```cpp
#include "SymTable.h"
#include "CoinTrial.h"
#include "Variable.h"
#include "prob_tables_coin.h"
#include <iostream>
#include <fstream>
#include <algorithm>
#include <math.h>
using namespace std;

Variable chce; /*!< \brief encapsulates the 'disk' being used to choose a type of coin.
     contains symbol table for its possible outcomes (eg. A vs B) */
Variable ht;   /*!< \brief encapsulates a heads-vs-tails coin toss.
         contain symbol table for its possible outcomes (eg. H vs T) */

vector<CoinTrial> data; /*!< \brief represents all the data
                             as a vector of CoinTrial objects */

void process_corpus(string file); /*!< \brief turn contents of filename into data
                                   * each line of the file is represented by a CoinTrial object */

//! splits a line into into tokens using white-space as separator
void tokenize(string line, vector<string> &words);

int main(int argc, char **argv)
{

    string filename;
    filename = string(argv[1]);

    // after this call the vector data corresponds to the contents of filename
    // each line of the file is represented by a CoinTrial object
    process_corpus(filename);

    cout << "read all data\n";
    cout << "total amount of extracted data is: " << data.size() << endl;

    // just show the data
    for (unsigned int d = 0; d < data.size(); d++)
    {
        data[d].show();
    }

    // hard wire some probs
    chce_probs[0] = 0.2;
    chce_probs[1] = 0.8;
    ht_probs[0][0] = 0.4;
    ht_probs[0][1] = 0.6;
    ht_probs[1][0] = 0.3;
    ht_probs[1][1] = 0.7;

    vector<vector<double> > gamma;
    /* purpose of gamma[d][z] is that it should be cond prob of (chce=z) given
    the visible coin toss outcomes of the d-th data item */

    // this just makes into a table of the right size
    gamma.resize(data.size());
    for (int dn = 0; dn < data.size(); dn++)
    {
        gamma[dn].resize(2);
    }

    // BEGIN INSERT: at present all gamma's entries are 0
    // insert code here to set the content of gamma based on data
    // and the probs in chce_probs and ht_probs so that gamma[d][z] does
    // give cond prob (chce=z) given the visible coin toss outcomes of
    // the d-th data item
    // feel free to add additional helper functions to this file also
    // note that can definitely complete this *without* modifying any other files
    for (int dn = 0; dn < data.size(); dn++)
    {
        int heads = 0;
        int tails = 0;
        for (int i = 0; i < data[dn].outcomes.size(); i++) {
            if (data[dn].outcomes[i] == 0) heads++;
            else tails++;
        }
        double pa = chce_probs[0] * pow(ht_probs[0][0], heads) * pow(ht_probs[0][1], tails);
        double pb = chce_probs[1] * pow(ht_probs[1][0], heads) * pow(ht_probs[1][1], tails);
        double sum = pa + pb;
        gamma[dn][0] = pa / sum;
        gamma[dn][1] = pb / sum;
    }
    // END INSERT
    // show gamma
    for (int dn = 0; dn < data.size(); dn++)
    {
        cout << dn + 1 << ": ";
        for (int z = 0; z < 2; z++)
        {
            cout << chce.table.decode_to_symbol(z) << "(" << gamma[dn][z] << ")   ";
        }
        cout << endl;
    }
}

void process_corpus(string afile)
{

    ifstream f;
    f.open(afile.c_str());
    if (!f)
    {
        cout << "prob opening " << afile << endl;
        exit(1);
    }
    else
    {
        cout << "processing " << afile << endl;
    }

    vector<string> raw_line;
    CoinTrial line_rep;
    string line = "";

    while (getline(f, line))
    {
        vector<string> pre_words;
        tokenize(line, raw_line);
        line_rep.outcomes.clear();
        // make line_rep from raw_line
        // then push to data
        string word;
        for (unsigned int i = 0; i < raw_line.size(); i++)
        {
            word = raw_line[i];
            if (i == 0)
            {
                line_rep.coin_choice = (chce.table.get_code(word));
            }
            else
            {
                line_rep.outcomes.push_back(ht.table.get_code(word));
            }
        }

        data.push_back(line_rep);
    }

    for (unsigned int d = 0; d < data.size(); d++)
    {
        data[d].set_ht_cnts();
    }

    f.close();
}

void tokenize(string line, vector<string> &words)
{
    /* empty the words vector */
    words.clear();

    if (line == "")
    {
        return;
    }

    /* update the words vector from line */
    string::iterator word_itr, space_itr;
    string token = "";
    word_itr = line.begin();                        /* word_itr is beginning of
line */
    space_itr = find(word_itr, line.end(), ' '); /* find space */

    while (space_itr != line.end())
    {
        token = string(word_itr, space_itr);
        words.push_back(token);

        word_itr = space_itr + 1;
        space_itr = find(word_itr, line.end(), ' '); /* find space */
    }

    token = string(word_itr, space_itr);
    words.push_back(token);

    return;
}

```