C++ Implementation of Hash table with linear probing (worth 10%, due Nov 5th 23:59PM, late submissions not accepted)

Mingyu Guo

1 Task Description

You are asked to use C++ to implement

- Hash table with linear probing

2 Submission Guideline

You must follow this guideline! Your submission will be marked automatically. Failure to follow this guideline will result in 0.

Your submission should contain exactly two files: `main.cpp` and `test.txt`.

You do not need to submit a design.

You are asked to implement a very specific hash table. The keys are lower-case English words (e.g., apple, pear). The length of a key is at most 10. The hash function is "simply using the last character". That is, the hash value of apple should be e, and the hash value of pear should be r. Your hash table contains exactly 26 slots (hash value a to hash value z). The total number of English words/keys you need to deal with is at most 26, so the table is never too small.

A table slot has three different statuses: "never used", "tombstone", and "occupied". Table starts with 26 "never used" slots.

Searching works as follows: given a key, take its last character as the hash value. First try the corresponding table slot, if the objective key is there, then you have found it. If the corresponding slot is never used, terminate because we are certain that the objective is not in the table. If the slot is occupied but it's not the objective, or the slot is a "tombstone", then we move on to the next slot (may need to wrap around the table if the current slot is the last one). We keep trying until we either find the key or are certain that the key does not exist in the table.

Insertion works as follows: given a key, take its last character as the hash value. If the corresponding table slot is not occupied (either "never used" or "tombstone"), put the key there (the slot is now occupied). If the corresponding slot is already occupied, try the next slot. Repeat trying until you find an unoccupied slot. During the above process, if you see that the key is already in the table, then terminate the insertion process and change nothing.

Deletion works as follows: given a key, use the searching process to locate its slot. (If the key is not in the table, then do nothing.) Once you find the key, change the slot status to "tombstone".

You should start your program by initializing an empty hash table. Your program takes one line as input. The input line contains n "modification moves" separated by spaces ($1 \leq n \leq 26$). The available modification moves are

- `AWord` (Character `A` followed by a lower-case English word of length at most 10): `Aapple` means insert key apple into the hash table. If apple is already in the table, do nothing.
- `DWord` (Character `D` followed by a lower-case English word of length at most 10): `Dapple` means delete key apple from the hash table. If apple is not in the tree, do nothing.

At the end, you need to go through the slots from a to z, and output all the keys separated by space.

You don't need to worry about invalid inputs.

Sample input 1: `Aaaa Accc Abbb`

Sample output 1: `aaa bbb ccc`

Sample input 2: `Abba Aaaa Acca`

Sample output 2: `bba aaa cca`

Sample input 3: `Abba Aaaa Acca Daaa`

Sample output 3: `bba cca`

You are responsible for writing your own test cases. Please submit `test.txt`. This file should contain up to 100 lines. Each line must start with a sequence of modification moves and then followed by the correct output.

For example, the following three lines represent the above three samples.

```
Aaaa Accc Abbb aaa bbb ccc
Abba Aaaa Acca bba aaa cca
Abba Aaaa Acca Daaa bba cca
```

3 Marking

Marking will be done automatically. The total mark is 10.

- 8 marks for code correctness: Your code will be tested against a deterministic subset of test cases submitted by the students.
  - If your code passes all test cases, then you receive 8 marks.
  - If your code passes 90% test cases, then you receive 7 marks.
  - If your code passes 80% test cases, then you receive 6 marks.
  - If your code passes 70% test cases, then you receive 5 marks.
  - If your code passes 60% test cases, then you receive 4 marks.
  - If your code passes 50% test cases, then you receive 3 marks.
  - If your code passes 40% test cases, then you receive 2 marks.
  - If your code passes 30% test cases, then you receive 1 mark.
- 2 marks for test case quality: If your test cases contain mistakes, then your receive 0 marks. We will run your test cases against a deterministic subset of students' submissions. We keep track of how many submissions you test cases can break (by "breaking" a submission, I mean at least one of your test cases proves that the submission is buggy). Let x be the number of buggy submissions (our definition of buggy: a submission is not buggy if and only if passes all test cases).
  - If your test cases can catch at least 75% of buggy submissions, then you receive 2 marks.
  - If your test cases can catch 50% buggy submissions, then you receive 1 mark.

4 SVN Instructions

First of all, you need to create a directory under version control:

```
svn mkdir --parents -m "Creating ADSA Assignment 3 folder" https://version-control.adelaide.edu.au/svn/aXXXXXXX/2017/s2/adsa/assignment3/
```

aXXXXXXX should be your student ID. The directory path needs to be exactly "2017/s2/adsa/assignmentK", where "K" is the assignment number. To check out a working copy, type

```
svn checkout https://version-control.adelaide.edu.au/svn/aXXXXXXX/2017/s2/adsa/assignment3/ adsa-17-s2-assignment3/
cd adsa-17-s2-assignment3
```

```
svn add main.cpp
svn add test.txt
```

Commit the files to SVN:

```
svn commit -m "Adding ADSA assignment 3 main.cpp test.txt"
```

SVN helps keeping track of file changes (over different commits). You should commit your work early and often.

5 Websubmission

You are asked to submit via the web interface `https://cs.adelaide.edu.au/services/websubmission/`. The submission steps should be self-explanatory. Simply choose the correct semester, course, and assignment. The websubmission system will automatically fetch the latest version of your work from your SVN repository (you may also choose to submit older versions). Once your work is submitted, the system will launch a script checking the format of your submission. Click "View Feedback" to view the results. Your mark will be calculated offline after the deadline. You are welcome to resubmit for as many times as you wish (before the deadline).

We will compile your code using `g++ -o main.out -std=c++11 -O2 -Wall main.cpp`. It is your responsibility to ensure that your code compiles **on the university system**.[1]

[1]g++ has too many versions, so being able to compile on your laptop does not guarantee that it compiles on the university system. You are encouraged to debug your code on a lab computer (or use SSH).