**Porter Stemming Algorithm**

The Porter Stemming Algorithm is one of the most common algorithms for stemming English, and the [Snowball](https://en.wikipedia.org/wiki/Snowball_programming_language) programming framework. In the algorithm, R1 is the region after the first non-vowel following a vowel, or the end of the word if there is no such non-vowel. R2 is the region after the first non-vowel following a vowel in R1, or the end of the word if there is no such non-vowel. A short syllable in a word as either (a) a vowel followed by a non-vowel other than w, x or Y and preceded by a non-vowel, or \* (b) a vowel at the beginning of the word which is then followed by a non-vowel. A word is called short if it ends in a short syllable, and if R1 is null.

The following are the rules to be followed while implementing the algorithm:

* An apostrophe (') may be regarded as a letter.
* If the word has two letters or less, leave it as it is.
* Otherwise, Remove initial ', if present. Then, Set initial y, or y after a vowel, to Y, and then establish the regions R1 and R2. Search for the longest among the suffixes, and remove if found.

The Snowball Implementation of the Porter Stemmer Algorithm is as follows:

integers ( p1 p2 )

booleans ( Y\_found )

routines (

prelude postlude

mark\_regions

shortv

R1 R2

Step\_1a Step\_1b Step\_1c Step\_2 Step\_3 Step\_4 Step\_5

exception1

exception2

)

externals ( stem )

groupings ( v v\_WXY valid\_LI )

stringescapes {}

define v 'aeiouy'

define v\_WXY v + 'wxY'

define valid\_LI 'cdeghkmnrt'

define prelude as (

unset Y\_found

do ( ['{'}'] delete)

do ( ['y'] <-'Y' set Y\_found)

do repeat(goto (v ['y']) <-'Y' set Y\_found)

)

define mark\_regions as (

$p1 = limit

$p2 = limit

do(

among (

'gener'

'commun'

'arsen'

) or (gopast v gopast non-v)

setmark p1

gopast v gopast non-v setmark p2

) )

backwardmode (

define shortv as (

( non-v\_WXY v non-v )

or

( non-v v atlimit )

)

define R1 as $p1 <= cursor

define R2 as $p2 <= cursor

define Step\_1a as (

try (

[substring] among (

'{'}' '{'}s' '{'}s{'}'

(delete)

)

)

[substring] among (

'sses' (<-'ss')

'ied' 'ies'

((hop 2 <-'i') or <-'ie')

's' (next gopast v delete)

'us' 'ss'

) )

define Step\_1b as (

[substring] among (

'eed' 'eedly'

(R1 <-'ee')

'ed' 'edly' 'ing' 'ingly'

(

test gopast v delete

test substring among(

'at' 'bl' 'iz'

(<+ 'e')

'bb' 'dd' 'ff' 'gg' 'mm' 'nn' 'pp' 'rr' 'tt'

// ignoring double c, h, j, k, q, v, w, and x

([next] delete)

'' (atmark p1 test shortv <+ 'e')

)

)

)

)

define Step\_1c as (

['y' or 'Y']

non-v not atlimit

<-'i' )

define Step\_2 as (

[substring] R1 among (

'tional' (<-'tion')

'enci' (<-'ence')

'anci' (<-'ance')

'abli' (<-'able')

'entli' (<-'ent')

'izer' 'ization'

(<-'ize')

'ational' 'ation' 'ator'

(<-'ate')

'alism' 'aliti' 'alli'

(<-'al')

'fulness' (<-'ful')

'ousli' 'ousness'

(<-'ous')

'iveness' 'iviti'

(<-'ive')

'biliti' 'bli'

(<-'ble')

'ogi' ('l' <-'og')

'fulli' (<-'ful')

'lessli' (<-'less')

'li' (valid\_LI delete)

)

)

define Step\_3 as (

[substring] R1 among (

'tional' (<- 'tion')

'ational' (<- 'ate')

'alize' (<-'al')

'icate' 'iciti' 'ical'

(<-'ic')

'ful' 'ness'

(delete)

'ative'

(R2 delete)

)

)

define Step\_4 as (

[substring] R2 among (

'al' 'ance' 'ence' 'er' 'ic' 'able' 'ible' 'ant' 'ement'

'ment' 'ent' 'ism' 'ate' 'iti' 'ous' 'ive' 'ize'

'ion'

)

)

define Step\_5 as (

[substring] among (

'e' (R2 or (R1 not shortv) delete)

'l' (R2 'l' delete)

)

define exception2 as (

[substring] atlimit among(

'inning' 'outing' 'canning' 'herring' 'earring'

'proceed' 'exceed' 'succeed'

)

)

)

define exception1 as (

[substring] atlimit among(

/\* special changes: \*/

'skis' (<-'ski')

'skies' (<-'sky')

/\* special -LY cases \*/

'gently' (<-'gentl')

'early' (<-'earli')

'only' (<-'onli')

/\* invariant forms: \*/

'sky'

'news'

'howe'

'atlas' 'cosmos' 'bias' 'andes' // not plural form

)

)

define postlude as (Y\_found repeat(goto (['Y']) <-'y'))

define stem as (

exception1 or

not hop 3 or (

do prelude

do mark\_regions

backwards (

do Step\_1a

exception2 or (

do Step\_1b

do Step\_1c

do Step\_2

do Step\_3

do Step\_4

do Step\_5

) )

do postlude )

)