Project 2

Source Code:

#include <iostream>

#include <fstream>

#include <string.h>

#include <locale>

#include <hash.hpp>

#include <stdc++.h>

using namespace std;

class stringBuild

{

public:

string mn;

string scratch;

const std::string::size\_type ScratchSize = 1024; // other arbitrary number to choose from.

stringBuild & append(const std::string & str)

{

scratch.append(str);

if (scratch.size() > ScratchSize)

{

mn.append(scratch);

scratch.resize(0);

}

return \*this;

}

const std::string & tostring()

{

if (scratch.size() > 0)

{

mn.append(scratch);

scratch.resize(0);

}

return mn;

}

};

class Node

{

public:

string W;

Node\* next;

Node(string key, Node\* next)

{

this->W = key;

this->next = next;

}

};

class Bucket

{

public:

Node\* first;

bool get(string in) { //return key true if key exists

Node\* next = first;

while (next != NULL)

{

if (next->W == in)

{

return true;

}

next = next->next;

}

return false;

}

void put(string key)

{

for (Node\* curr = first; curr != NULL; curr = curr->next)

{

if (key == curr->W) {

return; //search hit: return

}

}

first = new Node(key, first); //search miss: push\_back new node

}

};

class Dict

{

public:

int M = 1319;

Bucket\*\* array;

Dict()

{

this->M = M;

array = new Bucket\*[M];

for (int i = 0; i < M; i++)

{

array[i] = new Bucket();

}

}

int hash(string key)

{

boost::hash<std::string> string\_hash;

return (key.string\_hash() & 0x7fffffff) % M;

}

//call hash() to decide which bucket to put it in, do it.

void add(string key) {

array[hash(key)]->put(key);

}

//call hash() to find what bucket it's in, get it from that bucket.

bool contains(string input)

{

locale loc;

for (int i = 0; i<input.length(); i++)

input[i] = tolower(input[i], loc);

return array[hash(input)]->get(input);

}

void build(string filePath)

{

string line;

ifstream myfile(filePath);

if (myfile.is\_open())

{

while (myfile >> line){

add(line);

}

myfile.close();

}

}

//this method is used in my unit tests

string\* getRandomEntries(int num)

{

string \*toRet = new string[num];

for (int i = 0; i < num; i++)

{

//pick a random bucket, go out a random number

Node\* n = array[(int)rand() % M]->first;

int ran = (int)(rand() % (int)(sqrt(num)));

for (int j = 0; j<ran && n->next != NULL; j++) n = n->next;

toRet[i] = n->W;

}

return toRet;

}

};

class SpellCheck

{

public:

Dict\* dict;

static string filePath = "path to your W file";

static char alphabet[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' };

SpellCheck() {

dict = new Dict();

dict->build(filePath);

}

void run() {

bool done = false;

string input;

while (true) {

cout << "\nEnter a word of your choose: ";//Input of user entering a word!!!

cin >> input;

if (input == "")

{

break;

}

cout << "\n" << input;

if (dict->contains(input))

{

cout << " is spelled correctly";

}

else

{

cout << " is not spelled correctly, ";

cout << printSuggestions(input);

}

}

}

string printSuggestions(string input)

{

stringBuild \*sb = new stringBuild();

vector<string> print = makeSuggestions(input);

if (print.size() == 0) {

return "and I have no idea what W you could mean.\n";

}

sb->append("perhaps you meant:\n");

for (int i = 0; i<print.size(); i++) {

string s = print[i];

sb->append("\n -" + s);

}

return sb->tostring();

}

vector<string> makeSuggestions(string input) {

vector<string> toReturn;

vector<string> temp;

temp = charAppended(input);

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

toReturn.push\_back(temp[i]);

temp.clear();

temp = charMissing(input);

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

toReturn.push\_back(temp[i]);

temp.clear();

temp = charsSwapped(input);

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

toReturn.push\_back(temp[i]);

return toReturn;

}

vector<string> charAppended(string input)

{

vector<string> toReturn;

char c;

for (int i = 0; i<26; i++)//Loop of alphabet under 26

{

c = alphabet[i];

string atFront = c + input;

string atBack = input + c;

if (dict->contains(atFront)) {

toReturn.push\_back(atFront);

}

if (dict->contains(atBack)) {

toReturn.push\_back(atBack);

}

}

return toReturn;

}

vector<string> charMissing(string input)

{

vector<string> toReturn;

int len = input.length() - 1;

//try removing char from the front

if (dict->contains(input.substr(1)))

{

toReturn.push\_back(input.substr(1));

}

for (int i = 1; i < len; i++)

{

//try removing each char between (not including) the first and last

string working = input.substr(0, i);

working = working + input.substr((i + 1));

if (dict->contains(working))

{

toReturn.push\_back(working);

}

}

if (dict->contains(input.substr(0, len)))

{

toReturn.push\_back(input.substr(0, len));

}

return toReturn;

}

vector<string> charsSwapped(string input)

{

vector<string> toReturn;

for (int i = 0; i < input.length() - 1; i++)

{

string working = input.substr(0, i);

working = working + input.at(i + 1);

working = working + input.at(i);

working = working + (input.substr((i + 2)));

if (dict->contains(working)) {

toReturn.push\_back(working);

}

}

return toReturn;

}

};

int main()

{

SpellCheck \*sc = new SpellCheck();

sc->run();

}