Project 2. A Pronunciation Dictionary App

Introduction

In this project you are going to implement a linguistic application that uses a pronunciation dictionary for finding words with similar pronunciation.

Example. You enter a word, and it reports similar-sounding words:

```
> donut
```

```
Pronunciation    : D OW1 N AH2 T

Identical        : DOUGHNUT
Replace phoneme  : DONAT DONATE
Add phoneme      : DONUTS DONUTS' DOUGHNUTS
Remove phoneme   : DON'T
```

We are going to use The CMU Pronouncing Dictionary as our reference. It is available as a simply formatted plain text file, a direct link to it is: cmudict.0.7a or the local copy here: cmudict.0.7a

An excerpt from it is shown below:

```
PROGRAM  P R OW1 G R AE2 M
PROGRAM'S  P R OW1 G R AE2 M Z
PROGRAMME  P R OW1 G R AE2 M
PROGRAMMER  P R OW1 G R AE2 M ER0
PROGRAMMERS  P R OW1 G R AE2 M ER0 Z
PROGRAMS  P R OW1 G R AE2 M Z
PROGRAMS'  P R OW1 G R AE2 M Z
PROGRESS  P R AA1 G R EH2 S
PROGRESS(1)  P R AH0 G R EH1 S
PROGRESS(2)  P R OW0 G R EH1 S
PROGRESSED  P R AH0 G R EH1 S T
PROGRESSES  P R AA1 G R EH2 S AH0 Z
PUSH-UP  P UH1 SH AH2 P
PUSH-UPS  P UH1 SH AH2 P S
```

In linguistics, a phoneme is a perceptually distinct units of sound that distinguishes one word from another, for example p, b, d, and t in the English words “pad”, “pat”, “bad”, and “bat”.

Each line of the dictionary file contains a **word** followed by the list of its **phonemes** (`P` `R` `OW1` `G` `R` `AE2` `M`). Vowel phonemes, such as `OW` or `AE` , end with an additional digit `0` , `1` , or `2` , indicating the type of stress on that vowel (no stress, primary stress, secondary stress). If a word has multiple pronunciations, such alternatives are labeled with `(1)`, `(2)`, `(3)`, and so on (see the word `PROGRESS` in the example above). Comment lines start with triple semicolons (these lines can be ignored). For more information about the dictionary file formatting, read its web page referenced above.

For this project, to make the task easier, your program should **ignore all words that contain non-alphabetic characters**, and also **ignore all alternative pronunciations**, The only non-letter character that is **allowed** in a word is **apostrophe** `'`.

So, your program should ignore entries like:

```
PROGRESS(1)  P R AH0 G R EH1 S     < ignore
PROGRESS(2)  P R OW0 G R EH1 S     < ignore
PUSH-UP  P UH1 SH AH2 P            < ignore
PUSH-UPS  P UH1 SH AH2 P S         < ignore
%PERCENT  P ER0 S EH1 N T          < ignore
&AMPERSAND  AE1 M P ER0 S AE2 N D  < ignore
```

However, the following entries are considered good:

```
PROGRAM  P R OW1 G R AE2 M         < good
PROGRAM'S  P R OW1 G R AE2 M Z     < good
PROGRAMS'  P R OW1 G R AE2 M Z     < good
'BOUT  B AW1 T                     < good
```

Programming Task

Write a program `pronounce.cpp` that

- Lets the user input a word (let’s call the input word ***W***).
- If the word is not found in the dictionary, print “**Not found**”. Otherwise, report:

- `Pronunciation :` the pronunciation of the word *W* (as given in the dictionary),
- `Identical :` other words from the dictionary with **the same** pronunciation as *W*,
- `Replace phoneme :` words that can be obtained from *W* by **replacing** one phoneme.
- `Add phoneme :` words that can be obtained from *W* by **adding** one phoneme,
- `Remove phoneme :` words that can be obtained from *W* by **removing** one phoneme,

When listing words, include *all words from the dictionary* that meet the criteria, the order of listed words should be the same as they appear in the dictionary.

Your program should expect that the dictionary file `cmudict.0.7a` is located in the current working directory.

User input should be case-insensitive (accepting `donut`, `DONUT`, `DOnUt`, etc.)

Please, don't make complex user interface that allows multiple queries. The program should just ask for one word, report the answer, and exit. See examples below.

You are allowed to use only the constructs of the language that were mentioned in lecture slides and covered in class. For strings, you can use only the operations mentioned in class.

Examples:

```
> accord

Pronunciation    : AH0 K AO1 R D

Identical        : ACORD
Replace phoneme  : ABOARD ADORED AFFORD AWARD SCORED
Add phoneme      : ACCORD'S ACCORDS MCCORD RECORD
Remove phoneme   : CHORD CORD
```

```
> Ackerman

Pronunciation    : AE1 K ER0 M AH0 N

Identical        : ACKERMANN AKERMAN AKKERMAN
Replace phoneme  : ACKERSON ADERMAN AKERSON AMERMAN AMMERMAN
ANGERMAN ATTERMANN AUKERMAN ECKERMAN OCKERMAN
Add phoneme      :
Remove phoneme   : ACKMAN



> DRAFT

Pronunciation    : D R AE1 F T

Identical        : DRAUGHT
Replace phoneme  : CRAFT DRIFT GRAFT KRAFFT KRAFT
Add phoneme      : DRAFT'S DRAFTEE DRAFTER DRAFTS DRAFTY DRAUGHTS
Remove phoneme   : DAFT RAFT



> colonel's

Pronunciation    : K ER1 N AH0 L Z

Identical        : COLONELS KERNELS
Replace phoneme  : CANALES JOURNAL'S JOURNALS KENNELS
Add phoneme      :
```

```
Remove phoneme   : COLONEL KERNEL



> FLOWERS'

Pronunciation    : F L AW1 ER0 Z

Identical        : FLOURS FLOWERS
Replace phoneme  : CLOWERS FLIERS FLOWERED FLOWERY FLUOR'S FLYERS
Add phoneme      :
Remove phoneme   : FLOUR FLOWER FOWERS



> Gorilla

Pronunciation    : G ER0 IH1 L AH0

Identical        : GUERILLA GUERRILLA
Replace phoneme  : CHURILLA GUILLA KURILLA
Add phoneme      : GORILLAS GUERILLAS GUERRILLAS GUERRILLAS'
Remove phoneme   :



> aLiGnEd

Pronunciation    : AH0 L AY1 N D

Identical        :
Replace phoneme  : AFFINED ALIGNS ALINES ASSIGNED BLIND
Add phoneme      : MALIGNED UNLINED
Remove phoneme   : ALIGN ALINE ALLIED LINED
```

```
> allusion

Pronunciation    : AH0 L UW1 ZH AH0 N

Identical        :
Replace phoneme  : ALEUTIAN ILLUSION
Add phoneme      : ALLUSIONS COLLUSION OCCLUSION
Remove phoneme   :



> design

Pronunciation    : D IH0 Z AY1 N

Identical        :
Replace phoneme  : DEFINE DESIRE DIVINE RESIGN
Add phoneme      : DESIGNED DESIGNER DESIGNS
Remove phoneme   :



> drafty

Pronunciation    : D R AE1 F T IY0

Identical        :
Replace phoneme  : CRAFTY DRAFT'S DRAFTEE DRAFTER DRAFTS DRAUGHTS
Add phoneme      :
Remove phoneme   : DRAFT DRAUGHT RAFFETY
```

```
> defer

Pronunciation    : D IH0 F ER1

Identical        :
Replace phoneme  : DEFOE DEFY DEMUR DETER DUFUR
Add phoneme      : DEFERRED DEFERS DEFLEUR
Remove phoneme   :



> birb

Not found



> PROGRESS(1)

Not found



> ...ELLIPSIS

Not found
```

Hint

For dividing a string into words, we can give you a helper function, which receives a string argument `s`, and splits it **into two strings on the very first space** it finds within the string `s`:

Example:

`"Fortune favors the bold"` → `"Fortune"` and `"favors the bold"`.

```
void splitOnSpace(string s, string & before, string & after) {
  // reset strings
    before = "";
    after = "";
  // accumulate before space
    int i = 0;
    while (i < s.size() && not isspace(s[i])) {
        before = before + s[i];
        i++;
    }
  // skip the space
    i++;
    // accumulate after space
    while (i < s.size()) {
        after = after + s[i];
        i++;
    }
}
```

The two arguments passed by reference, `before` and `after`, will contain the two resulting halves of the string: before and after the space.

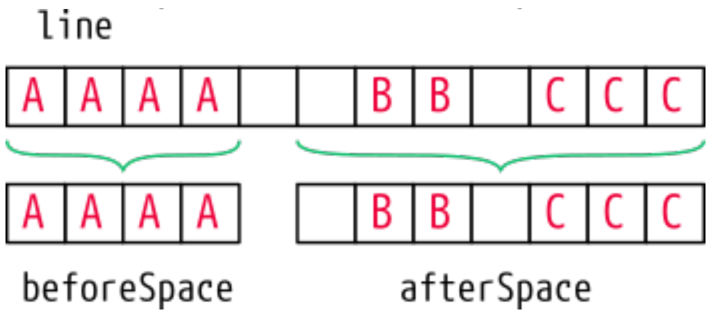
Example:

```
string line = "AAAA  BB CCC";
string beforeSpace;
string afterSpace;
splitOnSpace(line, beforeSpace, afterSpace);
```

After the function call, the second and the third argument variables got updated with the following values:

```
beforeSpace == "AAAA" // contains everything before the first space
afterSpace  == " BB CCC" // contains everything after it
```

Notice that since there were **two spaces** between `"AAAA"` and `"BB"` in the input string, after splitting on the first space character, the second space is preserved in the beginning of the variable `afterSpace` (also see the diagram below).



If you wan to cut off that leading space, you can either write your own function for that, or actually, you can call the same splitting function again on `afterSpace`.

Phase I

User enters a word, and the program only reports pronunciation as a list of phonemes. Example:

```
> donut

Pronunciation    : D OW1 N AH2 T
```

Phase II

User enters a word — the program reports pronunciation as a list of phonemes AND a list of words that sound the same (is made up of exactly the same sequence of phonemes). Example:

```
> FLOWERS'

Pronunciation    : F L AW1 ER0 Z

Identical        : FLOURS FLOWERS
```

Phase III

In addition to the functionality of phases I and II, print a list of words that can be obtained by **replacing** just one phoneme — words made up of exactly the same sequence of phonemes with just one of them different. Example:

```
> accord

Pronunciation    : AH0 K AO1 R D

Identical        : ACORD
Replace phoneme  : ABOARD ADORED AFFORD AWARD SCORED
```

Phase IV

In addition to the functionality of phases I, II and III, print a list of words that can be obtained by **adding** just one phoneme — words made up of exactly the same sequence of phonemes with just one additional one — anywhere in the original sequence of phonemes. Example:

```
> DRAFT

Pronunciation    : D R AE1 F T

Identical        : DRAUGHT
Replace phoneme  : CRAFT DRIFT GRAFT KRAFFT KRAFT
Add phoneme      : DRAFT'S DRAFTEE DRAFTER DRAFTS DRAFTY DRAUGHTS
```

Also, start reviewing all of your code and making necessary changes in order to meaningfully organize it into functions. “Check Identical”, “Check Replace Phoneme”, and “Check Add Phoneme” should ideally be independent functions, called on each dictionary entry, as the dictionary is traversed after the original pronounciation has been determined.

Phase V

In addition to the functionality of phases I, II, III and IV, print a list of words that can be obtained by **removing** just one phoneme — words made up of exactly the same sequence of phonemes with just one missing — anywhere in the original sequence of phonemes. Example:

```
> colonel's

Pronunciation    : K ER1 N AH0 L Z

Identical        : COLONELS KERNELS
Replace phoneme  : CANALES JOURNAL'S JOURNALS KENNELS
Add phoneme      :
Remove phoneme   : COLONEL KERNEL
```

Please make sure that your code is now meaningfully organize into functions. “Check Identical”, “Check Replace Phoneme”, “Check Add Phoneme” and “Check Remove Phoneme” should be independent functions, called on each dictionary entry, as the dictionary is traversed after the original pronounciation has been determined.

How to submit your programs

Everything should be submitted through Gradescope.

Upload only the source code (.cpp) files, not the compiled executables.
Each program should start with a comment that contains your name and a short program description, for example:

```
/*
Author: your name
Course: CSCI-135
Instructor: their name
Assignment: title, e.g., Lab1A

Here, briefly, at least in one or a few sentences
describe what the program does.
*/
```