```
 1
 2 /* given an orthoPhrase, returns all possible orthoPhrases it could be misheard as*/
 3 vector<string> discoverOronymsForPhrase( string origOrthoPhrase ) {
 4    vector<string> orthoMisheardAsPhrases;
 5
 6    vector<vector<phone> > allPhoneSeqsOfOrigPhrase = findAllPhoneSeqsForOrthoPhrase( origOrthoPhrase
 7
 8    int numUniquePhoneticInterpretations = allPhoneSeqsOfOrigPhrase.size();
 9    for(int i = 0; i < numUniquePhoneticInterpretations; i++) {
10       vector<phone> curPhoneSeq( allPhoneSeqsOfOrigPhrase.at(i) );
11       string strOfCurPhoneSeq = phoneVectToString( curPhoneSeq );
12       cerr << "Phonetic interpretation "<<i<<" ("<< strOfCurPhoneSeq <<")"<<endl;
13       vector<string> altOrthoPhrases = interpretPhrase( curPhoneSeq );
14       for( int j = 0; j < altOrthoPhrases.size(); j++) {
15          cerr << i <<"~~>" << altOrthoPhrases.at(j) << endl;
16          //TODO change so it only shows fully valid strings
17          orthoMisheardAsPhrases.push_back( altOrthoPhrases.at(j) );
18       }
19    }
20    //TODO deduplicate orthoMisheardAsPhrases
21    return orthoMisheardAsPhrases;
22 }
23 /*This function does the phoneme-tree-traversal thing for oronyms
24    returns orthographic phrases (I *think* each string is a full phrase...)*/
25 vector<string> interpretPhrase( vector<phone> sampaPhraseOrig ) {
26    vector<phone> sampaPhrase = getNoEmphsPhoneVect(sampaPhraseOrig);
27    vector<string> misheardOrthoPhrases;
28    //assert(0);
29    cerr << "INTERPRET PHRASE for " << phoneVectToString(sampaPhrase) << endl;
30    if( sampaPhrase.size() == 0 ) {
31       misheardOrthoPhrases.push_back("");
32       return misheardOrthoPhrases;
33    }
34
35    string sampaStr = "";
36    vector <phone> usedPhones;
37
38    for (int i = 0; i < sampaPhrase.size(); i++) {
```

```
39        phone p = sampaPhrase[i];
40        if( strcmp( "\"", p.c_str() ) == 0) {
41           assert(0);
42           continue; //TODO incorporate someday, but ignore emphases for now.
43        } else if ( strcmp( "$", p.c_str() ) == 0) {
44           assert(0);
45           continue; //TODO incorporate someday, but ignore emphases for now.
46        } else if ( strcmp( "%", p.c_str() ) == 0) {
47           assert(0);
48           continue; //TODO incorporate someday, but ignore emphases for now
49        }
50        sampaStr += p;
51        usedPhones.push_back(p);
52        vector<string> orthoMatches = queryDBwithSampaForOrthoStrs( sampaStr );
53        //if there are no exact matches
54        if ( orthoMatches.size() == 0 ) {
55           vector<string> prefixMatches = queryDBForOrthoStrsWithSampaPrefix( sampaStr );
56           //if there are no partial matches, we have a dead end, so exit
57           if( prefixMatches.size() == 0 ) {
58              misheardOrthoPhrases.push_back("DEADBEEF");
59              //TODO might have to delete rest of phone seq? we'll see.
60              continue;
61           } else {
62              continue; //go to next loop iter and add next phone
63           }
64           //return misheardOrthoPhrases;
65           cerr <<" OLD RETURN STATEMENT WAS HERE for if no exact matches"<< endl;
66        }
67
68        for (int j = 0; j < orthoMatches.size(); j++) {
69           string orthoWord = orthoMatches[j];
70           vector<phone> sampaPhraseTail( sampaPhrase.begin(), sampaPhrase.begin() + i );
71           vector<string> orthoLeaves = interpretPhrase ( sampaPhraseTail );
72           if ( orthoLeaves.size() == 0 ) {
73              if( sampaPhraseTail.size() > 0 ) {
74                 cerr<< "--"<<orthoWord<<"---no leaves, has tail: "<< phoneVectToString(sampaPhraseTail
75                 //TODO RESTART TRACE AT NEXT LINE!perhaps want a continue?
76                 misheardOrthoPhrases.push_back( orthoWord.append( "DEADBEEF" ) );
```

```
77              }
78              //return misheardOrthoPhrases;
79              cerr <<" OLD RETURN STATEMENT WAS HERE for if no ortholeaves"<< endl;
80              continue;
81           } else {
82
83              for (int k = 0; k < orthoLeaves.size(); k++) {
84                 string orthoLeaf = orthoLeaves[k];
85                 misheardOrthoPhrases.push_back( orthoWord + orthoLeaf );
86              }
87
88           }
89        }
90     }
91     return misheardOrthoPhrases;
92  }
93
```