# Project 2, Part 2

(50 points) Write a program that reads in a line of text and computes the frequency of the words in the text. Assume that the input contains words separated by white spaces, comma, period, or exclamation point. Sample input/output:

```
Input: I came, I saw, I conquered!
Output:
I               3
came            1
saw             1
conquered       1
```

1) Name your program `frequency.c`.
2) Assume input is no longer than 1000 characters. Assume the input contains no more than 1000 words. Assume each word is no more than 50 characters.
3) Store a word into a string array when it is first encountered. Create a parallel integer array to hold a count of the number of times that each particular word appears in the input. If the word appears in the input multiple time, do not add it to the string array, but make sure to increment the corresponding word frequency counter in the parallel integer array.
4) You may use any string library functions such as `strtok, strcmp,` and `strcpy`.
5) To read a line of text, use the `read_line` function in the lecture notes.

**Before you submit:**

1. Compile with –Wall. Be sure it compiles on ***circe*** with no errors and no warnings.

*gcc –Wall frequency.c*

2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

*chmod 600 frequency.c*

3. Test your program with the shell scripts on Unix:

*chmod +x try_frequency*
*./try_frequency*

4. Submit frequency.c on Canvas.

Total points: 100 (problem 1: 60 points, problem 2: 40 points)

1. A program that does not compile will result in a zero.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality 80%

## Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your **<u>name</u>**.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.
3. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: **`#define PI 3.141592`**
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
9. Use underscores to make compound names easier to read: **`tot_vol`** or **`total_volumn`** is clearer than totalvolumn.