Project 6, Program Design

Problem – File suffix

You are given N files as command-line arguments. Ex:

```
./a.out file1.txt file2.txt … fileN.txt
```

Each file contains a list of words, with exactly one word per line (there is no limit to the number of words). Each word is composed of at most 20 lowercase letters. Ex:

file1.txt	file2.txt	file3.txt
dog cat rat	woodpecker hippopotamus chimpanzee	hippopotamus chimpanzee woodpecker

Your task is to compute a suffix with at most M characters ($M \geq 20$) for each file. The rules for creating the suffix are the following:

1) the suffix for each file starts as an empty string;
2) one by one, you must read a word from the file and concatenate it to the suffix if the length of the resulting string is below the limit of M characters;
3) every time a word cannot be concatenated to the suffix due to the lack of space, you must "reset" the suffix (set it to an empty string again) to open space for the new word.

Below we exemplify these rules for the three files above with M = 20:

Steps	Suffix (length)	Next word (length)	Action
1	"" (0)	"dog" (3)	Concatenate
2	"dog" (3)	"cat" (3)	Concatenate
3	"dogcat" (6)	"rat" (3)	Concatenate
4	"dogcatrat" (9)		

Steps	Suffix (length)	Next word (length)	Action
1	"" (0)	"woodpecker" (10)	Concatenate
2	"woodpecker" (10)	"hippopotamus" (12)	Reset & Concatenate
3	"hippopotamus" (12)	"chimpanzee" (10)	Reset & Concatenate
4	"chimpanzee" (10)		

Steps	Suffix (length)	Next word (length)	Action
1	"" (0)	"hippopotamus" (12)	Concatenate
2	"hippopotamus" (12)	"chimpanzee" (10)	Reset & Concatenate
3	"chimpanzee" (10)	"woodpecker" (10)	Concatenate
4	"chimpanzeewoodpecker" (20)		

For each input file, print one line with the file name followed by the computed suffix.

Hint #1: You can use the function strlen() to check the length of the next word and the current suffix.
Hint #2: You can use the function strcpy() to reset the suffix.

Hint #3: You can use the function strcat() to concatenate two strings.
Example #1:

```
$ ./a.out file1.txt file2.txt file3.txt
Enter the value of M: 20
file1.txt: dogcatrat
file2.txt: chimpanzee
file3.txt: chimpanzeewoodpecker
```

Example #2:

```
$ ./a.out file1.txt file2.txt file3.txt
Enter the value of M: 40
file1.txt: dogcatrat
file2.txt: woodpeckerhippopotamuschimpanzee
file3.txt: hippopotamuschimpanzeewoodpecker
```

Testing guidelines:

1. Download the files *try_project6_suffix* and *project6_files.zip* from Canvas, and upload them to the **student cluster (sc.rc.usf.edu)**. Change the file permissions of the test script with the following command:

   chmod +x try_project6_suffix

2. Unzip the test files with the following command:

   unzip project6_files.zip

3. Develop your solutions with the following program name: *project6_suffix.c*
4. Upload your solutions to the student cluster.
5. Compile and test your solution for Project 6:

   gcc -Wall -std=c99 project6_suffix.c

   ./try_project6_suffix

Submission instructions:

1. Download the program *project6_suffix.c* from student cluster and submit it on Canvas>Assignments.

Grading
Total points: 100 (50 points each problem)

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. 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.
4. Use consistent indentation to emphasize block structure.
5. Full line comments inside function bodies should conform to the indentation of the code where they appear.
6. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: **`#define PI 3.141592`**
7. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
8. Use underscores to make compound names easier to read: **`tot_vol`** or **`total_volumn`** is clearer than totalvolumn.