Fall 2022

CPSC2620 Assignment 1

Due: September 26 (Monday), 2022 11:55pm

(20 marks)

Please note that due to the potential COVID situation, the requirements of this assignment might be changed and also the compilation and submission procedure might be changed. You will be notified should a situation arise.

Notes:
(*) Please note that no collaboration is allowed for this assignment. Should any plagiarism be identified, all the involved students will get zero for the assignment and will be reported to the Chair of the department for further action. The marker has been asked to check your submissions carefully.
(*) Your submission will be marked on the correctness and readability (including comments) of your program.
(*) Please follow the guidelines to name your files (see below).
(*) Write a Makefile that is used to compile your program.
(*) Late submission policy: 1 day late: 5% off, 2 days late: 10% off, more than 2 days late: you will get 0 for the assignment.

In this assignment, you will write a program (**dictionary1.cc**) that a user can add a word, delete a word, search a word, list all words, etc., in a dictionary. The user can issue the following commands (all in capital letters).

ADD *aword* → Add *aword* into the dictionary. If *aword* already exists in the dictionary, print out "aword is already added.". Otherwise, add it to the end of the dictionary and print out "aword is added.".
DELETE *aword* → Delete *aword* from the dictionary. If *aword* does not exist in the dictionary, print out "aword is not found.". Otherwise, delete it from the dictionary and print out "aword is deleted.".
UPDATE *oldword newword* → Replace *oldword* with *newword* in the dictionary. If the *oldword* does not exist in the dictionary, print out "oldword is not found.". Otherwise, do the replacement and print out 'oldword is replaced by newword.'.
SEARCH *aword* → Search *aword* in the dictionary. If *aword* does not exist in the dictionary, print out "aword is not found.". Otherwise, print out "aword is found." and display the number of comparisons.
LIST → List and print out all the words in the current dictionary, one word one line. Print out the total number of words in the dictionary.
RLIST → List the print out all the words in the current dictionary, one word one line, in the reverse order. Print out the total number of words in the dictionary.
EXIT → Print out "bye bye" and exit to the terminal's prompt.

The following is a sample session of running your program, where % is the prompt from the terminal window, dictionary is the name of your program, and >>> is the prompt from your program. Please read it carefully and follow the format in your own program.

```
%dictionary
>>> ADD good
good is added
>>> ADD morning
morning is added.
>>> ADD morning
morning is already added.
>>> ADD evening
evening is added.
>>> UPDATE evening afternoon
evening is replaced by afternoon
…
…
…
>>> SEARCH morning
morning is found. 2 comparisons made.
>>> DELETE evening
evening is not found
>>> SEARCH night
night is not found.
>>> LIST
good
morning
…
There are 5 words in the dictionary.
>>> RLIST
…
morning
good
There are 5 words in the dictionary.
>>> DELETE aword
aword is not found.
>>> EXIT
bye bye
%
```

In order to accomplish our task, please follow and write the following help functions.

(*) string* myAdd(string *dictionary, int &wordCount, string aWord);
This function implements the command ADD as discussed above. In this function, if the aword is added to the dictionary, you need to resize the dictionary and return the new dictionary.

(*) **string *myDelete(string *dictionary, int &wordCount, string aWord);**
This function implements the command DELETE as discussed above. In this function, if aword is deleted from the dictionary, you need to resize the dictionary.

(*) **void mySearch(string dictionary[], int wordCount, string aWord);**
This function implements the command SEARCH as discussed above. In this function, you will call another function called linear_search which will be discussed below.

(*) **void myUpdate(string dictionary[], int wordCount, string oldWord, string newWord);**
This function implements the command UPDATE as discussed above. In this function, you will call another function called linear_search, which will be discussed below, and then replace the oldWord with the newWord, if the search is successful.

(*) **void myList(string dictionary[], int wordCount);**
This function implements the command LIST as discussed above. In this function, you list each word, one on a line, from the dictionary.

(*) void myRList(string dictionary[], int wordCount);
This function implements the command RLIST as discussed above. In this function, you list each word, one on a line, from the dictionary, in the reverse order.

(*) void myExit();
This function implements the command EXIT as discussed above.

(*) bool linearSearch(const string dictionary[], int wordCount, string aword, int &count);
This function returns true if aword is found in dictionary of wordCount words, and false, otherwise. Use the linear search algorithm. The count parameter should return the number of comparisons made to array elements.

(*) Write the **main** function which prompts the user to issue a command, processes the command, and then repeats. It can be easily seen that the main function is just a switch board. It analyzes the input string, figures out the command, and then calls the corresponding function. **You must implement the dictionary as a dynamic array**.

Final notes:
(*) Your source file should be named dictionary1.cc and your make file should be named Makefile. You need to submit them on the Moodle.
(*) Assume that all the input commands are valid. A user cannot input, for instance, “DEELTE aword”, “Add aword” or “ADD aword anotherword”.
(*) Assume that all words in the dictionary do not have spaces in them.
(*) Function linearSearch is very important. You need to call it from several other functions. Also the parameters in linearSearch can be used in various ways.