John Connell

Project 5 Report

Notable Obstacles

The main issue I encountered throughout the project was dealing with the first word in each input line. My initial design checked if the next word would go over the line length only after a word had been printed and spaces had been incremented. This meant that words at the beginning of each input line would not be checked by the isWordOverLine function. Once I realized the issue, it was easy to fix; however, it took me quite a while to figure out the issue. I need to learn how to use the debugger. Branching off of this, my main issue in this project was general organization. I should have spent an hour or two finalizing my design rather than brute forcing every first whim that came to my brain. After wasting about ten hours on the project, I took a step back and rethought my design, allowing me to make the rest of the process less painful. I thought of much more efficient designs closer to the end of the project, and if I had thought more before I typed, the whole process would have been less painful.

Design Summary

* Render takes in a line length, input file stream, and output file stream
  + If the line length is invalid(<1 or >255)
    - Return 2

//Initialize variables used throughout the function

* + Initialize int currentLength and set it to 0 //tracks the current output line length and set it to 0
  + Initialize bool isFirstBreak and set it to true //used to only print a single paragraph break when multiple @P@’s are present
  + Initialize bool isLastInput and set it to false //tracks if dealing with last input line
  + Initialize int returnValue and set it to 0
  + Initialize temporary length and set it to 0 //used to account for spaces when checking if a word would go over the lineLength
  + Initialize int fillerParBreaks and set it to 0 //used to omit @P@ at beginning of first line
  + Initialize bool no hyphen and set it to true //used for parsing through word segments with hyphen
  + Initialize bool firstWordOutted and set it to false //deals with case of first input word being too large
  + While there is still another line of input to receive
    - Fill string inputReader with the next line of input
    - Peek the next line and set isLastInput to true if it is equal to end of file
    - While the input line is only whitespace characters
      * Increment fillerParBreaks
    - While the input line is a paragraph break and firstWordOutted is false
      * Increment fillerParBreaks by 3 //to avoid @P@ before any output is written
      * While the input line has whitespace characters
        + Increment fillerParBreaks //accounts for whitespace characters between @P@’s
    - Call function wordSegment //places spaces after every hyphen and the rest of the input one to the right
    - For every index of inputReader starting at fillerParBreaks and ending when a \0 is present
      * if function noMoreWords returns true
        + Break out of the loop
      * Initialize int lastCharIndex to index-1 unless that is out of range
      * While isParagraphBreak returns true
        + Output two new lines and reset currentLength unless isLastInput is true or a paragraph break was just outputted(isFirstBreak is false)
        + Increment the index by 3
        + Reset the currentLength to 0
        + While the current character is a whitespace character

Increment index

* + - * If the current index is 0
        + If isWordOverLine returns 1 and firstWordOutted is true

Output a new line

Reset the currentLength to 0

While the word is still longer than the lineLength

Set the returnValue to 1

Output the character at index

Increment the currentLength

If the currentLength exceeds lineLength

Output a new line

Reset the currentLength

* + - * + Else if isWordOverLine returns 1 and firstWordOutted is false

Set the returnValue to 1 //word must be longer than lineLength if no words have been outputted

Output the character at the currentIndex until the output line is the same length as lineLength

While the word is still longer than the lineLength

Output the character at index

Increment the currentLength

If the currentLength exceeds lineLength

Output a new line

Reset the currentLength

* + - * + Else

//Since no new line was outputted

Output a space //to divide the last word of the previous input line

If the lastCharIndex character of inputReader is a valid punctuation mark

Output another space

* + - * If the current index of inputReader contains a whitespace character
        + While the current index of inputReader contains a whitespace character

Increment index

* + - * + While isParagraphBreak returns true

Output two new lines and reset currentLength unless isLastInput is true or a paragraph break was just outputted(isFirstBreak is false)

Increment the index by 3

Reset the currentLength to 0

While the current character is a whitespace character

Increment index

* + - * + Set temporaryLength to currentLength //holds spaces that haven’t been outputted to make isWordOverLine calculation valid

If lastCharIndex is greater than or equal to 0 and the currentLength is not 0 //accounts for no spaces at the beginning of an output line

If a !,?,:,. Is present

Increment temporaryLength

Else if a - is present

Set noHyphen to false

If currentLength is not 0 and noHyphen is true

Increment temporaryLength

Set noHyphen back to true

* + - * + If isWordOverLine returns 1 and firstWordOutted is true

Output a new line

Reset the currentLength to 0

While the word is still longer than the lineLength

Set the returnValue to 1

Output the character at index

Increment the currentLength

If the currentLength exceeds lineLength

Output a new line

Reset the currentLength

* + - * + //now that the length calculation is complete, spaces can be outputted

If a !,?,:,. Is present

Output a space

Increment currentLength

Else if a - is present

Set noHyphen to false

* + - * + If currentLength is not 0 and noHyphen is true

Output a space

Increment currentLength

* + - * + Set noHyphen back to true
        + Output the next character of inputReader
        + /\*all spaces and paragraph breaks have been incremented through by this point\*/
        + Increment currentLength
        + Set firstWordOutted to true//indicates a word has printed
    - Set the inputReader C String to empty by copying an empty string
  + Return the returnValue
  + //function isWordOverLine returns 1 if the word about to be printed would go over line length
  + Takes in the current index, current input line, current length, and lineLength
    - Initialize int wordLength to 0
    - While the character is not the zero byte or issprint
      * Increment wordLength
    - If wordLength plus current length is more than line length
      * Return 1
    - Otherwise, return 2
* //function hasHyphen returns true if the current word has a hyphen
* Takes in the current index and the current input line
  + Increment until the zero byte or a isspace character
    - If a hyphen is found
      * Return true
  + Return false
* //function isParagraphBreak returns true if a correctly formatted @P@ is present
* Takes in the current index and the current input line
  + Repeatedly from index until index + 3
    - If the zero byte is present
      * Return false //verifies checking for paragraph break won’t go out of bounds
  + If the characters ‘@’ ‘P’ ‘@’ ‘(some isspace character)’ appear in that order
    - Return true
  + Return false
* //function isRestParagraphBreak returns true if there are no remaining words that are not paragraph breaks
* Takes in the current index and the current input line
  + Repeatedly until the zero byte is reached
    - If the character isspace
      * Continue
    - Else if the character is an @
      * Return false if the next character is the zero byte or not a ‘P’
        + Return false if the next character is the zero byte or not a ‘@’
    - Else
      * Return false
  + Return true
* //function noMoreWords returns true if the rest of the input line is comprised of whitespace characters
* Takes in the current index and the current input line
  + For each character in the current input line until the zero byte
    - If the character is not a whitespace character
      * Return false
  + Return true
* //function wordSegment adds a space after every instance of a hyphen after moving all characters to the right of the hyphen one index to the right
* Takes in the current index and the current input line
  + Initialize int length to 0
  + For each character from index to the zero-byte
    - Increment length
  + For each character i from 0 to length in the input line
    - If a hyphen is present
      * For every character from length to the current i
        + Set the address one after the character equal to the current character
      * Set the character to the right of the hyphen to a space
      * Increment length to account for the additional space
      * Increment i past the space

Test Cases

\* Assume each case uses a lineLength of 50 and that the output goes to cout unless otherwise stated. Below is the text of the input file passed to render.

Paragraph Break Test Cases

* “@P@ @P@ @P@ Hi there” - paragraph break at beginning, multiple paragraph breaks at beginning
* “Hi there @P@ @P@ @P@” - paragraph break at end, multiple paragraph breaks at end
* “Hi @P@ @P@ @P@ there” - paragraph break in middle, multiple paragraph breaks at end
* “Hi @P@-@P@ there” - paragraph break as a word segment attached to another paragraph break
* “Hi-@P@ @P@ there” - paragraph break after paragraph break as a word segment
* “Hi @P@ @P@-there” - paragraph break before paragraph break as a word segment
* “@P@Hi there” -paragraph break as beginning of non-hyphenated word
* “Hi there@P@” - paragraph break as end of non-hyphenated word

Hyphen Test Cases

* “Hi-there” with a line length of 3 - line break with part of word on first line
* “Hi-there” with a line length of 10 - no unnecessary breaking of hyphenated word
* “Hi-there” with a line length of 2 - Word broken over multiple lines despite hyphen
* “Hi-there-I’m-writing-project-five-right-now” word length of 10 - large number of word segments
* “-----” line length of 1 -word segments of length 1

General Spacing Test Cases

* “. . . ! ? :” - two spaces after words with significant punctuation
* “ I “ -spaces at beginning and end of input line
* “ @P@ “ - Input line of nothing but paragraph break should not print extra line
* “ \t@@   
  \n\n\n hi” -whitespace characters prevalent, (MY CODE DOES NOT PASS THIS)
* “Hi   
    
  Bye” - empty line does not cause additional space in output

Various Return Cases

* Called with linelength of 0 - invalid lineLength too small
* Called with linelength of 251 -invalid lineLength too large
* “Hi there Hiihihihiiihihiihii” called with lineLength of 2 - return 1 and spread across lines, return updated when last word is over lineLength
* “@P@-@P@-@P@-@P@-@P@” lineLength of 4 -not cause error
* “THISISTHEFIRSTWORDOFINPUT” lineLength of 1 - return updated when first word is over lineLength

Other Cases

* “@P@” with a lineLength of 2 - lineLength shorter than full paragraph break(MY CODE DOES NOT HANDLE THIS CASE)
* ““ - empty file passed