#include <string>

#include <iostream>

using namespace std;

class node {

string word;

node \* left;

node \* right;

public:

node() { }

node(string newWord) {

word = newWord;

left = NULL;

right = NULL; }

bool isLessThan(string newWord);

void add(string newWord) {

if(this -> isLessThan(newWord)) {

if(right == NULL)

right = new node(newWord);

else

right -> add(newWord); }

else {

if(left == NULL)

left = new node(newWord);

else

left -> add(newWord); } }

bool isWord(string x);

bool hasWord(string x) {

if(this -> isWord(x))

return true;

else if(this -> isLessThan(x)) {

if(right == NULL)

return false;

else

return right -> hasWord(x); }

else {

if(left == NULL)

return false;

else

return left -> hasWord(x); } } };

class tree {

node \* top;

public:

tree() {

top = NULL; }

tree(node \* iNode) {

top = iNode; }

void add(string newWord) {

if(top == NULL)

top = new node(newWord);

else

top -> add(newWord); }

bool hasWord(string x) {

return top -> hasWord(x); } };

int value(string c) {

//’A’ has ASCII Code 65, this treat’s A’s value as 0

int value = c[0] – 65;

//’Z’ – 65 = 25, so anything after will be changed

if(value > 25)

/\*’a’ – 65 – 32 = 0, giving it the same value as ‘A’

This will also make all capital letters equivalent

lower case letters.

The drawback is that other symbols will be falsely

equivalent ie:’`’ = ‘@’, but this is acceptable

because it is unlikely that strings such as “c@t”

and “c`t” will need to be compared. The code could

be rewritten to make sure that the two are not equivalent,

but it would require more if statements, and it

would be unnecessary.\*/

return value - 32;

else

return value; }

bool node::isLessThan(string newWord) {

/\*This method should return return true if

word comes before newWord alphabetically

and return false if word comes after

newWord alphabetically (and if they are the same word) \*/

if(newWord.size() <= word.size()) {

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

/\*The program must be case insensitive

value function renders 'A' and 'a' to the same value

it does the same for all letters \*/

if(value(newWord) < value(word))

return false;

else if(value(newWord) > value(word))

return true; }

/\*If it gets to this point, word and newWord might be the same,

for example: newWord = "green" and word = "greenish

in either case, send newWord to the left node \*/

return false; }

else {

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

if(value(newWord) < value(word))

return false;

else if(value(newWord) > value(word))

return true; }

/\*For the function to get here, newWord

must be greater in length than the current word

in this case, newWord must share all of word's letters

and be longer, so it goes right alphabetically \*/

return true; } }

bool node::isWord(string x) {

if(word.size() != x.size())

return false;

else {

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

if(value(word) != value(x))

return false; }

return true; } }

int main() {

tree wordTree;

string input = "";

cout << "Enter ';' to end reading.\n";

while(true) {

cout << "Enter a word: ";

cin >> input;

if(input == ";")

break;

wordTree.add(input); }

input = "";

cout << "Enter ';' to stop checking for words.\n";

while(true) {

cout << "Enter a word: ";

cin >> input;

if(input == ";")

break;

if(wordTree.hasWord(input))

cout << "yes\n";

else

cout << "no\n"; }

cout << "Bye.\n";

return 0; }