry object with the given word, its translation, and the language of translation. It creates a Translation object with the provided meanings and language and adds it to the translations list. The deleted flag must be set to false by default.

Public Member Functions:

1. addTranslation(string newMeanings, string language):

Adds a translation of the word in the specified language to the translations list. If a translation in the same language already exists, the new meanings should be appended to the existing translation. If the language is new, a new Translation object must be created and added to the list. Do not add the same meaning again if it is already present for the given language.

2. print():

Prints all translations of a word in the following format.

```
German    : Mater; Muttertier; Mutter; Mütterchen; Tiermutter
French    : mère
Spanish   : madre
```

Class Translation

The Translation class provides a structured representation of translations for a word in a specific language, facilitating efficient management and retrieval of translations within the dictionary (hash table). It allows for the addition of new meanings dynamically using the addMeaning method.

Following is the list of attributes and methods of the Translation class

Attributes:

- language:** A string representing the language of the translations.
- meanings:** A list of strings representing the meanings or translations of the word in a specific language (separated by ;).

Constructor:

- **Translation(string meanings, string language):** Initializes a Translation object with the given meanings and language. It adds each meaning to the meanings list.

Public Member Functions:

1. addMeaning(string newMeanings):

- Adds a new meaning or meanings (separated by ;) to the meanings list.
- This method should not add a meaning if it already exists in the meanings list.

4 Grading

Description	Score (/20)
Quality in Code Organization & Modularity <ul style="list-style-type: none">- Defining ALL the needed classes correctly, with their sets of attributes and methods (including the constructors and destructors)- Creating objects from each class properly- Proper initiation and termination of the system	3
Implementing all required commands correctly	11
Proper error handling of missing and invalid input, etc.	1
Properly commenting the code (i.e., code documentation)	2
Clear and complete report (in pdf format) of the documentation and experimental results exploring the performance (avg. number of collisions) of at least 3 hash functions when applied to the same supplementary datasets. (2 pts) Documenting both the design (in pseudo code) and the implementation (in C++) of each of the hash functions in the report (1 pt)	3

You should solve and **work individually** on this assignment.

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

1. hashtable.cpp
2. hashtable.h
3. makefile
4. main.cpp
5. vector.h (if you prefer to use your own vector implementation)
6. A PDF report of the hash function evaluation

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 3 days late**, afterwards you will receive zero points. For late submissions, 5% will be deducted from the assignment grade per late day.