Project #9 – Gettysburg Word Analysis

1 Objective

Use Python sets, along with lists and text file I/O, to create a list of distinct words found in a specific file. Also generate a word-length frequency report.

2 Problem Overview

A file containing words from the Gettysburg Address is provided for you; it contains the words from Lincoln’s famous address. The desired output includes various analyses of the words in the file.

3 User Interface

3.1 Input

One file is provided for you: **GettysburgAddress.txt**. It contains the words from the address with no punctuation, with all words in lower case. Words are separated by spaces. No other input is necessary; the user does not need to be asked for any additional information.

3.2 Output

Output must include:

- A file GettysburgWords.txt which will contain an alphabetized list of distinct words from the original file. “Distinct” means that the word “the” would be present only once, for example, regardless of how many times it might appear in the original address.
- On-screen output showing the number of distinct words in the address, the longest word in the address, and the length of the longest word.
- On-screen output showing a word length frequency report.

For example, if the original file were to contain the following Steve Jobs quote:

your work is going to fill a large part of your life and the only way to be truly satisfied is to do what you believe is great work and the only way to do great work is to love what you do if you haven't found it yet keep looking don't settle as with all matters of the heart you'll know when you find it

…then the output file would contain these words, *one per line*, in this order:

```
a
all
and
as
be
believe
do
don't
fill
find
found
going
great
haven't
heart
if
is
it
keep
know
large
life
looking
love
matters
of
only
part
satisfied
settle
the
to
truly
way
what
when
with
work
yet
you
you'll
your
```

For that same data, on the screen you would also see this nicely aligned output (no tabs, please!):

```
Number of distinct words:   42
Length of longest word:      9
Longest word:  satisfied

Length =   1 , Count =   1
Length =   2 , Count =   8
Length =   3 , Count =   6
Length =   4 , Count =  13
Length =   5 , Count =   7
Length =   6 , Count =   2
Length =   7 , Count =   4
Length =   8 , Count =   0
Length =   9 , Count =   1
```

Note: if there are multiple words of the maximum length, your algorithm may show any one of them. Leave index 0 unused; there should be no words of zero length.

4 Functions

A main function should drive the program. It should call the functions shown below. Please read carefully and distinguish where *set* is used vs. where a *list* is used. You must use the exact name/case function names shown below, with the exact parameters shown (order and data types), or your instructor's test code will fail, and the work returned to you for correction and resubmission.

- **readFile** – reads the specified file and returns a *set* containing the distinct words in the file. Parameter: the name of the file to be read. Return: *set* of distinct words.
- **findLongest** – finds the longest word in a *set* of words. Parameter: *set* of words. Return: longest word, length of longest word
- **generateSortedList** – given a *set* of words, returns an alphabetically sorted *list* of words. Uses a Selection Sort (not Python's built-in sort method) to do its work; start with the code provided in class. Parameter: *set* of words. Return: *list* of words.
- **writeWordFile** – writes words to a file, one word per line. Parameters: the name of the file to write and a *list* of words. Return: nothing.
- **generateLenFreqs** – creates a list of word length frequencies. Parameter: *set* of words, maximum word length in the set. Ask about how to create the initial list; there is a good Python trick! Return: *list* of word frequencies.

5 Code Specifications

- At the bottom of the program, you should have a call to main().
- Include header comments at the top of each file. Include your name, the date, and a brief description of what the program does.
- Include comments for each section saying what is going on in the lines of code below.
- Use comments elsewhere as you think they help guide the reader. Do not overdo, though! Not every line needs a comment; think about describing a block of related code.
- Use blank lines to separate sections and provide visual "breathing room."
- Use descriptive variable and function names.

6 Extra Credit Options

6.1 Extra Credit

For 3% extra credit, create a PyUnit test script called TestWordAnalysis.py. In it, create an automated test case for these functions: findLongest, generateSortedList, and generateLenFreqs. Use the __main__ trick shown in class in your core program, to avoid test code calling your main() function.

6.2 Super Extra Credit

For 3% *additional* extra credit, also create a test case for readFile and writeWordFile; those will take more thought and perhaps the creation of a simple, shorter test file. If you create any such test files, include them in your submission so the tests will run properly for grading.

Use the materials in the Canvas Resource module (at the top), including the Introduction to Software Testing document, and the two linked videos that demonstrate how to construct automated tests.

7 Summary

At the bottom of your program, add comments that answer these questions:

- How did you approach this assignment? Where did you get stuck, and how did you get unstuck?
- How did you test your program? What does not work as you'd like, perhaps things that you'd like to fix as you learn more?
- What did you learn from this assignment? What will you do differently on the next project?

8 Grading Matrix

Area	**Pct**
File read	10
Distinct set of words generated	10
Sorting	10
File written	10
Longest word/length found	10
Frequency report generated	20
Output content and alignment	10
Test cases	10
Comments, variable names, white space	5
Summary report	5
Extra credit	3
Super extra credit	3
Total	**106**