**CSCI 323 Project 1 (C++)**

Tyler Gaugler

Due Date: 9/7/2016

Algorithm Steps for this project:

Step 0: prepare the skeleton of your program, including classes, etc.

Step 1: open inFile1 // common word file (You may assume there is no duplicate words)

Step 2: CwordlistHead <-- make a new linked list by the list constructor

step 3: commonWord <-- read a word from inFile1

step 4: spot <-- findSpot(commonWord) // see algorithm steps below

step 5: newNode <-- make a new node for commonWord

step 6: call listInsert (Spot, newNode) // you should know how to insert newNode after Spot

step 7: call printList (C CwordlistHead) // output to outfile1

step 8: repeat step 3 to step 7 until the inFile1 is empty

Source Code

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class listNode{

public:

listNode(void);

listNode(string d);

listNode(string d, listNode\* n);

listNode\* next;

string data;

int counter;

private:

};

listNode::listNode(){

data = "";

next = NULL;

counter = 1;

};

listNode::listNode(string d){

data = d;

next = NULL;

counter = 1;

};

listNode::listNode(string d, listNode\* n){

data = d;

next = n;

counter = 1;

};

class linkedList{

public:

linkedList();

listNode \*listHead;

listNode\* findSpot(string d);

void listInsert(listNode \*spot, listNode \*newNode);

string printList();

};

linkedList::linkedList(){

listHead = new listNode();

listNode \*first= new listNode();

first -> data = "dummy";

listHead -> next = first;

};

listNode\* linkedList::findSpot(string d){

listNode \*current = listHead -> next;

while(current->next){

listNode \*nextNode = current -> next;

// cout<< nextNode->data<<endl;

if( nextNode -> data > d ){

return current;

}

else if( current -> data == d){

return nextNode;

}

else{

current = nextNode;

}

}

return current;

};

void linkedList::listInsert(listNode \*spot, listNode \*newNode){

newNode -> next = spot -> next;

spot -> next = newNode;

};

string linkedList::printList(){

listNode \*current = listHead -> next;

string text="";

while(current->next){

listNode \*nextNode = current -> next;

text.append("("+current->data+"," + nextNode->data + ")->");

current= nextNode;

}

text.append("("+current->data +", null)\n");

return text;

};

int main(int argc, char\* argv[])

{

linkedList newList;

ifstream inFile;

ofstream outFile;

inFile.open(argv[1]);

outFile.open(argv[2]);

if(inFile.is\_open()){

string word;

while(inFile >> word){

listNode \*spot = newList.findSpot(word);

// cout<< spot->data <<endl;

if(spot -> data == word){

spot -> counter= 1;

}

else{

listNode \*newNode = new listNode();

newNode -> data = word;

newList.listInsert(spot,newNode);

outFile << "Insert data("+word+")\n";

outFile << newList.printList()+"\n";

}

}

}

inFile.close();

outFile.close();

return 0;

Output

Insert data(you)

(dummy,you)->(you, null)

Insert data(I)

(dummy,I)->(I,you)->(you, null)

Insert data(he)

(dummy,I)->(I,he)->(he,you)->(you, null)

Insert data(she)

(dummy,I)->(I,he)->(he,she)->(she,you)->(you, null)

Insert data(are)

(dummy,I)->(I,are)->(are,he)->(he,she)->(she,you)->(you, null)

Insert data(is)

(dummy,I)->(I,are)->(are,he)->(he,is)->(is,she)->(she,you)->(you, null)

Insert data(was)

(dummy,I)->(I,are)->(are,he)->(he,is)->(is,she)->(she,was)->(was,you)->(you, null)

Insert data(am)

(dummy,I)->(I,am)->(am,are)->(are,he)->(he,is)->(is,she)->(she,was)->(was,you)->(you, null)

Insert data(were)

(dummy,I)->(I,am)->(am,are)->(are,he)->(he,is)->(is,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(of)

(dummy,I)->(I,am)->(am,are)->(are,he)->(he,is)->(is,of)->(of,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(and)

(dummy,I)->(I,am)->(am,and)->(and,are)->(are,he)->(he,is)->(is,of)->(of,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(if)

(dummy,I)->(I,am)->(am,and)->(and,are)->(are,he)->(he,if)->(if,is)->(is,of)->(of,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(or)

(dummy,I)->(I,am)->(am,and)->(and,are)->(are,he)->(he,if)->(if,is)->(is,of)->(of,or)->(or,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(him)

(dummy,I)->(I,am)->(am,and)->(and,are)->(are,he)->(he,him)->(him,if)->(if,is)->(is,of)->(of,or)->(or,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(a)

(dummy,I)->(I,a)->(a,am)->(am,and)->(and,are)->(are,he)->(he,him)->(him,if)->(if,is)->(is,of)->(of,or)->(or,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(an)

(dummy,I)->(I,a)->(a,am)->(am,an)->(an,and)->(and,are)->(are,he)->(he,him)->(him,if)->(if,is)->(is,of)->(of,or)->(or,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(one)

(dummy,I)->(I,a)->(a,am)->(am,an)->(an,and)->(and,are)->(are,he)->(he,him)->(him,if)->(if,is)->(is,of)->(of,one)->(one,or)->(or,she)->(she,was)->(was,were)->(were,you)->(you, null)

Insert data(yes)

(dummy,I)->(I,a)->(a,am)->(am,an)->(an,and)->(and,are)->(are,he)->(he,him)->(him,if)->(if,is)->(is,of)->(of,one)->(one,or)->(or,she)->(she,was)->(was,were)->(were,yes)->(yes,you)->(you, null)

Insert data(no)

(dummy,I)->(I,a)->(a,am)->(am,an)->(an,and)->(and,are)->(are,he)->(he,him)->(him,if)->(if,is)->(is,no)->(no,of)->(of,one)->(one,or)->(or,she)->(she,was)->(was,were)->(were,yes)->(yes,you)->(you, null)

Input

A

you

I he she

are is

was am were

of and if or

him a an one

yes no