EE231002 Introduction to Programming

Lab09. Word Processing

Due: Dec. 3, 2016

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 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:

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, the argument is different and also this function will detect the end of the input file as following. If the string it reads in is equal to the literal `EOF` then it returns 1, otherwise it returns 0. Using this return value properly, your program can handle text files of any size.

2. `void leftAlign(char *para);`
   This function prints out a paragraph with the left edge aligned. Assuming $N = 64$, example output of this function is:

   ```
   "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.
   ```

3. `void center(char *para);`
   This function prints out the paragraph with each line centered. Assuming $N = 64$, example output of this function is:

```
"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 and the end of a line to make the line printed at the center.

4. `void bothAlign(char *para);`
   This function prints out a paragraph with both left edge and right edge aligned. Assuming $N = 64$, example output of this function is:

```
"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 **evenly** in the line such that both edges of the printed line are aligned. Note also that when the remaining line has less than N characters, the it is printed with left edge aligned.

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 both-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 `b`. ℓ specifies left-edge aligned print out, `c` specifies centered print out, and `b` specifies both-edge-aligned print out. Example running the compiled program is as follows.

```
$ ./a.out 64 b < 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` or `gets` 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

   ```
   $ ~ee231002/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:

   ```
   $ ~ee231002/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.