EE231002 Introduction to Programming

Lab09. Word Processing

Due: Nov. 24, 2018

Word processing has been one of the major applications for computers. In this assignment you will try to write a program to perform simple word processing with a fixed character width, such as the workstation terminal display. Today's powerful word processors use a similar concept but with more complicated fonts and more flexible font positioning.

Let's assume the output has N characters per line. Thus, any line which has more than N characters will take more than one output line. It is very likely that line-change happens within a word. This would make the output less legible. Thus, a reasonable word processor would break a line only on word boundaries. Assuming each paragraph is read in as a single string, the number of characters of each paragraph can be less than, equal to or larger than N, your assignment is to write four functions, in addition to the `main` function.

1. `int readLine(char para[LSTR]);`
   This function reads in a paragraph of text input as a single string and stores it in the `para` array given by the function argument. The array size `LSTR` should be defined as a macro in your program as the following.

   ```
   #define LSTR 5000
   ```

   This function is similar to the `read_line` function in the text book but with three differences:

   1.1. The parameter of the function is different.

   1.2. This function detects the end of the input file. If the string it reads in is equal to the literal `"EOF"` then it returns 0, otherwise it returns 1. Using this return value properly, your program can handle text files of any size.

   1.3. In order to discern sentences in a paragraph, it will check the end of a sentence and ensure that exactly two spaces exist between two sentences. That is, if a period (`'.'`) is detected and it is not the end of a paragraph, then two spaces must be exist before next non-space character.

2. `void center(char *para);`
   This function prints out the paragraph with each line centered. Assuming $N = 65$, example output of this function is:

```
          11111111112222222222333333333344444444445555555555666666
 123456789012345678901234567890123456789012345678901234567890123456789012345
 ---------------------------------------------------------------------------
 |  "He has been phenomenal," Bryant said.  "We have watched some  |
 |tape on him.  We came up with a strategy that we thought would be|
 | effective, but he was knocking down his jump shot, penetrating  |
 |                  and he got around our guards.                  |
 ---------------------------------------------------------------------------
```

Note that blank spaces are added to the beginning of a line to make the line printed at the center.

3. `void lAlign(char *para);`
   This function prints out a paragraph with left edge aligned. Assuming $N = 65$, example output of this function is:

```
         11111111112222222222333333333344444444445555555555566666
12345678901234567890123456789012345678901234567890123456789012345
"He has been phenomenal," Bryant said.  "We have watched some
tape on him.  We came up with a strategy that we thought would be
effective, but he was knocking down his jump shot, penetrating
and he got around our guards.
```

4. `void rAlign(char *para);`
   This function prints out a paragraph with right edge aligned. Assuming $N = 65$, example output of this function is:

```
         11111111112222222222333333333344444444445555555555566666
12345678901234567890123456789012345678901234567890123456789012345
    "He has been phenomenal," Bryant said.  "We have watched some
tape on him.  We came up with a strategy that we thought would be
   effective, but he was knocking down his jump shot, penetrating
                                and he got around our guards.
```

With these functions, one can read in a text file and print it out in different formats. One can print it with very paragraph left-edge aligned, centered or right-edge aligned. Two files are provided for you to test your program's capabilities. They are `story1.txt` and `story2.txt`. Both files are ended with a line `"EOF"`.

Your program should accept two command line arguments. The first argument specifies the line length, N, to be printed. The range of N is: $60 <= N <= 100$. The second argument can be ℓ, `c`, or `r`. ℓ specifies left-edge aligned print out, `c` specifies centered print out, and `r` specifies right-edge aligned print out. Example running the compiled program is as follows.

```
$ ./a.out 65 c < story1.txt
```

where `< story1.txt` uses the `unix` input redirection. The content of the file `story1.txt` is read in as the standard input, and thus `scanf` can be used to read in strings directly from the file.

Notes.

1. Create a directory `lab09` and use it as the working directory.
2. Name your program source file as `lab09.c`.

3. The first few lines of your program should be comments as the following.

```
/* EE231002 Lab09. Word Processing
   ID, Name
   Date:
*/
```

4. After you finish verifying your program, you can submit your source code by

   ```
   $ ~ee2310/bin/submit lab09 lab09.c
   ```

   If you see a "submitted successfully" message, then you are done. In case you want to check which file and at what time you submitted your labs, you can type in the following command:

   ```
   $ ~ee2310/bin/subrec lab09
   ```

   It will show youe submission records for lab09.

5. You should try to write the program as efficient as possible. The format of your program should be compact and easy to understand. These are part of the grading criteria.