COP 5536 Fall 2018
Programming Project

Basic Information

- Name : Fatema Saifee
- UFID : 1508 1278
- UF Email account: fatema.saifee@ufl.edu

Function prototypes showing the structure of the programs

1. `class keywordcounter`

`void main(String[] args)`		
Description	Open input and output file stream Process each line from the input file and extract keywords - counts, output required and terminating condition create HashMap for the Keywords and Node referencing to the corresponding frequency in Fibonacci Heap Perform operations like insertNode, removeMax on the Fibonacci Heap Priority Queue	
Parameters	`args[0]`	Name of given input file.
Return value	Void, Write output to output_file.txt containing list of most frequent words for different	

`int getNumberOfLines(String fileName)`		
Description	Calculate the total number of lines in the file	
Parameters	`fileName`	Name of given input file.
Return value	Number of Lines in `fileName`	

2. `class FibonnacciMaxPQ`

`boolean insertNode(Node node)`	
Description	Insert a node in the Fibonacci heap Inserts a node into the root list and checks whether its value is lower than the currently lowest node and changes the access pointer if necessary.

Parameters	`node`	node to be inserted in the FibonnacciMaxPQ
Return value	Number of Lines in `fileName`	

`void increaseKey(int delta, Node node)`		
Description	Insert a node in the Fibonacci heap Inserts a node into the root list and checks whether its value is lower than the currently lowest node and changes the access pointer if necessary.	
Parameters	`node` `delta`	node to be inserted in the FibonnacciMaxPQ The amount to increase the count of the `node`
Return value	Void	

`Node removeMax(int numOfLines)`		
Description	Removes the Node with the maximum count from the Fibonacci heap	
Parameters	`node`	node to be inserted in the FibonnacciMaxPQ
Return value	A node having a maximum value of the frequency	

`boolean merge(Node root)`		
Description	Checks if two nodes in the root list have the same rank. If yes, the node with the higher key is moved into the children list of the other node.	
Parameters	`node`	node to be inserted in the FibonnacciMaxPQ
Return value	true if successful	

3. `class Node`

Since we have an unknown number of children in Fibonacci heaps, we have to arrange the children of a node in a linked list. So, we need at most two pointers to the siblings of every node. This results in a linear double-linked list. Now, we need another pointer to any node of the children list and to the parent of every node. All in all, there are 5 members. Furthermore, we define a rank(order) for every node, which says how many children a node has.

`Node (String key, int count)`		
Description	Constructor initializing key and count members	
Parameters	`key` `count`	Keyword whose frequency is given. Total frequency of the keyword.

Return value	true if successful

`void updateCount(int count)`		
Description	Updates the count of the current node This operation is done in time O(1).	
Parameters	`node`	node to be inserted in the FibonnacciMaxPQ
Return value	Void	

`boolean addChild(Node node)`		
Description	Adds a child node to the current node by inserting it into the children list and linking it. This operation is done in time O(1).	
Parameters	`node`	node to be inserted in the FibonnacciMaxPQ
Return value	true if successful	

`boolean addSibling(Node node)`		
Description	Adds a node into the child list the current node belongs to. This is done in time O(1) too.	
Parameters	`node`	node to be inserted in the FibonnacciMaxPQ
Return value	true if successful	

`boolean cascadingCut()`		
Description	Removes the node from the sibling list and refreshes the affected pointers.	
Parameters		
Return value	true if successful	

`Node rightMostSibling()`		
Description	Traverse the heap to get the rightmost node This is also done in time O(n).	
Parameters		

Return value	The node which is the rightmost sibling of the current node

`Node leftMostSibling()`		
Description	Traverse the heap to get the leftmost node This is also done in time O(n).	
Parameters		
Return value	The node which is the leftmost sibling of the current node	