CS 1713, Introduction to Computer Science
Assignment 8, Spring 1998
Due Tuesday, March 31, 1998
(Second Exam Postponed to April 2, 1998)
Letter Frequency Table

For this assignment you must write a program to produce a *letter frequency table* for an input file. For each lowercase letter from `'a'` to `'z'`, you should determine the frequency of occurrence of the letter. Input to the program will be any text file (like program source code or a file of English text), and the program's output will be a table of 26 letters and frequencies, sorted into decreasing order by frequency. Here's how to build up this program:

1. Start with a simple loop that reads each character of the standard input file, `stdin`. One version of this loop is:

```
int ch;
while ((ch = getchar()) != EOF) {
        /* do various things with ch */
}
```

   The activities mentioned in steps 2 to 5 below will all go inside this loop. Eventually, you will execute the resulting program by redirecting the largest file into it that you can find:

```
runner$ freq < somebigfile.text
```

2. Use the function `isalpha` to recognize if `ch` is a letter. Use the function `tolower` to convert the character `ch` to a lowercase letter in case it is an uppercase letter. (If it is not a letter or if it is already lowercase, `tolower` will leave it alone. `tolower` and `isalpha` need the header file `<ctype.h>`.)
3. Look up `ch` in a table of the 26 lowercase letters. (We will discuss this in class. For example, if the letter is a `'d'`, then the lookup should return index 3.)
4. Get a count of the total number of letters in the file. (Do not count other characters.)
5. Create an array of 26 counters, one for each lowercase letter. For each letter, the corresponding counter should be incremented. (For example, if the array of counters is:

```
int letter_count[26];
```

   then a `'d'` should cause `letter_count[3]` to be incremented.)
6. After the input loop terminates, and the program has counted the total number of letters and the number of each kind of letter, the program should sort the array of counters into *decreasing* order. You can use bubblesort, as discussed in the text and in class. Each time two counters are interchanged, you should also interchange the corresponding letters, so that the letter in position `i` always corresponds to the counter in position `i`, even after rearranging. The bubblesort must be a separate function, with the two arrays as parameters.
7. Finally, the two sorted arrays should be printed, in 26 lines, with the letter first and then the *percent* frequency, which will be 100.0 times the count for the individual letter divided by the total number of letters.

Here is the start of sample output from my own program using a file of English text:

```
runner% freq <tempwalden                                u:  2.908%
Frequency count of letters, out of total: 138567        m:  2.872%
e: 12.160%                                              p:  2.480%
t:  9.196%                                              f:  2.105%
a:  7.824%                                              y:  2.021%
i:  7.802%                                              g:  1.827%
o:  7.588%                                              w:  1.794%
n:  6.872%                                              b:  1.473%
s:  6.770%                                              v:  1.189%
r:  5.948%                                              k:  0.815%
h:  4.653%                                              x:  0.212%
l:  4.071%                                              j:  0.142%
d:  3.698%                                              q:  0.115%
c:  3.380%                                              z:  0.086%
```