```
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
using namespace std;

#include "ctimer.h"
#include "tvector.h"

class Unique
{
  public:
    virtual void add(const string& s) = 0;
    void add(const tvector<string>& list);
    virtual int size()                            const = 0;
    virtual void getAll(tvector<string>& list) const = 0;
};

void Unique::add(const tvector<string>& list)
{
    for(int k=0; k < list.size(); k++){
        add(list[k]);
    }
}

class DoubleListUnique : public Unique
{
  public:
    DoubleListUnique()
        : myFirst(0),
          myCount(0)
    { }
    virtual ~DoubleListUnique(){ }

    virtual void add(const string& s){
        myFirst = addHelper(s,myFirst);
    }
    virtual int size() const {
        return myCount;
    }
    virtual void getAll(tvector<string>& list) const {
        Node * current = myFirst;
        while (current != 0){
            list.push_back(current->info);
            current = current->next;
        }
    }

  protected:

    struct Node {
        string info;
        Node * next;
        Node * prev;
        Node (const string& s, Node * after, Node * before)
            : info(s),
              next(after),
              prev(before)
        { }
    };

    virtual Node * addHelper(const string& s, Node * list) {

        Node * first = list;
        while (list != 0){
            if (list->info == s){
                return first;
            }
            list = list->next;
        }
        first = new Node(s,first,0);  // add to front
        myCount++;
        return first;
    }

    Node * myFirst;
    int myCount;
};

class TreeUnique : public DoubleListUnique
{
  public:
    TreeUnique()
        : DoubleListUnique()   // not needed, make it clear
    { }

  protected:

    virtual Node * addHelper(const string& s, Node * list) {

        if (list == 0) {
            myCount++;
            return new Node(s,0,0);
        }
        if (list->info == s) return list;
        if (list->info < s){
            list->next = addHelper(s,list->next);
        }
        else {
            list->prev = addHelper(s,list->prev);
        }
        return list;
    }
};

double stress(const tvector<string>& list, Unique * unique){

    CTimer timer;
    timer.Start();
    unique->add(list);
    timer.Stop();
    return timer.ElapsedTime();
}

int main(int argc, char * argv[])
{
    tvector<string> list;
    string filename,word;
    if (argc > 1) {
        filename = argv[1];
    }
    else {
        cout << "enter filename: ";
        cin >> filename;
    }
    ifstream input(filename.c_str());
    while (input >> word) {
        list.push_back(word);
    }
    Unique * listUn  = new DoubleListUnique();
    Unique * treeUn  = new TreeUnique();

    cout << "list time =\t "        << stress(list,listUn)
         << " size = " << listUn->size() << endl;
    cout << "tree time =\t "        << stress(list,treeUn)
         << " size = " << treeUn->size() << endl;
    return 0;
}
```