```
/*cheatDetector.cpp
 *
 *Dylan Jeffers
 *Tahmid Rahman
 *
 *We implemented large parts of the following code for our
 *CS31 class during Fall of 2014 at Swarthmore
 *
 *We changed our code to take into account our SPLAVL Tree and Splay Tree
 *
 *The cout statements in main help gather data piped to an output file
 *
 *We used cheatDetector to test how our SPLAVL tree compares to both
 *Splay and AVL Trees
 *
 *cheatDetector implementation - compares documents to each
 *other and outputs how many phrases of a given length match
 */

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "pair.h"
#include "SPLAVL.h"
#include "SplayTree.h"
#include "AVLTree.h"
#include "library/circularArrayList.h"
#include <time.h>

using namespace std;

BST<string, int>* storeDoc(string fileName, int useAVL, int maxR,
      int maxC, int phraseSize);
List < Pair<string, BST<string, int>* > >*
loadDocs(string flist, int useAVL, int maxR, int maxC, int phraseSize);
void cleanUp(List < Pair<string, BST<string, int>* > >* docList);
void bestCompare(List < Pair<string, BST<string, int>* > >* docList);
int compare(BST<string, int>* doc1, BST<string, int>* doc2);


int main(int argc, char* argv[]){
  if(argc != 4){
    cerr << "Incorrect number of arguments" << endl;
    cerr << "Usage: cheatDetector file-list phrase-size useAVL" << endl;
    return 1;
  }

  int phraseSize = atoi(argv[2]);
  int useAVL = atoi(argv[3]);
  string flist(argv[1]);

  if (phraseSize <= 0){
      cerr << "Incorrect number for phrase-size" << endl;
      cerr << "Phrase size must be greater than 0" << endl;
      return 1;
  }

  if (useAVL == 0) {
```

```
    int maxR = 40; //For our tests, we changed maxR

    cout << "For maxR = " << maxR << endl;

        cout << "Order is: " << endl;

        cout << "maxC Total-time Load-time - Compare-time" << endl;

          for (int maxC = 1; maxC < 300; maxC=maxC+50){

                  List < Pair<string, BST<string, int>* > >* docList;

                  clock_t t1, t2, t3;

              t1 = clock();
                  docList = loadDocs(flist, useAVL, maxR, maxC, phraseSize);

              t2 = clock();
                  bestCompare(docList);

                  t3 = clock();

             cout << maxC << " ";
                  cout << ((float)(t3-t1))/CLOCKS_PER_SEC << " ";
                  cout << ((float)(t2-t1))/CLOCKS_PER_SEC << " ";
                  cout << ((float)(t3-t2))/CLOCKS_PER_SEC << " ";
                  cout << endl;

                  cleanUp(docList);
                  delete docList;
                }
}
else{

      cout << "Option: " << useAVL << endl;

      cout << "Order is: " << endl;

      cout << "Total-time Load-time - Compare-time" << endl;

      int maxR, maxC;
      maxR = maxC = 0;

      List < Pair<string, BST<string, int>* > >* docList;

      clock_t t1, t2, t3;

    t1 = clock();
      docList = loadDocs(flist, useAVL, maxR, maxC, phraseSize);

    t2 = clock();
      bestCompare(docList);

      t3 = clock();

      cout << ((float)(t3-t1))/CLOCKS_PER_SEC << " ";
      cout << ((float)(t2-t1))/CLOCKS_PER_SEC << " ";
      cout << ((float)(t3-t2))/CLOCKS_PER_SEC << " ";
      cout << endl;
```

```
        cleanUp(docList);
        delete docList;
  }

  return 0;
}
```

```
/*storeDoc
 *inputs: name of file, useAVL, phraseSize
 *opens a single document and loads it into a binary tree
 *returns binary tree version of document
 */
BST<string, int>* storeDoc(string fileName, int useAVL, int maxR,
      int maxC, int phraseSize) {
      BST<string, int>* document;
      if (useAVL == 0){
            document = new SPLAVL<string, int>(maxC, maxR);
      }
      else if (useAVL == 1){
            document = new SplayTree<string, int>();
      }
      else {
            document = new AVLTree<string, int>();
      }
      /*Loading algorithm:
       *1. Upload first n words determined by phraseSize
       *     into a Circurlar Array List.
       *
       *2. Concatenate words into single phrase and
       *     upload into BST
       *
       *3. Remove first word from phrase, inserts next word into phrase.
       *
       *4. Continues uploading phrases into BST until end of file.
       */
      List<string>* words = new CircularArrayList<string>();
      ifstream inFile;
      string word;
      string phrase;

      inFile.open(fileName.c_str());
      if (!inFile.is_open()) {
            cerr << "Specific file did not open." << endl;
            delete words;
            return document;
      }
      for (int i = 0; i < phraseSize; i++) {
            inFile >> word;
            words->insertAtTail(word);
            phrase = phrase + words->peekTail() + " ";
      }
```

```
      document->insert(phrase, 1);

      inFile >> word;
      while(!((inFile.eof()))){
            words->removeHead();
            phrase = "";
            words->insertAtTail(word);

            for (int i = 0; i < phraseSize; i++) {
                  phrase = phrase + words->get(i) + " ";
            }

            if (!document->contains(phrase)) {
                  document->insert(phrase, 1);
            }
            else {
                  int value = document->find(phrase);
                  document->update(phrase, value+1);
            }

            inFile >> word;
      }

      delete words;
      return document;
}




/*loadDocs
 *inputs: name of file containing list of documents,
 *        useAVL, phraseSize
 *opens each document file and stores it as a BST.
 *return: list of all documents.
 */
List < Pair<string, BST<string, int>* > >*
loadDocs(string flist, int useAVL, int maxR, int maxC, int phraseSize){
      ifstream inFile;
    string fileName;
    Pair< string, BST<string, int>* > currentDoc;
    List < Pair<string, BST<string, int>* > >* docList
      = new CircularArrayList< Pair<string, BST<string, int>* > >();

      inFile.open(flist.c_str());
      if (!inFile.is_open()) {
            cerr << "Unable to open file!" << endl;
            return docList;
      }

      inFile >> fileName;
      while(!inFile.eof()){
            currentDoc.first = fileName;
            currentDoc.second = storeDoc(fileName, useAVL, maxR, maxC, phraseSize);
            docList->insertAtTail(currentDoc);
            inFile >> fileName;
      }
```

```
	return docList;
}




/*cleanUp
 *inputs: list of pairs where each pair is <name of file, BST>
 *ensures all BSTs strored as second element in pair get deleted.
 */
void cleanUp(List < Pair<string, BST<string, int>* > >* docList){
	BST<string, int>* BSTtoDelete;

	for (int i = 0; i < docList->getSize(); i++){
		BSTtoDelete = docList->get(i).second;
		delete BSTtoDelete;
	}
}






/*compare
 *inputs: Two BSTs representing two documents
 *compares the keys of one BST to the other, counting the number
 *of matches by referrencing the values of each node.
 *return: total number of matches.
 */
int compare(BST<string, int>* doc1, BST<string, int>* doc2){
	Queue< Pair<string, int> >* phrasesToCompare;
	Pair<string, int> currentPair;
	int doc1Value, doc2Value;
	int matches = 0;

	phrasesToCompare = doc1->getInOrder();
	while(!phrasesToCompare->isEmpty()){
		currentPair = phrasesToCompare->dequeue();

		if (doc2->contains(currentPair.first)){
			doc1Value = currentPair.second;
			doc2Value = doc2->find(currentPair.first);
			if (doc2Value <= doc1Value){
				matches = matches + doc2Value;
			}
			else{
				matches = matches + doc1Value;
			}
		}

	}

	delete phrasesToCompare;
	return matches;
```

```
}



/*bestCompare
 *inputs: List of pairs where each pair is <filename,BST>
 *calls compare between each document, and
 *for each document i,
 *    print name of document j with highest number of matches
 *print max height of all document BSTs
 *also print average height,size, and size to height ratio of all documents.
 */
void bestCompare(List < Pair<string, BST<string, int>* > >* docList){
      BST<string, int>* doc1;
      BST<string, int>* doc2;
      int numDoc = docList->getSize();
      string bestFile = "";
      int bestMatch, currentMatch;
      float currentSize;
      int maxHeight, currentHeight;
      float totalHeight, totalSize, ratio, totalRatio;


      maxHeight = totalHeight = totalSize = currentSize =
      totalRatio = ratio = 0;
      for (int i = 0; i < numDoc; i++){
            doc1 = docList->get(i).second;
            currentHeight = doc1->getHeight();

            currentSize = doc1->getSize();

            if (currentSize != 0){

                  if (currentHeight == 0){
                        ratio = currentSize/1;
                  }
                  else{
                        ratio = currentSize/currentHeight;
                  }

                  totalRatio = totalRatio + ratio;

                  totalSize = totalSize + currentSize;
                  totalHeight = totalHeight + currentHeight;

                  bestMatch = 0;

                  for (int j = 1; j < numDoc; j++){
                        currentMatch = 0;
                        doc2 = docList->get((i+j)%(numDoc)).second;

                        if(currentHeight > maxHeight){
                              maxHeight = doc1->getHeight();
                        }
```

```
                        currentMatch = compare(doc1, doc2);
                        if (currentMatch >= bestMatch){
                         bestMatch = currentMatch;
                           bestFile = docList->get((i+j)%(numDoc)).first;
                        }
                    }
                }
            }
        }
```