PROBLEM 1: TOKENIZATION

The Algorithm operates in the following manner:

- Get folder path using command line argument.
- Read all files in the folder one by one.
- Within each file, it reads line by line and splits into tokens with the help of split() method. (separated by white spaces)
- For each token generated, it performs following operations:
  - Handles case sensitivity by converting all the words into lowercase
  - Remove digits.
  - Ignore SGML tags.
  - Removes possessives.
  - Removes special characters such as !, @, #, $, %, ^, &, *, (,), [,], {, }, |, \, /, ', +, ; , :, .
  - Spilt tokens with "," and "-" and treat each one as different tokens.
- To store these tokens, algorithm uses Hash Map data structure. Hash Map is a useful data structure that has the structure of format <Key, Value>. So each tokens will be stored as a key and value will be the count of the respective token. Whenever token is encountered first time its value in Hash Map is initialized to 1. Next time when the same token encounters it just increases the frequency count of the token.
- For printing top frequency words, It sorts previously created Hash Map by its value(frequency count) in decreasing order and takes top 30 tokens.

1. **How long the program took to acquire the text characteristics.**
   It took 1 second to execute.

2. **How the program handles:**

A. Upper and lower case words (e.g. "People", "people", "Apple", "apple");
Converted each word into lower case before processing so it treat all the tokens as case insensitive.

B. Words with dashes (e.g. "1996-97", "middle-class", "30-year", "tean-ager")
For every token which contains "-", the code splits all the tokens with "-"and treats each split as a separate token.

C. Possessives (e.g. "sheriff's", "university's")
Makes one word from each word before considering it as a token.

D. Acronyms (e.g., "U.S.", "U.N.")
Removed all the "." characters from the word.

Additionally, the program also covers these corner cases:
It removes special characters like: !, @, #, $, %, ^, &, *, (,), [,], {, }, |, \, /, ', +, ; , :, .
For every token which contains "**,** "the code splits all the tokens with "**,** "and treats each split as a separate token.

3. **Major algorithms and data structures.**
   Algorithm uses HashMap to maintain information about the tokens and the frequency in which they appear in the document.

4 Program Output

```
Number of Tokens in the carnfield text collection: 230782

Number of Unique Words in the carnfield text collection: 8932

Number of Words that occur once in the carnfield text collection: 3412

The 30 most frequent words in the Carnfield text collection
Words => Frequency
the=>19449
of=>12714
and=>6671
a=>5974
in=>4650
to=>4560
is=>4113
for=>3491
are=>2428
with=>2263
on=>1943
flow=>1848
at=>1834
by=>1756
that=>1570
an=>1388
be=>1272
pressure=>1206
boundary=>1156
from=>1116
as=>1114
this=>1081
layer=>1002
which=>975
number=>973
results=>885
it=>856
mach=>823
theory=>788
shock=>712

Average number of word tokens per documents: 164

Time required to acquire text characteristic (in sec): 1
```

PROBLEM 2: STEMMING

Implemented by referencing to:
http://chianti.ucsd.edu/svn/csplugins/trunk/soc/layla/WordCloudPlugin/trunk/WordCloud/src/cytoscape/csplugins/wordcloud/Stemmer.java
Uses the tokenizer algorithm as described in problem 1 and then apply stemming on generated tokens.

- **Program Output**

```
Number of distinct stems in the carnfield text collection: 6148

Number of stems that occur once in the carnfield text collection: 2305

The 30 most frequent stems in the Carnfield text collection
Stems => Frequency
the=>19449
of=>12714
and=>6671
a=>5974
in=>4650
to=>4560
is=>4113
for=>3491
ar=>2457
with=>2263
on=>2261
flow=>2079
at=>1834
by=>1756
that=>1570
an=>1388
pressur=>1381
be=>1369
number=>1347
boundari=>1185
layer=>1134
from=>1116
as=>1114
result=>1087
thi=>1081
it=>1043
effect=>996
which=>975
method=>886
theori=>881

Average number of word stems per documents: 164

Time required to acquire text characteristic (in sec): 1
```