C++ Assessment 1, 2016-7

This assessment is worth 40% of the total marks available for C++ programming. The submission date is Wednesday 7th December 2016.

Constructing a spell-checker algorithm

Word processors and other text editing applications, such as email and text applications, typically spell-check words as they are typed in. The task for this assessment is to write a small application that detects misspellings in typed-in words and prompts the client for corrections to words that it does not recognize. A word is recognized if it is in a dictionary held by the application.

Clients enter text one word at a time. When the client enters a word the application must

i) Check to see if it in the dictionary. If it is then the application prompts the client to enter the next word, or to end.

If the word is not in the dictionary then the application

ii) Computes a distance between it and every word in the dictionary.

iii) Prints out some of the closer words in the dictionary as suggested corrections.

iv) Asks the client to do one of three things:

- a) Choose one of the suggested words as a correction,
- b) Add the entered word to the dictionary,
- c) Re-enter the word.

If the client chooses:

(a) then proceed to the next word.

(b) add the word to the dictionary and then proceed to the next word,

(c) return to step (ii).

A small dictionary of 200 common English words is given in small_dictionary.cpp as a `std::vector` of `std::string`s. This can be used as a starting point. Ideally the updated dictionary constructed by the application would be stored in a file so that additions to it are not lost. However this is not required for this assessment. Nor is it necessary to implement tree-based search algorithms or similar refinements for step (i) nor refinements for step (ii) so that the distance is not computed for every single word in the dictionary. Code should be as computationally efficient as possible, but only within the scope of the C++ module.

Distances between words must be computed using the Damerau-Levenshtein distance measure.

The Damerau-Levenshtein distance

One way to measure how far apart two strings are is to compute the Damerau-Levenshtein distance. This is the minimum number of edits needed to change one string into the other. The allowable edit operations are:

i) deleting a character from a string,

ii) adding a character a string,

iii) substituting one character for another in a string,

iv) transposing two successive characters in a string.

Let a and b be two strings whose Damerau-Levenshtein distance is to be computed. Write a_i for the ith character of a (1-based) and a_0 and b_0 are defined to be the null character The Damerau-Levenshtein distance between a and b is given by $d_{a,b}(|a|, |b|)$ where $d_{a,b}(|a|, |b|)$ is defined recursively. For $0 \le i \le |a|$, $0 \le j \le |b|$, set

$$d_{a,b}(i,j) = \begin{cases} max(i,j), & i = 0 \text{ or } j = 0, \\ min\begin{Bmatrix} d_{a,b}(i-1,j)+1, \\ d_{a,b}(i,j-1)+1, \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)}, \\ d_{a,b}(i-2,j-2)+1 \end{Bmatrix}, & i > 1 \text{ and } j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_{j,} \\ min\begin{Bmatrix} d_{a,b}(i-1,j)+1, \\ d_{a,b}(i,j-1)+1, \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{Bmatrix}, & i = 1 \text{ or } j = 1 \text{ or } a_i \neq b_{j-1} \text{ or } a_{i-1} \neq b_j. \end{cases}$$

where $1_{(a_i \neq b_j)}$ is the indicator function with value 1 if $a_i \neq b_j$, value 0 otherwise. (See the Damerau-Levenshtein distance Wikipedia page and references therein.)

You should construct a clear user interface and write code in a clear and maintainable style. The code you submit must to be able to run without addition or modification with the DevCpp installation on the machines in the teaching area.

The report
Your report must include a clear description of your application and an assessment of its performance. You should include screen shots illustrating the application responding to user input showing a range of possible responses. (See the general requirements.) You will be assessed on

a) The quality of your code in terms of readability, maintainability, clarity, sophistication, generalizability, and general presentation.
b) The degree to which your code has the required functionality.
c) Your assessment of the efficacy of your implementation.
d) The presentational quality of the assessment as a whole.

Group work is not permitted
You are encouraged to discuss the assessment with other students but the code you submit must be yours alone. Clear similarities in code, in the write-up or in the results, will be taken as evidence of group work (for instance if identical or very similar screen shots as given, or if code is clearly shared.)

Code closely modelled on that from other sources is not permitted
Code to perform components of this exercise is to be found in various places and can, for instance, be downloadable from the web. Use of code taken from, or cosmetically altered from, such sources is not permitted. You must devise your code from scratch.

Specifically *excluded* from this prohibition is code I have declared that you may use. In particular you may use the library code I have made available, without attribution.

Nick Webber