MSc (Computing Science) 2016-2017
C/C++ Laboratory Examination

Imperial College London

Monday 9 January 2017, 14h00 – 16h10

Eye halve a spelling chequer
It came with my pea sea
It plainly marques four my revue
Miss steaks eye kin knot sea.

Eye strike a quay and type a word
And weight four it two say
Weather eye am wrong oar write
It shows me strait a weigh.

As soon as a mist ache is maid
It nose bee fore two long
And eye can put the error rite
It's rare lea ever wrong.

Eye have run this poem threw it
I am shore your pleased two no
It's letter perfect awl the weigh
My chequer tolled me sew.

Anonymous

- ☞ You are advised to use the first 10 minutes for reading time.
- ☞ Log into the Lexis exam system using your DoC login as both your login and as your password (**do not use your usual password**).
- ☞ You must add to the pre-supplied header file **spell.h**, pre-supplied implementation file **spell.cpp** and must create a **makefile** according to the specifications overleaf.
- ☞ You will find source files **spell.cpp**, **spell.h** and **main.cpp**, and data file **words.dat** in your Lexis home directory (**/exam**). If one of these files is missing alert the invigilators.
- ☞ **Save your work regularly.**
- ☞ Please log out once the exam has finished. No further action needs to be taken to submit your files.
- ☞ No communication with any other student or with any other computer is permitted.
- ☞ You are not allowed to leave the lab during the first 15 minutes or the last 10 minutes.
- ☞ **This question paper consists of 5 pages.**

Problem Description

Spell checking is the process of flagging words in a document that may not be spelled correctly – vital functionality that is taken for granted by most users of modern PCs. Your task today is to write some C++ routines to support spell checking.

A spell checker depends crucially on having a way to measure the degree of difference between the word someone actually typed and the word they were meaning to type – the so-called **edit distance**. More precisely, the edit distance between two strings a and b is defined as the number of character insertions, deletions, replacements and transpositions (swaps of adjacent characters) needed to transform a into b.

The edit distance between some string a (of length $|a|$, and made up of characters $a_0, a_1, \ldots, a_{|a|-1}$) and some string b (of length $|b|$, and made up of characters $b_0, b_1, \ldots, b_{|b|-1}$) can be computed with the help of the so-called **Damerau–Levenshtein distance function** as follows.

For some integer indices i and j, define recursive function $d_{a,b}(i,j)$ as:

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

where $\mathbb{1}_{a_{i-1}\neq b_{j-1}}$ is the indicator function equal to 0 when $a_{i-1} = b_{j-1}$ and equal to 1 otherwise.

Then the edit distance between a and b is given by the function value $d_{a,b}(|a|,|b|)$.

Note each recursive call in the definition of $d_{a,b}(i,j)$ above matches one possible edit of a:

- $d_{a,b}(i-1,j)+1$ corresponds to a character deletion.
- $d_{a,b}(i,j-1)+1$ corresponds to an character insertion.
- $d_{a,b}(i-1,j-1)+\mathbb{1}_{a_{i-1}\neq b_{j-1}}$ corresponds to a character replacement, or to a perfect match.
- $d_{a,b}(i-2,j-2)+1$ corresponds to the transposition of two successive characters.

A simple way for a spell checker to suggest corrected spellings for a single word is as follows[1]. Given a word w, and a dictionary of known words D,

1. If w is in D, return w.
2. If there are one or more words in D at an edit distance of one from w, return the one that occurs most frequently in typical English text[2].
3. If there are one or more words in D at an edit distance of two from w, return the one that occurs most frequently in typical English text.
4. Otherwise, give up and return the original word, even though it is not known.

[1]Inspired by Peter Norvig (Google's Director of Research).
[2]If there are several candidate words with the same highest frequency, the tie may be broken by choosing one arbitrarily.

Pre-supplied functions and files

You are supplied with a main program in **main.cpp**, and with a **words.dat** data file representing a dictionary of English words and their corresponding frequencies, i.e. the number of times each word was observed in a corpus made up of several works of literature.

You can use the UNIX command **less** to inspect the **words.dat** file, which you should find contains data in two columns like this:

```
20096 a
 1058 A
    1 Aachen
    1 Aah
    1 Aaliyah
    1 aardvark
    1 aardvarks
    6 Aaron
    2 ab
    1 abaci
    4 aback
    2 abacus
    ...
```

The first column indicates the number of times the word in the second column appeared in the corpus – so we can observe "a" is a very common word, appearing 20 096 times in the corpus; while "aardvark" is rather less common, appearing just once.

You are also supplied with the beginnings of the header file **spell.h** (for your function prototypes) and the beginnings of the implementation file **spell.cpp** (for your function definitions).

Specific Tasks

1. Write an integer-valued function `frequency(target)` which takes as its parameter a target word and which returns the number of times the target word appears in the corpus (by looking this up in **words.dat**). If the word cannot be found in **words.dat** then the function should return 0.

   For example, the code:

   ```
   cout << "The frequency of the word 'a' in the corpus is "
        << frequency("a") << endl;
   ```

   should display the output

   ```
   The frequency of the word 'a' in the corpus is 20096
   ```

   while the code:

   ```
   cout << "The frequency of the word 'nonexistential' in the corpus is "
        << frequency("nonexistential") << endl;
   ```

   should display the output

   ```
   The frequency of the word 'nonexistential' in the corpus is 0
   ```

2. Write an integer-valued function `edit_distance(a,b)` which returns the edit distance between read-only input character strings a and b.

   For example, the code:

   ```
     cout << "The edit distance between 'an' and 'na' is "
          << edit_distance("an","na") << endl;
   ```

   should display the output

   ```
   The edit distance between 'an' and 'na' is 1
   ```

   since “an” may be transformed into “na” by a single transposition of (adjacent) characters.

   Likewise, the code:

   ```
     cout << "The edit distance between 'korrectud' and 'corrected' is "
          << edit_distance("korrectud","corrected") << endl;
   ```

   should display the output

   ```
   The edit distance between 'korrectud' and 'corrected' is 2
   ```

   Since the replacement of two characters ('k' and 'u') sufficies to transform the first word into the second.

3. Write a Boolean function `spell_correct(word, fixed)` which suggests a spell checked variant of a word. Here `word` is a read-only input parameter specifying the word to be spell checked, while `fixed` is an output parameter giving the suggested correction. The return value of the function should be `true` if `fixed` differs from `word` and `false` otherwise.

   For example, the code

   ```
     bool corrected = spell_correct("korrectud", fixed);
     cout << "The corrected spelling of 'korrectud' is '"<< fixed << "'" << endl;
     cout << "The spell checker was " << (corrected ? "" : "NOT ")
          << "needed."<< endl;
   ```

   should result in the output

   ```
   The corrected spelling of 'korrectud' is 'corrected'
   The spell checker was needed.
   ```

   with Boolean variable `corrected` set to `true`.

(The three parts carry, resp., 30%, 40% and 30% of the marks)

What to hand in

Place your function implementations in the file **spell.cpp** and corresponding function declarations in the file **spell.h**. Use the file **main.cpp** to test your functions. Create a **makefile** which will compile your submission into an executable file entitled **spell**.

(P.T.O. for hints)

Hints

1. You will save time if you begin by studying the main program in **main.cpp**, and the given data file **words.dat**.
2. Feel free to define any of your own helper functions which would help to make your code more elegant. This will be particularly useful when answering Question 2.
3. Try to attempt all questions. If you cannot get one of the questions to work, try the next one.
4. You are not explicitly required to use recursion in your answer to any of the questions. However, you may make use of recursion wherever you feel it would make your solution more elegant (especially when answering Question 2).
5. You may find your first implementation of Question 2 not quite efficient enough for you to obtain a solution to Question 3 in reasonable time. You may be able to address this problem using up to two additional default parameters in your solution to Question 2 to restrict the maximum edit distance explored.

Bonus Challenge

Can you write a function `edit_distance_bonus(a,b)`, which behaves similarly to the function `edit_distance(a,b)` of Question 2, but which (a) makes use of pointer arithmetic, (b) does not use any helper functions and (c) is directly recursive? You can add up to two default parameters, provided these are required for efficiency reasons (i.e. they cannot be used for i and j).