```
#include "wordBreakdown.h"

using namespace std;

//GLOBALS for database access
sqlite3 *db;
vector< vector< string > > databaseResults;

string deadEndDelim1 = "xxx"; //formerly "DEADBEEF"
string deadEndDelim2 = "fff"; //formerly "DEADerBEEF"

static int callback(void *queryterm, int nCol, char **values, char **headers){
   int i;
   vector<string> rowEntry;
   //fprintf(stderr,"===Callback for query %s===\n",(char *) queryterm);
   for(i=0; i< nCol; i++){
      //fprintf(stderr,"%s = %s\n", headers[i], values[i] ? values[i] : "NULL");
      rowEntry.push_back( values[i] );
   }
   //printf("\n");
   databaseResults.push_back(rowEntry);
   return 0;
}

vector< vector<phone> > getPhoneSeqsForSampaStrs( vector<string> sampaStrings ) {
   vector< vector <phone> > sampaSyllPhrases;
   for(int i = 0; i < sampaStrings.size(); i++) {
      vector<phone> sampaSylls = parseSAMPAintoPhonemes( sampaStrings[i] );
      sampaSyllPhrases.push_back(sampaSylls);
    }
   return sampaSyllPhrases;
}

vector< vector<phone> > getPhoneSeqsForOrthoWord( string orthoWord ) {
   vector<string> sampaStrings = queryDBwithOrthoForSampaStrs( orthoWord );
   return getPhoneSeqsForSampaStrs( sampaStrings );
}

```

```
/*given an phrase of ortho words, gives all the sampa permutations
 that it could possibly be, where each inner vector is a phonetic permutation of
 the full phrase */
vector< vector<phone> > findAllPhoneSeqsForOrthoPhrase( string orthoPhrase ) {

   vector<string> orthoWords = strTokOnWhitespace( orthoPhrase );
   vector< vector<phone> > fullPhrasePhoneSeqs;
   //cerr << "FIND ALL PERMUTATIONS" << endl;
   for (int i = 0; i < orthoWords.size(); i++) {
      vector<string> sampaStrings = queryDBwithOrthoForSampaStrs( orthoWords[i] );
      vector< vector<phone> > nextWordSAMPAPhoneSeqs = getPhoneSeqsForOrthoWord( orthoWords[i] );
         /*
         //DEBUG
         cerr << j <<": ";
         for ( int k = 0; k < nextWordSAMPAPhoneSeqs[j].size(); k++ ) {
            cerr<< "_" << nextWordSAMPAPhoneSeqs[j][k] << "_";
         }
         cerr << endl;
         //END DEBUG
         */
         if( nextWordSAMPAPhoneSeqs.empty() ) {
            cout<< "The word '"<<orthoWords[i]<<"was not found in our phonetic dictionary";
            cout<< "Enter a different word now, or watch this program crash and burn: ";
            string temp;
            cin >> temp;

            nextWordSAMPAPhoneSeqs = getPhoneSeqsForOrthoWord( temp );
         }

         //if this is ths first orthoWord
         if( i == 0 ) {
            for( int j = 0; j < nextWordSAMPAPhoneSeqs.size(); j++ ) {
               fullPhrasePhoneSeqs.push_back( nextWordSAMPAPhoneSeqs[j] );
            }
         } else {
            int numFullPhrases = fullPhrasePhoneSeqs.size();
            //cerr << "\tnumFullPhrases = "<< numFullPhrases << endl;//TODO debug
            if ( nextWordSAMPAPhoneSeqs.size() > 1 ) {
               //cerr << "\tnextWordSAMPAPhoneSeqs.size() = "<< nextWordSAMPAPhoneSeqs.size() << endl
               for(int m = 1; m < nextWordSAMPAPhoneSeqs.size(); m++) {
                  //if there's more than one phonetic interpretation of the
                  // ortho word to be added, then we need to create duplicates
                  // of all existing sampaPhrase entries for each of them.
                  for( int n = 0; n < numFullPhrases; n++){
                     vector< phone > copyOfFullPhraseN( fullPhrasePhoneSeqs[n] );
                     fullPhrasePhoneSeqs.push_back( copyOfFullPhraseN );
                  }
               }
            }
            for( int m = 0; m < fullPhrasePhoneSeqs.size(); m++){
               int phrsToAppendNdx = m / numFullPhrases;
               vector<phone> phraseToAppend( nextWordSAMPAPhoneSeqs[phrsToAppendNdx] );
               fullPhrasePhoneSeqs[m].insert( fullPhrasePhoneSeqs[m].end(),
                                       phraseToAppend.begin(),
                                       phraseToAppend.end() );
            }
                  //DEBUG
                  for ( int e = 0; e < fullPhrasePhoneSeqs.size(); e++ ) {
                     cerr << e <<"***sampa phrase after append  ";
                     for ( int f = 0; f < fullPhrasePhoneSeqs[e].size(); f++ ) {
                        cerr<< "_" << fullPhrasePhoneSeqs[e][f] << "_";
                     }
                     cerr << endl;
                  }
                  //END DEBUG
         }
         /*
         //DEBUG
         cerr << j <<"++SAMPA+PHRASES++  ";
         for ( int k = 0; k < fullPhrasePhoneSeqs[j].size(); k++ ) {
            cerr<< "-" << fullPhrasePhoneSeqs[j][k] << "-";
         }
         cerr << endl;
         //END DEBUG
      */

```

```
   }
        // assert(0);

/*
   vector<string> misheard;
   for (int i = 0; i < fullPhrasePhoneSeqs.size(); i++){
       //misheard.push_back( interpretPhrase( fullPhrasePhoneSeqs[i] ) )
   }

   for (int i = 0; i < misheard.size(); i++) {
      string s = misheard[i];
      DDDDDDDDDDDEBUG(s);
   }


   return misheard;
*/



   return fullPhrasePhoneSeqs;
}


/*given an phrase of ortho words, gives all the sampa permutations
 that it could possibly be, where each inner vector represents all the
 possible phonetic interpretations for each phoneme. For example:
 Given: a nice
 Returns:   vector[0]: { e, @, A }
            vector[1]: { n }
            vector[2]: { aI, i }
            vector[3]: { s }
             */
 vector< set<phone> > findPhoneTreeForOrthoPhrase( string orthoPhrase ) {
   vector< set<phone> > phoneTree;
   cerr <<"findPhoneTreeForOrthoPhrase is broken! ("<<orthoPhrase<<")"<<endl;
   assert(0);
   //using set because it doesn't allow for duplicates
   vector<string> orthoWords = strTokOnWhitespace( orthoPhrase );
   vector< vector<phone> > oldFullPhrasePhoneSeqs = findAllPhoneSeqsForOrthoPhrase( orthoPhrase );
   vector< vector<phone> > fullPhrasePhoneSeqs;

   if( oldFullPhrasePhoneSeqs.size() > 0 )   {
      for( int i = 0; i < oldFullPhrasePhoneSeqs.size(); i++) {
         //strip all the phoneSeqs of emph values
         fullPhrasePhoneSeqs.push_back( getNoEmphsPhoneVect( oldFullPhrasePhoneSeqs.at(i) ) );
      }

      for(int i = 0; i < fullPhrasePhoneSeqs[0].size(); i++) {

         set<phone> dummySet;

         for( int j = 0; j < fullPhrasePhoneSeqs.size(); j++) {
            //I assume that all phone seqs will be equal length. If not, assert.
            if( fullPhrasePhoneSeqs[j].size() != fullPhrasePhoneSeqs[0].size() ){
               cerr << "0-size="<< fullPhrasePhoneSeqs[0].size()<< endl;
               cerr << j<<"-size="<< fullPhrasePhoneSeqs[j].size()<< endl;
               cerr << j<<"="<< phoneVectToString( fullPhrasePhoneSeqs[j] )<< endl;

               assert(0);
            }
            //put the i-th phone from each fullPhrasePhoneSeq into the i-th phoneTree set.
            int deleteme1 = dummySet.size();
            phone toAdd = fullPhrasePhoneSeqs[j][i];
            cerr << "dummySet.size()="<<deleteme1<<",  toAdd="<<toAdd<<endl;
            dummySet.insert( toAdd );
         }
         phoneTree.push_back( dummySet );
      }
   }




   //DEBUG
   for ( int j = 0; j < phoneTree.size(); j++) {
      cerr << j <<"++PHONE+TREE++  ";
      vector<phone> temp( phoneTree[j].begin(), phoneTree[j].end() );

      for ( int k = 0; k < temp.size(); k++ ) {

         cerr<< "-" << temp.at(k) << "-";
      }
      cerr << endl;
   }
   //END DEBUG

   return phoneTree;
}


/*given an orthoPhrase, returns all possible orthoPhrases it could be misheard as*/
vector<string> discoverOronymsForPhrase( string origOrthoPhrase ) {
   vector<string> orthoMisheardAsPhrases;
   vector<vector<phone> > allPhoneSeqsOfOrigPhrase = findAllPhoneSeqsForOrthoPhrase( origOrthoPhrase

   int numUniquePhoneticInterpretations = allPhoneSeqsOfOrigPhrase.size();
   for(int i = 0; i < numUniquePhoneticInterpretations; i++) {
      vector<phone> curPhoneSeqWithEmph( allPhoneSeqsOfOrigPhrase.at(i) );
      string strOfCurPhoneSeq = phoneVectToString( curPhoneSeqWithEmph );

      cerr << "Phonetic interpretation "<<i<<" ("<< strOfCurPhoneSeq <<")"<<endl;

      //remove emphasis marking for easier lookups
      vector<phone> curPhoneSeq = getNoEmphsPhoneVect( curPhoneSeqWithEmph );

      //vector<string> altOrthoPhrases = interpretPhrase( curPhoneSeq );
      vector<string> altOrthoPhrases = findOrthoStrsForPhoneSeq( curPhoneSeq );

      cerr << "exits findOrthoStrsForPhoneSeq"<<endl;
      for( int j = 0; j < altOrthoPhrases.size(); j++) {
         string altOrthoPhrase = altOrthoPhrases.at(j);
         cerr << i <<"~~>" << altOrthoPhrase << endl;

```

```cpp
         //ensure it contains no deadEndDelims so we only add fully valid strings
         if ( altOrthoPhrase.find( deadEndDelim1 ) == string::npos
              && altOrthoPhrase.find( deadEndDelim2 ) == string::npos ) {
            orthoMisheardAsPhrases.push_back( altOrthoPhrase );
         }
      }
   }
   //deduplicate orthoMisheardAsPhrases by putting in a set and back again
   cerr << "DEDUPLICATION TIME!" <<endl;
   set<string> tempSetForDeduplication( orthoMisheardAsPhrases.begin(), orthoMisheardAsPhrases.end()
   orthoMisheardAsPhrases.assign( tempSetForDeduplication.begin(), tempSetForDeduplication.end() );
   return orthoMisheardAsPhrases;
}

/*This function does the phoneme-tree-traversal thing for oronyms
   returns orthographic phrases (I *think* each string is a full phrase...)*/
vector<string> interpretPhrase( vector<phone> sampaPhraseOrig ) {
   vector<phone> sampaPhrase = getNoEmphsPhoneVect(sampaPhraseOrig);
   vector<string> misheardOrthoPhrases;
   assert(0);
   /*
   cerr << "INTERPRET PHRASE for " << phoneVectToString(sampaPhrase) << endl;
   if( sampaPhrase.size() == 0 ) {
      misheardOrthoPhrases.push_back("");
      cerr << "<<<<<<phraseSize == 0, so returning all of the phrases" <<endl;
      return misheardOrthoPhrases;
   }

   string sampaStr = "";
   vector <phone> usedPhones;
   cerr << "sampaPhrase.size() ="<<sampaPhrase.size()<<endl;
   for (int i = 0; i < sampaPhrase.size(); i++) {
      cerr << "i = "<<i<<"; sampaPhrase[i]= phone p = "<<sampaPhrase[i]<<endl;
      phone p = sampaPhrase[i];
      if( strcmp( "\"", p.c_str() ) == 0) {
         assert(0);
         continue; //TODO incorporate someday, but ignore emphases for now.
      } else if ( strcmp( "$", p.c_str() ) == 0) {
         assert(0);
         continue; //TODO incorporate someday, but ignore emphases for now.
      } else if ( strcmp( "%", p.c_str() ) == 0) {
         assert(0);
         continue; //TODO incorporate someday, but ignore emphases for now
      }
      sampaStr += p;
      cerr<< "Sampastr = "<<sampaStr<<endl;
      usedPhones.push_back(p);
      vector<string> orthoMatches = queryDBwithSampaForOrthoStrs( sampaStr );
      cerr << "orthoMatches.size() =="<<orthoMatches.size()<<endl;
      //DEBUG
      for(int o = 0; o < orthoMatches.size(); o++) {
         cerr<<"++"<<orthoMatches.at(o);
      }
      cerr<<endl;
      //END DEBUG
      //if there are no exact matches
      if ( orthoMatches.size() == 0 ) {
         vector<string> prefixMatches = queryDBForOrthoStrsWithSampaPrefix( sampaStr );
         //if there are no partial matches, we have a dead end, so exit
         if( prefixMatches.size() == 0 ) {
            misheardOrthoPhrases.push_back( deadEndDelim1 );
            //TODO might have to delete rest of phone seq? we'll see.
            continue;
         } else {
            continue; //go to next loop iter and add next phone
         }
         //return misheardOrthoPhrases;
         cerr <<" OLD RETURN STATEMENT WAS HERE for if no exact matches"<< endl;
      }

      for (int j = 0; j < orthoMatches.size(); j++) {
      cerr << "enter orthomatches loop"<<endl;
         string orthoWord = orthoMatches[j];
         cerr << "----"<<i<<"--orthoword--"<< orthoMatches[j] << endl;
         vector<phone> sampaPhraseTail( sampaPhrase.begin()+j+1, sampaPhrase.end() );
         cerr << "----"<<i<<"--sampaPhraseTail--"<< phoneVectToString(sampaPhraseTail) <<"|--"<< endl

         vector<string> orthoLeaves = interpretPhrase ( sampaPhraseTail );
         if ( orthoLeaves.size() == 0 ) {
            if( sampaPhraseTail.size() > 0 ) {
               cerr<< "--"<<orthoWord<<"---no leaves, has tail: "<< phoneVectToString(sampaPhraseTai
               //TODO RESTART TRACE AT NEXT LINE!perhaps want a continue?
               misheardOrthoPhrases.push_back( orthoWord.append(  deadEndDelim1  ) );
            }
            //return misheardOrthoPhrases;
            cerr <<" OLD RETURN STATEMENT WAS HERE for if no ortholeaves"<< endl;
            continue;
         } else {

            for (int k = 0; k < orthoLeaves.size(); k++) {
               string orthoLeaf = orthoLeaves[k];
               misheardOrthoPhrases.push_back( orthoWord + orthoLeaf );
            }

         }
      }
   }

   cerr<<"EXITING interpretPhrase"<<endl;
   return misheardOrthoPhrases;
   */
}

vector<string> findOrthoStrsForPhoneSeq( vector<phone> phoneSeq ) {
   //cerr<<"+++findOrthoStrsForPhoneSeq, for "<< phoneVectToString( phoneSeq );
   //cerr<<",  size = "<<phoneSeq.size()<<endl;
   vector<phone> usedPhonesForOrtho;

   vector<phone> curPhoneSeq;

   vector< string > fullOrthoStrs;
   if( phoneSeq.size() == 0 ) {
      //cerr<<"+++ PHONE SEQ SIZE = 0, we're SUCCEEEDED! WOOOHOOO!"<<endl;
      fullOrthoStrs.push_back("___SUCCESS!___");
```

```
        return fullOrthoStrs;
    }
    for ( int i = 0; i < phoneSeq.size(); i++) {
        phone p = phoneSeq.at(i);
        curPhoneSeq.push_back(p);
        string curPhoneSeqStr = phoneVectToString( curPhoneSeq );

        //cerr<<"+++"<<"+++p"<<i<<":"<<p<<"  full subseq = "<<curPhoneSeqStr<<"."<<endl;

        /////STEP 1: EXACT MATCHES
        //Query for exact ortho matches of the curPhoneSeq
        vector<string> orthoInterps = queryDBwithSampaForOrthoStrs(curPhoneSeqStr);

        //cerr<<"+++"<<"+++orthoInterps.size() = "<<orthoInterps.size()<<endl;

        //If there is one or more exact ortho match for the phoneSeq
        if( orthoInterps.size() > 0 ) {

            //Since we're using the phones in curPhoneSeq for our ortho matches,
            //we don't want to re-look-up those phonemes. the line below
            // removes all the phones we used in curPhoneSeq from the phoneSeq to
            // make newPhoneSeqTail;
            vector<phone> newPhoneSeqTail( phoneSeq.begin() + i + 1 , phoneSeq.end() );
            //cerr<<"+++++++++newPhoneSeqTail is "<< phoneVectToString( newPhoneSeqTail )<<endl;

            //findOrthoStrings for the tail phonemes
            vector<string> tailOrthoStrs = findOrthoStrsForPhoneSeq(newPhoneSeqTail);

            //for each orthoInterpretation of the curPhoneSeq,
            for(  int j = 0; j < orthoInterps.size(); j++ ) {
                for ( int k = 0; k < tailOrthoStrs.size(); k++ ) {
                    string headPlusTailOrtho = orthoInterps.at(j) + " "+ tailOrthoStrs.at(k);
                    fullOrthoStrs.push_back( headPlusTailOrtho );
                    cerr<<"+++"<<"+++"<<"+++"<<headPlusTailOrtho<<endl;
                }
            }
        } else if ( i == phoneSeq.size() - 1 ) {
            //then there are no phonemes left after this one.
```

```
         //it would be stupid to check for partials if there's nothing to append
         //so we DEADBEEF THAT SHIT

         fullOrthoStrs.push_back( deadEndDelim1 );


      } else {
         //STEP 2: PARTIAL PREFIX MATCHES

         // query for ortho matches that have the current phoneSeq as a prefix
         vector<string> orthoPartials = queryDBForOrthoStrsWithSampaPrefix(curPhoneSeqStr);

         //if there are partial matches
         if( orthoPartials.size() > 0 ) {
            continue;
         } else {
            // there are no partial matches even.
            // How should I denote thsi?
            fullOrthoStrs.push_back( deadEndDelim2 );
            break;
         }
      }
   }
   return fullOrthoStrs;
}

vector<string> queryDBforStrings( char* sqlQuery, string queryCallback4thArg ) {
   char* zErr;
   /*The following line calls the callback function, passing its 4th arg as the
    first param of the callback function.  The sqlite3_exec function
    queries the database, then for every result that it gets, it calls the
    callback function.*/
   int rc = sqlite3_exec(db, sqlQuery, callback, (void*)queryCallback4thArg.c_str(), &zErr);
   if ( rc != SQLITE_OK ) {
      if ( zErr != NULL ) {
         fprintf(stderr, "SQL error: %s\n", zErr);
         sqlite3_free(zErr);
      }
```