MTH 3300
Spring 2017
E. Fink
April 3, 2017

Homework 8

Directions: Write the following 2 programs, and submit your source code to me via Blackboard, using the template files I have provided. You should send only the source code, in cpp format; do NOT send your .exe files. READ THE INSTRUCTIONS on how to submit your work in the Course Documents section of Blackboard.

This programming assignment is meant to give more practice on functions and arrays.

---

1) translate.cpp

Write a bad English-to-Spanish translator. It will read a file word-by-word, and then use supplied files to replace English words with their "Spanish" equivalents. It is "bad" for two reasons – the files only cover the 1000 most common English words; and the provided translations are very literal and rigid, and so the translations generally will be grammatically incorrect or use different meanings of words. The upside is that it is easy to build.

The program should work as follows: consider the files `english.txt` and `spanish.txt`. The first is a list of 1000 common English words, each on a different line; the second is a list of corresponding translations. So, for example, the third line in the English file is `about`, while the third line in `spanish.txt` is `acerca de` – because `acerca de` is a Spanish translation of `about`.

Your program should read each of these files into arrays of length 1000 – one should get the 1000 English words, the other with the corresponding Spanish translations in the corresponding order. (Note that the Spanish translations may contain more than one word, so `getline()` will be your friend here.)

Then, write a function called `translate`. This function should an English word as input, and return either the word's Spanish translation, or the string `"???"` if the word is not one of the English words in the dictionary. Your function may take more inputs also.

Then, test your function! Remember the test code I gave you for the twin prime problem? Write your own test code here, to make sure that `translate` is working the way you expect it to.

Now, your `main` function should ask the user to enter the names of an input file to read, and an output file to write the input's translation to. The program should open the input file, and translate each word, writing the translation to the specified output file.

To simplify your task slightly, you may assume that

- every word is all lowercase;
- every line ends with the characters `"!!!"`;
- aside from that, there is no punctuation.

See the example input (`english_doc.txt`) and corresponding output (`trans.txt`) files that I have.

Specifications: your program must

- write a function `translate` that takes in an English word (possibly with other arguments), and returns either the corresponding Spanish word or `"???"`.
- ask the user to enter the names of files for input and output.
- read the input file word-by-word, and then print the translations of each word into the output file. Remember that the word `"!!!"` should become a newline in the translation.

---

2) yield.cpp

Situation: you are given seven constants: B; c_1, c_2, and c_3; and T_1, T_2 and T_3. Given those constants, you would like to solve the equation

$$B = c_1 e^{-xT_1} + c_2 e^{-xT_2} + c_3 e^{-xT_3}$$

for x.

Several questions arise:

1. Why would we ever want to solve such an equation? Answer: because secretly, we're calculating the yield of a bond with three payments! See the very bottom of this sheet if you're interested.
2. Can we solve this algebraically? Answer: probably not.
3. Can we use computer magic to solve this numerically? Answer: well, we wouldn't be talking about this if we couldn't!

In this problem, we solve this equation for y using Newton's method. First we'll talk about Newton's method; then I'll give you the problem description; and then I'll say some words about bonds.

Newton's method is a method to approximate solutions to an equation $f(x) = 0$ very quickly. It works by starting with a (more-or-less random) guess x_0 for a solution, and then coming up with better and better approximations x_1, x_2, x_3, etc., using the following process:

$$\begin{aligned} x_1 &= x_0 - f(x_0)/f'(x_0) \\ x_2 &= x_1 - f(x_1)/f'(x_1) \\ x_3 &= x_2 - f(x_2)/f'(x_2) \end{aligned}$$

and so on and so forth, with $x_{n+1} = x_n - f(x_n)/f'(x_n)$ in general. For reasons that are best explained by a textbook (or me, in person), x_1 will usually be close to a solution, and x_2 will be closer, and x_3 closer still, etc.

Time for an example: Let's try to solve $x^2 - 4x + 1 = 0$. Here, $f(x) = x^2 - 4x + 1$. We start with a guess of $x_0 = 1$.

Then $x_0 = 1$; $f(x_0) = -2$; $f'(x_0) = -2$; and so

$$x_1 = 1 - (-2)/(-2) = 0.$$

Then $x_1 = 0$; $f(x_1) = 1$; $f'(x_1) = -4$; and so

$$x_2 = 0 - (1)/(-4) = 0.25.$$

Then:

$$x_3 = 0.25 - f(0.25)/f'(0.25) = 0.267857142 \text{ (approximately).}$$

Then:

$$x_4 = 0.267857142 - f(0.267857142)/f'(0.267857142) = 0.267949190 \text{ (approximately).}$$

And so on. The actual value of one solution to 9 places is 0.267949192, so we were already at 8 decimal precision after only four iterations – not bad.

Now, for your task. Your program will ask the user to enter in B, c_1, c_2, c_3, T_1, T_2, T_3, and will then solve the equation $f(x) = 0$, where

$$f(x) = c_1 e^{-xT_1} + c_2 e^{-xT_2} + c_3 e^{-xT_3} - B. \tag{1}$$

(To be very specific: x is the only variable on the right side of the above equation.) Your program should ask for a guess of x_0 also, and then do 100 iterations of Newton's method. You should report the value of x_{100}.

Hints: you can use the same variable for x_0, x_1, x_2, etc., because as soon as you know the value of, say, x_{12}, you don't need to know the value of x_{11} anymore.

Also, $f'(x) = -T_1 c_1 e^{-xT_1} - T_2 c_2 e^{-xT_2} - T_3 c_3 e^{-xT_3}$.

Check values: if $B = 100$; $c_1 = 5$, $c_2 = 5$, $c_3 = 105$; and $T_1 = 1$, $T_2 = 2$, and $T_3 = 3$, then the output yield should be approximately 0.0487902, or $\approx 4.88\%$.

Specifications: your program must

- ask the user to enter in values for B, c_1, c_2, c_3, T_1, T_2, T_3, and the initial guess x_0.
- Use 100 iterations of Newton's method to estimate the solution to equation 1: that is, print out the value of x_{100}.
- Use *user-defined functions* to calculate values of the functions $f(x)$ and $f'(x)$.
- Check that x_{100} is a pretty good solution: if $-0.0000001 < f(x_{100}) < 0.0000001$, print out a message saying that we have a good approximate solution.

Motivation (you may ignore if you like): So, what is it that we're really doing here? As I mentioned, we're calculating the **yield** of a bond.

A *bond* is a financial instrument that you can buy at a certain time, that will entitle you to fixed payments at certain later dates. For example: you might buy a bond for \$100 today, that will entitle you to a payment of \$5 a year from today, \$5 two years from today, and \$105 three years from today. (I.e., $B = 100$; $c_1 = 5$, $c_2 = 5$, $c_3 = 105$; and $T_1 = 1$, $T_2 = 2$, and $T_3 = 3$.) Notice how you get paid back \$115 in total: you better get more than you put in – otherwise, why would you let some yahoo hold on to your money for 3 years?

You may have a variety of investment opportunities that you would like to compare, and it would be good to be able to measure which are more lucrative. One way to do so is to look at the *yield*, which is the (continuously compounded) interest rate such that if someone would offer it to you, you could invest the same as you paid for the bond, and get the same payouts at the same times. The yield for the bond described above would be approximately 4.88%: this means, essentially, that you could invest \$100 at 4.88% continuously compounded, and be able to spend \$5 of the resulting cash at year 1, \$5 more at year 2, and still have \$105 at year 3.

The equation we're using actually uses *discounted future values* to compute the yield. Really, there's lots more to talk about here; if you're interested in more detail, talk to me.