Project 6, Program Design

1. (100 points) Write a program that prompts the user to enter the name of a file. The program finds the anagrams in the file.

```
Enter the file name: words.txt

Output: anagrams are written to file: words.txt.ang
```

The program reads the content of the file and stores the words in an array of strings, the program then finds the anagrams and writes the anagrams to the output file.

1. Name your program *fileIO.c*. The output file name should be the same name as the input file but with an added extension of `.ang`. In this example, the original file name is `words.txt`. The output file name is then `words.txt.ang`. Assume the file name is no more than 100 characters. Assume the length of each line in the input file is no more than 100 characters. Assume the input file contains no more than 1000 words.

2. The following function is provided in *`anagram.c`*:

```
int are_anagram(char *word1, char *word2);
```

`word1` and `word2` are strings containing the words to be checked for anagram. The function returns 1 if the two words are anagram of each other and return 0 otherwise.

3. The output file should be in the following format. The output file contains all the anagrams.

```
1       inch
        chin

2       roast beef
        eat for BSE

…
```

Before you submit:

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

gcc –Wall fileIO.c

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

chmod 600 fileIO.c

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

chmod +x try_file
./try_file

Total points: 100

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 **name**.
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.