EE231002 Introduction to Programming

Lab09. Word Processing

Due: Nov. 28, 2015

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 write a function to print out a string broken to exactly `len` characters per line.

```
/* the following function print out a paragraph line by line            *
 *   Input: char *para, is the string of a paragraph,                   *
 *          int len, is the number of characters for the first output line.
 *   Output: the paragraph is printed directly using printf or puts,     *
 *           no return value is needed.                                 *
 *   Side effects: the contents of the paragraph can be modified by this *
 *                 function.                                            */
void printLine(char *para,int len);
```

Using this function, and other functions if you wish, to print out a text file by the following rules:

1. The first line of each paragraph needs to be indented by 4 spaces.
2. The right edge of each line needs to be aligned. The `N`'th character should not be a space, unless it is the last of a paragraph and the line length is less than `N`. In the latter case, the right edge needs not be aligned.
3. To satisfy the right edge aligned rule, some space characters need to be added to the output lines. These added spaces should be evenly distributed for better legibility. Again, the last line of each paragraph needs no extra space characters.

The number of characters per line, `N`, should be a global variable, and it should be given by the command line when the program is executed. The range of this variable is 50<=`N`<=100. For example, the following program execution sets the output line length to 64.

```
$ ./a.out 64 < story1.txt
```

Two files are provided for you to test your program: `story1.txt` and `story2.txt`. Both files have a special line to end the file: `EOF`. Your program should also detect this special line to end the program execution. As shown above, these files can be read directly using linux input redirection. The output of your program can also be saved to a file using output redirection.

```
$ ./a.out 64 < story1.txt > story1.out
```

The file `story1.out` will be created and can edited using `vim` command. It would be useful to save the output to a file and view the file to check if some unprintable characters have been created by your program. Example output is listed at the end of this `pdf` file.

Notes.

1. Please make sure your program can handle different line length output. The range of `N` is 50 – 100.
2. Create a directory **`lab09`** and use it as the working directory.
3. Name your program source file as **`lab09.c`**.
4. The first few lines of your program should be comments as the following.

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

5. 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
   ```

   It will show the last few submission records.

6. 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.

Example output:

```
    Today You, Tomorrow Me

    --posted by Justin Horner on Mar 10, 2011

    During  this  past year  I've  had  three instances  of  car
trouble: a blowout on  a freeway, a bunch of  blown fuses and an
out-of-gas  situation. They  all happened  while  I was  driving
other people's cars, which for some  reason makes it worse on an
emotional level. And on a practical level as well, what with the
fact that I carry  things like a jack and extra  fuses in my own
car, and know  enough not to park  on a steep incline  with less
than a gallon of fuel.

    Each time, when these things  happened, I was disgusted with
the way people didn't bother to help. I was stuck on the side of
⋮
⋮
    In the  several months since  then I've changed a  couple of
tires, given a few rides to gas stations and once drove 50 miles
out of my way to get a girl to an airport. I won't accept money.
But  every time  I'm able  to help,  I  feel as  if I'm  putting
something in the bank.

    [From a post on reddit.com and re-published in NY Times.]
```