CSCI-1200 Data Structures — Fall 2022
Homework 2 — Spelling Bee Classes

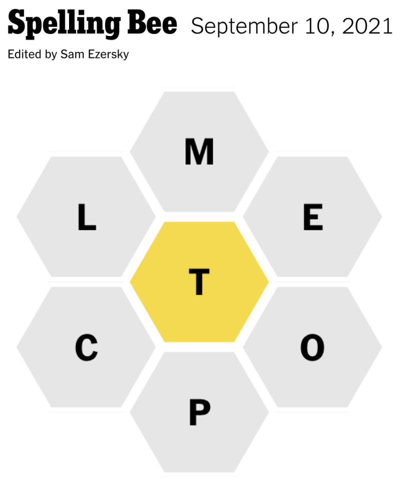
In this assignment you will practice using C++ classes to parse and compute statistics from the New York Times online word game, *Spelling Bee*: https://www.nytimes.com/puzzles/spelling-bee.
Please carefully read the entire assignment before beginning your implementation.

Every day the New York Times Games editors make a new Spelling Bee game consisting of 7 letters, with one letter highlighted in gold as the center letter. To play the game, you need to build English words from these letters, subject to the following constraints:

- Words must contain at least 4 letters.
- Words must include the center letter.
- Letters can be used more than once.
- The solution list excludes proper nouns and words considered obscure or offensive.

Your game score is computed as follows:

- 4-letter words are worth 1 point each.
- Longer words earn 1 point per letter.
- Each game includes at least one "pangram" which uses every letter. These words are worth an extra 7 points!



In the example above, "tool", "poet", and "motel" are valid words (worth 1, 1, and 5 points, respectively). "top" is not a valid word (it is too short) and "cope" is not a valid word (it does not use the center letter 't'). This game contains 1 pangram, "complete", which uses all 7 letters and is worth 8+7 = 15 points. The New York Times Games editors have declared the game for this date to have 56 official answers. The maximum score (sum of the points for those answers) is 226.

```
-------------------
September 10, 2021
[ t ] c e l m o p
-------------------
clot
collect
collet
colt
comet
compete
complete
compote
coopt
coot
cote
elect
electee
emote
locomote
loot
lotto
meet
melt
mete
```
etc.

An archive of all past Spelling Bee games, the solutions, and some interesting statistics is available at https://nytbee.com/. We have scraped this data and prepared a variety of input files of different sizes for this assignment. Each input file contains 1 or more Spelling Bee games and all of the answer words for each game. For example, *a portion of* the input file `sample.txt`, which contains the game above, is shown on the left.

Each game contained in an input file starts with a row of dashes, then the date, then the 7 letters, with the center letter first, surrounded by square brackets. After another row of dashes, the valid answer words are listed. This file, and other sample input files, are available on the course webpage.

File I/O and Command Line Arguments

Your program will run with three command-line arguments. The first is the input file containing one or more games as described above. The second is the output file where you will write the computed statistics. The third argument will indicate which data table should be printed. Valid options for the third argument are: `--game_stats`, `--word_stats`, `--letter_stats`, or `--custom_stats`. For example, here is a valid command line to your program:

```
./bee_stats.out small_test.txt output_small_test_games.txt --game_stats
```

Statistics Collected and Output

To illustrate each of the output formats, we'll use the `small_test.txt` input file example. This file contains 3 games. These dates have some of the lowest ever maximum scores (and similarly the fewest number of solution words) of all Spelling Bee games archived in the https://nytbee.com/ website.

When `--game_stats` is specified, your program should create a table of data with one row for each game including the date, letters, maximum score, number of pangrams, total number of words, and breakdown of the words for each game by length. The first game in the table below has 15 4-letter words, 5 5-letter words, and 1 8-letter word.

```
date                letters           score  pangrams  #words    4  5  6  7  8
------------------------------------------------------------------------------
December  14, 2019  [ u ] a b c f k l    55         1      21   15  5        1
May       24, 2020  [ i ] k l n r w y    52         1      21   16  3     2
May       17, 2019  [ l ] a b k r u w    50         1      21   16  4     1
```

The games in the table should be sorted first by number of **pangrams** (words using all of the 7 different letters), with the highest number of pangrams first. Games with the same number of pangrams should be ordered by maximum score, with the highest score first. And games that also tie in score should be sorted chronologically, with the most recent game last. You should strive to match the provided sample output exactly – see the example code from lecture and look at STL `iomanip` library references.

When `--word_stats` is specified, your program should create a table with all of the words contained in all of the games in the input file. The row for each word will include the count of how many games in the input contain the word as a solution. Furthermore, that count should be broken into subcounts of how many of those games had each letter `a-z` as the center letter. This table should be sorted first by the number of times the word appears in the file. And secondly alphabetically by word. Here is the *top portion* of the `--word_stats` output:

```
word   count    a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z
--------------------------------------------------------------------------------------------
bulb       2                                     1                          1
bulk       2                                     1                          1
bull       2                                     1                          1
luau       2                                     1                          1
lull       2                                     1                          1
lulu       2                                     1                          1
aural      1                                     1
balk       1                                     1
ball       1                                     1
bawl       1                                     1
blab       1                                     1
bluff      1                                                                1
```

etc.

When `--letter_stats` is specified, your program should create a table with a row for each letter `a-z` and record the count of the number of times that letter was the center letter for a game in the input file and the average maximum score for those games. And it should also count the number of games that include that letter as one of the non-center letters, and the average maximum score for those games.

```
letter   #center   avg center   #other   avg other
--------------------------------------------------
a                                    2        52.5
b                                    2        52.5
c                                    1        55.0
```

```
d
e
f                                    1        55.0
g
h
i              1         52.0
j
k                                    3        52.3
l              1         50.0        2        53.5
m
n                                    1        52.0
o
p
q
r                                    2        51.0
s
t
u              1         55.0        1        50.0
v
w                                    2        51.0
x
y                                    1        52.0
z
```

Finally when the `--custom_stats` option is specified, it is a chance for you to be creative. You will design and implement some other statistic from the Spelling Bee Game data. An important task for this part of the assignment is to write a concise description (∼ 100-200 words) of your new statistic. Put this description in your `README.txt` along with any other notes for the grader.

You are encouraged to modify the provided datasets and create new datasets to test and debug your code, and to demonstrate your custom statistic. Paste an interesting sample of the output from your custom statistic output into your README.txt. Extra credit will be awarded to custom statistics that require non-trivial extensions beyond the required portions of the assignment.

Useful Code

To control the formatting of your tables, you'll want to read up on the various STL iomanipulators: `std::setw(int)`, `std::setprecision(int)`, and `std::fixed`. And don't forget about the `sort` function that can be used to order the contents of a `vector`.

You should be able to parse the input files using only the `>>` input stream function for `int`, STL `string`, and `char` used in Lecture 3. You should not need `getline` or `eof`. Your code should not depend on the presence or location of newlines within the input file. Be sure to study the File Parsing Example with Different Data Types from the course webpage.

Program Requirements & Submission Details

Your program should involve the definition of *at least two classes* that have their own `.h` and `.cpp` files, named appropriately. You should also continue your practice of STL `vector`, STL `string`, and STL input/output/file streams. You *may not* use other STL containers that we have not yet covered in lecture yet (e.g., `map` or `set`) – we will cover these other data structures in a few weeks, and we will also discuss their structural differences and performance advantages in great detail.

Be sure to read "Good Programming Practices" as you put the finishing touches on your solution. You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" statement. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.