```
//Project Created for Homework 1 of Coure CSE 250A 2015 Fall, UCSD.
//All copyrights reserved by creater Jingyuan Li.
//Created on Oct. 4. 2015.
//Implemeented the simulator of the gamem Hangman. Input text file containing
    the corpus. Output the next best guess letter as well as its probability.

#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <unordered_map>

using namespace std;

//Customized compare function for std::sort(pair).
bool compare(const pair<string, int> &p1, const pair<string, int> &p2){
    return p1.second > p2.second;
}


class Wordlist {
    public:
        int wordlength;
        double totalWords;
        vector<pair<string, int> > wordlist;
        unordered_map<string, double> priorprob;
        vector<int> unknownposition;
        unordered_map<char, vector<int> > lastguessresult;
        Wordlist (int);
        void readFiletoSortedWordlist (ifstream& Inputfile);
        void priorProbability ();
        void updateUnknownPosition ();
        void guessResultGenerator (string answer, char thisguess);
        unordered_map<char, double> nextGuessGenerator ();
        vector<pair<char, double> > nextBestGuess (unordered_map<char, double>
            probforeachletter);
        string answerGenerator();
};

//Constructor: Set word length. Set all the letters to be unknown.
Wordlist::Wordlist(int numofletters){
    wordlength = numofletters;
    for (int i = 0; i < numofletters; i++) unknownposition.push_back(i);
}

//Read the corpus into a vector that containing pairs. The key in each pair is
    the word and the value is the number of its occurance. Also count the
    total number of words in the corpus. Sort the word list in descending
    order of the occurrence frequency in the end.
void Wordlist::readFiletoSortedWordlist (ifstream& Inputfile){
    totalWords = 0;
    if (Inputfile.is_open()){
        string buffer, word;
        int count, i = 1;
        while ( Inputfile >> buffer ){
            if (i % 2 == 1) word = buffer;
            else {
                count = stoi(buffer);
                totalWords += count;
                pair<string, int> p (word, count);
                wordlist.push_back(p);
            }
            i++;
        }
        Inputfile.close();
    }
    else cout << "Unable to read file." << endl;
    sort(wordlist.begin(), wordlist.end(), compare);
}

//Calculate the prior probability of each wird in the corpus. Store the result
    in an unordered map, with the word as the key and the probability as the
    value.
void Wordlist::priorProbability (){
    for (int i = 0; i < wordlist.size(); i++){
        priorprob[wordlist[i].first] = wordlist[i].second/totalWords;
    }
}

//Randomly select one word from the corpus as the answer word.
string Wordlist::answerGenerator(){
    srand(time(NULL));
    int rnd = rand() % wordlist.size();
    return wordlist[rnd].first;
}

//According to the result of the last guess, which is condition E in this
    question, determine the P(Li = l for some positions | E ) for each
    unguessed letter.
unordered_map<char, double> Wordlist::nextGuessGenerator (){
    //Define a map to store the calculation result for this round.
    unordered_map<char, double> probforeachletter;

    //Since we have to calculate P(E | W = w')P(W = w'), we have to find all
        the words which satisfy condition E, because P(E | W = w') = 1 only
        for these words. Store all these words in the bigger candidate word
        set.
    vector<string> biggercandidatewords;
    for (int j = 0; j < wordlist.size(); j++){
        bool flag = true;
        for (unordered_map<char, vector<int> >::iterator itr = lastguessresult
            .begin(); itr != lastguessresult.end(); itr++){
            //If this letter is not in the answer, it should not appear in
                this candidate word.
            if ((*itr).second[0] == -1){
                for (int n = 0; n < wordlength; n++){
                    if (wordlist[j].first[n] == (*itr).first) { flag = false;
                        break; }
                }
            }
            //If this word doesn't have any wrong letters, then check if right
                letters also in the right position of this word and these
                letters are not in the unknown position.
            else{
                for (int m = 0; m < (*itr).second.size(); m++){
                    if (wordlist[j].first[(*itr).second[m]] != (*itr).first) {
                        flag = false; break; }
                    for (int s = 0; s < unknownposition.size(); s++){
                        if (wordlist[j].first[unknownposition[s]] == (*itr).
                            first) {flag = false; break;}
                    }
                }
            }
            if (!flag) break;
        }
        if (flag) biggercandidatewords.push_back(wordlist[j].first);
    }

    //Now we can calculate sum(P(E|W = w')P(W = w')).
    double probbaseforthisround = 0;
    for (int i = 0; i < biggercandidatewords.size(); i++){
        if (priorprob.find(biggercandidatewords[i]) != priorprob.end())
            probbaseforthisround += (*priorprob.find(biggercandidatewords[i]))
            .second;
    }

    //Since P(Li = l | W = w) = 1 only for those words that both satisfy
        condition E and have letter l in its ith position, we now generate the
        candidate word set for this letter from the bigger candidate word set
        and calcluate its correct pribability.
    for (int i = 0; i < 25; i++){
        vector<string> candidatewords;
        double probofthisletter = 0;
        //If you can find this letter in the guessed letter table, skip. And
            its probability is 0.
        if (lastguessresult.find(char(i + 'A')) != lastguessresult.end()) {
            probforeachletter[char(i+'A')] = 0;
            cout << "prob for " << char(i + 'A') << ": " << probofthisletter <
                < endl;
            continue;
        }
        //First traverse the bigger candidate set, find out words that have
            this at the unknown positions.
        for (int m = 0; m < biggercandidatewords.size(); m++){
            for (int k = 0; k < unknownposition.size(); k++){
                //if the kth position of the mth word in the bigger candidate
                    set is the same as this letter, it means this word is a
                    candidate word.
                if (biggercandidatewords[m][unknownposition[k]] == char(i + 'A
                    ')) { candidatewords.push_back(biggercandidatewords[m]);
                    break; }
            }
        }

        //Now calculate the prob.
        for (int j = 0; j < candidatewords.size(); j++){
            if (priorprob.find(candidatewords[j]) != priorprob.end())
                probofthisletter += (*priorprob.find(candidatewords[j])).
                second / probbaseforthisround;
        }
        cout << "prob for " << char(i + 'A') << ": " << probofthisletter <<
            endl;
        probforeachletter[char(i+'A')] = probofthisletter;
    }
    return probforeachletter;
}

//Receive the probability table of each letter, then generate the letters with
    the biggest probability.
vector<pair<char, double> > Wordlist::nextBestGuess (unordered_map<char,
    double> probforeachletter){
    double mx = 0;
    vector<pair<char, double> > nextbestguess;
    for (unordered_map<char, double>::iterator itr = probforeachletter.begin()
        ; itr != probforeachletter.end(); itr++){
        pair<char, double> p ((*itr).first, (*itr).second);
        if ((*itr).second == mx) nextbestguess.push_back(p);
        if ((*itr).second > mx) {
            if (nextbestguess.size() != 0) nextbestguess.clear();
            nextbestguess.push_back(p);
            mx = (*itr).second;
        }
    }
    return nextbestguess;
}

//After user inputing a new guess, check and its correctness and record this
    guess in the last guessed letter table. This table represents condition E.
void Wordlist::guessResultGenerator (string answer, char thisguess){
    bool flag = false;
    for (int i = 0; i < wordlength; i++){
        if(answer[i] == thisguess) {lastguessresult[thisguess].push_back(i);
            flag = true; cout << "Right letter in position: " << i + 1 << endl
            ;}
    }
    if (!flag) { lastguessresult[thisguess].push_back(-1); cout << "Wrong
        letter: " << thisguess << endl; }
}

//Update the unknown position of the guessing word according to condition E.
void Wordlist::updateUnknownPosition (){
    for (unordered_map<char, vector<int> >::iterator itr = lastguessresult.
        begin(); itr != lastguessresult.end(); itr++){
        if ((*itr).second[0] != -1) {
            for (int i = 0; i < (*itr).second.size(); i++){
                vector<int>::iterator del = find(unknownposition.begin(),
                    unknownposition.end(), (*itr).second[i]);
                if (del != unknownposition.end()) unknownposition.erase(del);
            }
        }
    }
}



int main(){
    //Test case: word length = 5.
    Wordlist w(5);
    ifstream infile;
    infile.open("hw1_word_counts_05.txt");
    w.readFiletoSortedWordlist(infile);
    w.priorProbability();

    //Print the eight most frequent words. The words are printed in descending
        order.
    cout << "Eight most frequent words: ";
    for (int i = 0; i < 8; i++){
        cout << w.wordlist[i].first << " ";
    }
    cout << endl;
    //Print the eight least frequent words. The words are printed from the
        least frequent to frequent.
    cout << "Eight least frequent words: ";
    for (int i = w.wordlist.size() - 1; i > w.wordlist.size() - 9; i--){
        cout << w.wordlist[i].first << " ";
    }
    cout << endl;

    //Randomly select a string from wordlist to be answer.
    string answer = w.answerGenerator();
    //Print the answer for the programmer's reference.
    cout << "Answer for this round: " << answer << endl;

    //Start guessing. the process stops when all positions are guessed right.
    while (w.unknownposition.size() != 0){
        //Use a vector<pair<char, double> > to store the next best guess
            letter because there may be ties.
        vector<pair<char, double> > nextbestguess;
        //Generate next best guess letter.
        nextbestguess = w.nextBestGuess(w.nextGuessGenerator());
        //Print next best guess.
        for (int i = 0; i < nextbestguess.size(); i++){
            cout << "Next best guess: " << nextbestguess[i].first << " with
                Probability= " << nextbestguess[i].second << "." << endl;
        }
        //After giving the advice, ask for the next guess from the player.
        char in;
        cout << "Please put in your guess: ";
        cin >> in;
        //Generate the result of this guess.
        w.guessResultGenerator(answer, in);
        w.updateUnknownPosition();
    }
    //When all the letters are guessed, you are done!
    cout << "Done!" << endl;

    return 0;
}
```