if (wordLength > lineLength) //word being longer than a given line

wordGTLine++;

if (c == '\0')

continue;

if (c == ' ') //space characters

{

wordToLine++; //counts as a character

wordLength = 0; //spaces always separate words

outf << c;

if (puncSpace > 0) //refers to when period follows

{

outf << " ";

puncSpace = 0; //reset the next time.

}

continue;

}

if (c == '.' || c == '?') //periods and questions

{

puncSpace++;

wordToLine+= wordLength;

outf << c;

continue;

}

As of 5/17/18 9:50 PM

if (lineLength <1)

return 2;

char c;

//inf.get(c); doesn't restart

char holder[140]; //will use for hyphen cases. 140 because 139 regular plus 1 zero byte.

int wordLength = 0; //used to check if word is bigger than line

int spaceCount = 0; //used if input file has multiple spaces in between

int wordToLine = 0; //used to shift line if line length max is met.

int wordGTLine = 0; //used to count whether a word or portion exceeds line length

int poundP = 0; //used to count for paragraph breaks.

int puncSpace = 0; //used for tracking spaces after a '.' or '?'

while (inf.get(c))

{

if (wordLength > lineLength) //word being longer than a given line

wordGTLine++;

if (wordToLine == lineLength)

{

outf << '\n';

wordToLine = 0; //reset character count whenever we move to new line.

}

if (c == '\0')

continue;

if (isspace(c)) //space characters

{

if (spaceCount != 0)

continue;

if (wordToLine == 0) //case when space starts off

continue;

wordToLine++; //counts as a character, unless multiple spaces.

wordLength = 0; //spaces always separate words

if (puncSpace == 0)

{

outf << ' '; //one space whether it's a tab or regular space

spaceCount++;

}

continue;

}

if (c == '.' || c == '?') //periods and questions

{

puncSpace++;

wordToLine ++;

spaceCount = 0;

outf << c;

continue;

}

if (c == '-') //hyphen case

{

wordToLine++;

spaceCount = 0;

outf << c;

while (inf.get(c))

{

if (wordToLine == lineLength)

{

outf << '\n'; //case when hyphen perfectly ends as last character in a line.

wordToLine = 0; //reset character count whenever we move to new line.

outf << c; //need this or character right after - won't print.

break;

}

if (c == '\0')

continue;

if (c == ' ')

{

wordToLine++;

continue;

}

if (c == '-')

{

wordToLine++;

continue;

}

else

{

wordToLine++;

outf << c;

}

}

continue;

}

wordToLine++;

if (puncSpace > 0) //refers to when period follows

{

outf << " ";

puncSpace = 0; //reset the next time.

}

outf << c; //for any other character in general

spaceCount = 0; //reset space count whenever a nonspace character is entered

}

outf << '\n' ;

outf << wordToLine << endl;

//outf << wordLength << endl;

if (wordGTLine > 0)

return 1;

else

return 0;

}

Just printing out each character following punctuation rule only.

As of 5/20/18 1:04 AM

char c;

//inf.get(c); doesn't restart

char holder[140] = ""; //will use for hyphen cases. 140 because 139 regular plus 1 zero byte.

char character[5] = ""; //will use to hold each nonwhite space character and then concatenate to holder. 2 because one character and one zero byte

char empty[10] = ""; //will use to empty out holder

int wordLength = 0; //used to check if word is bigger than line

int spaceCount = 0; //used if input file has multiple spaces in between

int spaceGo = 0; //will use to make spaces after spaceCount

int wordToLine = 0; //used to shift line if line length max is met.

//int starterWord = 0; //used to start every word

int wordGTLine = 0; //used to count whether a word or portion exceeds line length

//int poundP = 0; //used to count for paragraph breaks.

int puncSpace = 0; //used for tracking spaces after a '.' or '?'

int puncGo = 0; //will use to create space after . or ?

while (inf.get(c))

{

if (wordLength > lineLength) //word being longer than a given line

wordGTLine++;

if (wordToLine == lineLength)

{

outf << '\n';

wordToLine = 0; //reset character count whenever we move to new line.

}

if (c == '\0')

continue;

if (isspace(c)) //space characters

{

if (spaceCount != 0) //if there are arbitrary consecutive spaces

continue;

if (strlen(holder) == 0)

continue;

if (spaceGo > 0)

{

if (wordToLine > 0)

{

outf << ' '; cerr << "spaceGo check" << endl;

spaceGo = 0;

}

//if (puncSpace > 0) //when period or ? ends a word

//{

//outf << ' ';

//puncSpace = 0;

//puncGo = 0;

//}

}

if (puncGo > 0)

{

if (wordToLine > 0 && (holder[strlen(holder)-1]!='.' && holder[strlen(holder)-1]!='?'))

{

outf << ' '; cerr << "puncGo check" << endl;

puncGo = 0;

}

}

wordToLine += wordLength;

wordLength = 0; //spaces always separate words

if (spaceCount == 0)

spaceGo++;

spaceCount++;

if (puncSpace > 0)

puncGo++;

for (int j = 0; holder[j] != '\0'; j++)

{

outf << holder[j];

}

strcpy(holder, empty); //will use to empty out holder

continue;

}

if (c == '.' || c == '?') //periods and questions

{

puncSpace++;

wordLength++;

puncGo++;

char character[2] = { c, '\0' };

strcat(holder, character);

continue;

}

if (c == '-') //hyphen case

{

wordToLine++;

spaceCount = 0;

outf << c;

while (inf.get(c))

{

if (wordToLine == lineLength)

{

outf << '\n'; //case when hyphen perfectly ends as last character in a line.

wordToLine = 0; //reset character count whenever we move to new line.

outf << c; //need this or character right after - won't print.

break;

}

if (c == '\0')

continue;

if (c == ' ')

{

wordToLine++;

continue;

}

if (c == '-')

{

wordToLine++;

continue;

}

else

{

wordToLine++;

outf << c;

}

}

continue;

}

wordLength++;

char character[2] = { c, '\0' };

strcat(holder, character);

spaceCount = 0;

puncSpace = 0;

}

if (!inf.get(c)) //after copying last word

{

int puncCheck = strlen(holder) -1;

if (spaceGo > 0)

{

if (wordToLine > 0 && strlen(holder) > 0) //no point if no words in a line to begin with or if the most recent word has no characters

outf << ' '; cerr << '1' << endl;

cerr << "pass" << endl;

}

if (puncGo > 0)

{

if (wordToLine > 0 && strlen(holder) > 0)

{

if (holder[puncCheck]!='.' && holder[puncCheck]!='?')

{

outf << ' '; cerr << '2' << endl;

cerr << "pass2" << endl; cerr << holder[4] << endl;

}

}

}

//if (puncSpace > 0)

//outf << ' '<<' ';

for (int i = 0; holder[i] != '\0'; i++)

outf << holder[i];

}

outf << '\n' ;

outf << wordToLine << endl;

if (wordGTLine > 0)

return 1;

else

return 0;

Let’s check this., let’s check this.(…)

As of 5/21/18 3:42 PM

Adding hyphen case

if (!inf.get(c)) //after copying last word

{

int puncCheck = strlen(holder) -1;

if (wordLength + wordToLine + spaceTotal> lineLength) //note: will adjust for cases with words longer than line

{

for (int i = 0; holder[i] != '\0'; i++)

{

if (holder[i] == '-')

{

hyphenCount++;

hyphen = i; //position where hyphen is

}

}

if (hyphen > 0) //when last word has hyphen

{

cerr << "split" << endl;

if (hyphen + 1 + spaceTotal + wordToLine <= lineLength)

{

if (spaceGo > 0)

outf << ' ';

if (puncGo > 0)

outf << ' ';

for (int j = 0; j <= hyphen; j++)

{

outf << holder[j];

}

cerr << "wordPortion" << endl;

outf << '\n';

for (int j = hyphen + 1; holder[j] != '\0'; j++)

{

outf << holder[j];

}

spaceGo = 0;

puncGo = 0;

}

}

else

{

outf << '\n';

wordToLine = 0; //restart

}

}

if (spaceGo > 0)

{

if (wordToLine > 0 && strlen(holder) > 0) //no point if no words in a line to begin with or if the most recent word has no characters

outf << ' '; cerr << '1' << endl;

cerr << "pass" << endl;

}

if (puncGo > 0)

{

if (wordToLine > 0 && strlen(holder) > 0)

{

//if (holder[puncCheck]!='.' && holder[puncCheck]!='?')

{

outf << ' '; cerr << '2' << endl;

cerr << "pass2" << endl;

}

}

}

if (hyphen <= 0) //print like a regular word if there's no hyphen

{

for (int i = 0; holder[i] != '\0'; i++)

outf << holder[i];

}

}

outf << '\n'; cerr << "newline" << endl;

cerr << "wordToLine: " << wordToLine << endl;