CIS*2500 W20 - Assignment 3

Automating the Decoding of a Caesar Cipher

Question 1 (5 marks)

To automate the determining of the shift used in the Caesar encoding, you need to determine the frequency in the text of each letter in the alphabet (again see the website http://en.wikipedia.org/wiki/Caesar_cipher)

Write the following functions:

A function with the signature `int letter_count(char *)`
that counts the number of letters in the string (characters between A/a to Z/z)

A function with the signature `int * frequency_table(char *)` that returns a 1d array
- each element of the array holds the numbers of times each letter (upper or lower case) occurs in the string that is passed in as an argument
- i.e. the first index holds the count for 'a' / 'A', the second index hold the count for 'b' / 'B', etc.

To test these functions, write a program that take in text (either encoded or not) from stdin or a filename provided with the -F command line argument (as used in Lab 3) and in the mainline, prints the frequency table (a table of character counts) to stdout with details as follows:

Before the table, on its own line, print 2 numbers (appropriately labeled):
- the number of letters in the text
- the total number of characters in the text

The table itself should be printed in two columns:
- The first column should print out the letters
- The second column should hold the corresponding letter counts for each letter

The columns should be separated by tab characters (look it up in an ASCII table).
There should be a header row describing each column.

This program should be called **frequency_table.c**

Question 2 (25 marks)

Here are the letter frequencies that are commonly found in the English language:

a	8.167%	**b**	1.492%	**c**	2.782%	**d**	4.253%	**e**	12.702%	**f**	2.228%
g	2.015%	**h**	6.094%	**i**	6.966%	**j**	0.153%	**k**	0.772%	**l**	4.025%
m	2.406%	**n**	6.749%	**o**	7.707%	**p**	1.929%	**q**	0.095%	**r**	5.987%
s	6.327%	**t**	9.056%	**u**	2.758%	**v**	0.978%	**w**	2.360%	**x**	0.150%
y	1.974%	**z**	0.074%								

Let ENGLISH_FREQ[*i*] (which you can also denote EF[i])
be an array that stores the above table, where *i* = 0 to 25.

For example, `EF[offset('b')] == 0.01492` (as does `EF[offset('B')]`.

Here `offset(c)` shifts a letter character down so it is the offset from `'a'`or `'A'`,
i.e. the `offset('a') = 0` and `offset('d') = 3`

Note: the table should be a constant throughout your program.

Compute the frequencies in the input file (see question 1),
denoting the frequency of a character in that file as text_freq[*c*].

Now use the following formula, called the Chi-Squared formula (used in the Chi-Squared test - see Pearson's chi-square test for details) for a given shift "guess".

$$\texttt{chi\_sq(shift)} = \sum_{\substack{\text{for}\\ \text{all c}}} \frac{(n * \texttt{EF[offset(c)]} - \texttt{text\_freq[offset(encode(c, shift))]})^2}{\texttt{n} * \texttt{n} * \texttt{EF(offset(c))}}$$

Where `n` is the number of letters in the text (see question 1), and c is a character in the alphabet. Consequently, "for all c" means "for each character in the alphabet", not "for each character encountered in your text". Finally, `encode(c, shift)` is the Caesar cipher encode function you created for Lab 3

If the guess for the shift was right, `chi_sq(good_guess)` will be much smaller than when any other shift is used. So, examine all possible shifts, from 0 to 25, and determine which shift produces the smallest result when used as an argument for chi_sq; that shift should decode the message.

If the smallest `chi_sq(shift)` is too big (>= 0.5) assume the text isn't English and return a shift of 0.
Note: your input text must be large enough for this property to hold (at least 200 characters).

The above should be written as various support functions, together which are to be used to create the functions

- `int encode_shift(char *)` returns the minimum chi-sq shift found above
- `int to_decode(int shift)` takes a shift used to encode text and produces the shift that will decode it

To test this function, write a program named **decode.c**

Your program will take command line arguments

-F	The name of the input file. If missing use stdin (to end the input, press ^d on the keyboard, which inserts the EOF character).
-O	The name of the output file after decoding. If missing use stdout, where the decoded text should be the last thing printed.
-n	Suppresses the printing of the decoded file to stdout. Useful in combination with -s or –S. If -O is included as a command line argument -n does nothing.
-s	Computes the Caesar shift value used to decode the message, and prints it to stdout (e.g. shift = 4). The shift value is printed on its own line before anything else is printed. For example, if -O and -n are not used and the decoded file is printed to stdout, print the shift first and include a blank line before printing the decoded message/file.
-S	Computes the original Caesar shift value used to encode the message, and prints it to stdout (e.g. shift = 4). The shift value is printed on its own line before anything else is printed. For example, if -O and -n are not used and the decoded file is printed to stdout, print the shift first and include a blank line before printing the decoded message/file.
-t	Computes the character/letter count summary and frequency table and prints them to stdout using the same format at question 1. All other options that print to the stdout will print before the frequency table with the exception of the decoded message if -O is not used. If the decoded text is being printed to stdout, print the table first and include a blank line before printing the decoded message/file.
-x	Computes the chi squared value for all shifts, printing them out along with their corresponding shift values

Allow for command line arguments other than -F and -O to be used together with the other arguments in one command in any order. The order of the arguments need not affect the order of the printout.

For example

```
prompt: decode –stx –F myfile.txt –O decodedfile.txt
```

will read from myfile.txt, print to decodedfile.txt and print the shift value, character count summary and frequency table, as well as the chi squared values to stdout

Any error messages or warnings should be printed to stderr.

Question 3 (10 marks)

There is a file to be read that contains a series of records written in binary mode.
A record has the following structure:

- A string, up to 24 characters long (including ending null character), possibly encrypted
- 24 double values
- A string, up to 144 characters long (including ending null character), possibly encrypted
- 12 integer values

If a text field is encrypted, it is using a Caesar cipher encryption with the same shift as used in encoding a companion file stored in the same directory, which hold English text that contains at least 100 characters.

Write a program named **copyrecords.c** which will copy the records across to a new file (in binary format), possibly in reverse order, with the fields decoded using the shift discovered using the accompanying text file.

Briefly explain your approach to reversing the records in the readme file. Note: assume there are too many records to reverse them in memory (i.e. you will need use random access file commands).

Your program will take command line arguments

-F	The name of the input file. If missing use stdin and, if present, ignore the -r option (i.e. write the file in order) ← *modified* Note: to end the input, press ^d on the keyboard, which inserts the EOF character
-O	The name of the output file after decoding. If missing use stdout.
-D	The name of the text file to be used to find the Caesar cipher shift to be used to decode the text fields of each record. If missing, the text fields are left undecoded when copied.
-r	Copy the records in reverse order. If missing, the records are copied in order.

NOTES (IMPORTANT INFO ... READ CAREFULLY)

1. Be sure to copy your source code files into a directory where they will be safe in the event that you accidentally remove everything from your working directory. It is a good idea to make a directory named backup in which you regularly place copies of your C files.
2. Your code must compile cleanly with no error or warning messages using the -Wall flags in gcc.
3. The assignment must be written in C and run on the School's Linux server.
4. Your source code should contain brief comments describing the functionality and the major components of each procedure. Any complex structures should also be commented. Your source code should be properly formatted and meaningful variable names should be used.
5. A readme file should be submitted along with your source code explaining how your program should be run as well as any limitations of the code.
6. **If you hand in an assignment that does not compile you will get a zero grade.**
7. All work in this course is to be done independently.
   Submissions will be electronically examined for similarity.
8. You should hand in your source code file (.c and .h files) and a makefile.
   If no working makefile is supplied, you will be given a zero grade.

Hint: Answer to a Student's Question

Question: For the input string (from either a file or stdin), can we assume a max length?

Answer: No.

First, for the majority of the functions, a null character terminated string, that has already been read in elsewhere, is given as an argument, so you don't need to make any assumptions.

The problem, of course, is with the "main" programs, where the file is being read in.

If the -F is a command line argument:

You can use the file length seek/tell "trick" from A1 to get the length of the file.

However, if you think about it, you don't even need to do this for Q1. Just have a fixed size input array, and fill it as many times as you need to get the character counts to find the shift. Then rewind (or close then open) the file and apply the shift copying the file in "input array sized" chunks.

If the input is from stdin:

For Q1, you can read in the characters in bite-sized chunks and process them as you go along (counting characters). There is no need to store the entire file.

For Q2, an upper limit can be defined and stated in your readme, although this will lose 2 marks out of 50 marks (note: there are 40 marks for the questions + 10 style marks).

However, a better solution would be to create a simple linked list of filled input arrays. This approach would work for -F as well, so if implemented you don't need to do use the seek/tell trick; just use this routine to handle both cases.