Project 6, Program Design

Suppose you are given files containing a paragraph that are separated into lines. Write a program to remove the newline characters in the paragraph and replace with white spaces and store it in the output file.

Example input/output:

```
Enter the file name: message1.txt

Output file name: new_message1.txt
```

1) Name your program *project6_paragraph.c*
2) The output file name should be the same name but an added `new_` at the beginning. Assume the input file name is no more than 100 characters. Assume the length of each line in the input file is no more than 10000 characters.
3) Use fgets function to read from the input file.
4) The program should include the following function:

   `void convert(char *s1, char *s2);`

   The function expects `s1` to point to a string containing a line in the input file, `s2` to point to a string storing the content in s1 after the removing the newline character and replacing with a white space.

   This function should use pointer arithmetic– not subscripting – to visit array elements. In other words, eliminate the loop index variables and all use of the [] operator in the function.

5) String library functions are NOT allowed in the convert function.
6) String library functions are allowed in the main function.

Before you submit

1. Compile both programs with –Wall. –Wall shows the warnings by the compiler. Be sure it compiles on ***student cluster*** with no errors and no warnings.

   gcc –Wall project6_paragraph.c

2. Test your fraction program with the shell scripts on Unix:

   chmod +x try_project6_paragraph
   ./try_project6_paragraph

3. Submit project6_paragraph.c, and text files (for grading purposes) on Canvas.

Grading

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% (Including **functions implemented as required)**

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.