Project 5 report

1. A brief description of notable obstacles you overcame.
   1. I struggled with the test case “@P@-a-@P@” I wrote. According to my design (and the project spec), this cstring will be divided into 3 tokens (word portions): “@P@-”, “a-”, and “@P@”. However, according to my design, the 3rd token, which is “@P@”, will be treated as a valid New Paragraph command (It is actually not). So, I had to implement a new variable, called the prevTokenType, which records whether the prevToken contains ‘-’ or not. If the prevToken contains a ‘-’, that means there is no space between the previous token and the “@P@” token, making the “@P@” token a NON-VALID New Paragraph command, and a new paragraph should not be produced.
   2. Also, when I was developing my code, I realized there is a lot of redundancy and repetitive code in my solution, which makes debugging and reading very hard. Then, I thought of creating functions to remove such repetitive code, such as **void** outputToken and **void** processToken and **bool** getNextToken to improve the readability of my code. After that, I found debugging much easier since I don’t have to copy snippets of my code from here to there repetitively, and my solution is less prone to have bugs.
   3. In some cases, a space is required between outputting tokens, however, when a token contained ‘-‘at the end, a space is not needed. I initially struggled on how to implement this feature. Then, I realized that I can use an int variable called prevTokenType, to note whether the previous token is actually a word-portion that contained a ‘-‘ or not. Then, my program checked prevTokenType every time I print out a space, to prevent unwanted spaces from appearing in the output file.
2. Program Design

bool getNextToken(int& tokenType, istream& inf, char token[]){

while notEndOfFile:

getNextChar;

if char is space:

mark the end of token;

if char is ‘-’:

append char to token;

mark the end of token;

else:

append char to token;

void outputToken():

if token is exactly “@P@”:

if no new paragraph has been produced before:

output new paragraph before next token;

reset counter to 0;

else if token ends with ‘.’ Or ‘?’ or ‘!’ or ‘:’

output additional space before next token;

increment counter;

if token is not first word of line, and token does not have ‘-‘:

output space before token;

increment counter;

output token;

increment counter;

void processToken():

if token contains ‘-’:

if there is still space in Line for token:

outputToken();

else:

start the next line;

reset counter;

outputToken();

else:

if there is still space in Line for token and a space:

outputToken();

else:

start the next line;

reset counter;

outputToken();

if lineLength < tokenLength:

set returnOneFlag to True;

while(chars to be written will fit more than one line):

fill in whole line with chars in token;

start new line;

fill in the rest of line chars;

update counter;

int render():

if lineLength <1:

return 2;

else:

initialize all varaibles;

while (there is nextChar after currentChar):

processToken();

processToken(); //Process the last token

if returnOneFlag is true:

return 1;

else:

return 0;

1. Test Cases
   1. 0, “Test” (Test for lineLength < 1, should return 2)
   2. 10, “The quick brown fox jumps over a lazy dog” (Test for switching to next line – Also test for correctly putting spaces between tokens. No space at end or start of line. Return 0.)
   3. 20, “The. Quick! Brown: Fox? jumps over a lazy dog” (Test for extra space after period, question mark, exclamation point, or colon) Return 0.
   4. 10, “The. Quick! Brown: Fox? jumps over a lazy dog” (Test for extra space after period, question mark, exclamation point, or colon, but the current line cannot fit the next token after putting the double space)
   5. 3, “The quick brown fox jumps over a lazy dog” (Test word portion length larger than line length.) Return 1.
   6. 3, “The. Quick! Brown: Fox? jumps over a lazy dog” (Test for word portion length larger than line length, and with .?:!, but no double space should be produced.) Return 1.
   7. 10, “The. Quick! Brown: Fox? @P@ jumps over a lazy dog” (Test for valid @P@ command in word.)
   8. 10, “The. Quick! Brown: Fox? @P@ @P@ @P@ @P@ jumps over a lazy dog” (Test for multiple @P@, but only one new paragraph should be produced).
   9. 10, “@P@-a-@P@” (Test for invalid @P@ command)
   10. 10, “@P@@P@” (Test for invalid @P@ command)
   11. 10, “CS31@P@” (Test for invalid @P@ command)
   12. 1, “@P” (Test for invalid @P@ command)
   13. 10, “@P@” (Test for single @P@ Command, output file should be empty).
   14. 10, “” (Test for empty input. Output file should be empty).
   15. 10, “p @P@” (The very last output paragraph must not be followed by an empty line.)
   16. 10, “@P@ p” (The very first output paragraph must not be preceded by an empty line.)
   17. 10, “The-quick-brown-fox-jumps-over-a-lazy-dog” (Test for word portions splitted by ‘-‘)
   18. 2, “The-quick-brown-fox-jumps-over-a-lazy-dog” (Test for word portions splitted by ‘-‘, and handing word portions through different lines)
   19. 10, “The quick brown fox jumps over a lazy dog” (Test to see if renderer ignores white space between words)
   20. 5, “1234-123” (Word broken at hyphen to fit next line)
   21. 3, “-a--a---a----a” (Word portions at different splitted by hyphens – but word portion does not exceed line length) return 0
   22. 3, “a-aa-aaa-aaaa-” (Word portions at different splitted by hyphens – but word portion exceed line length) return 1