```
#include <iostream>
#include <string>
#include <queue>    // for priority-queue
using namespace std;

int main()
{

    priority_queue<string, vector<string>, greater<string> >pq;
    string word;

    while (cin >> word) {
        pq.push(word);
    }
    int wordsRead = pq.size();

    while (! pq.empty()) {
        word = pq.top();
        cout << word << endl;
        pq.pop();
    }
    cout << "read " << wordsRead << " words" << endl;
}
```

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

#include "tpq.h"

int main()
{
    tpqueue<string> pq;
    string word;

    while (cin >> word) {
        pq.insert(word);
    }
    int wordsRead = pq.size();

    while (! pq.isEmpty()) {
        word = pq.getmin();
        cout << word << endl;
        pq.deletemin();          // could use pq.deletemin(word)
    }
    cout << "read " << wordsRead << " words" << endl;
}
```

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

#include "ctimer.h"
#include "tpq.h"
#include "comparer.h"

/**
 * Revcomp compares strings "backwards": aardvark > zebra
 */

template <class Kind>
struct Revcomp : public Comparer<Kind>
{
    int compare(const Kind& lhs, const Kind& rhs) const
    {
        if (lhs < rhs)      return 1;
        else if (rhs < lhs) return -1;
        else                return 0;
    }
};

void printall(const tpqueue<string>& pq)
// post: pq elements printed
{
    CTimer timer;
    string w;
    timer.Start();
    tpqueue<string> copy = pq;

    while (copy.size() > 0) {
        copy.deletemin(w);
        cout << w << endl;
    }
    timer.Stop();
    cout << "remove time: " << timer.ElapsedTime() << endl;
}

int main()
{
    Revcomp<string> rcomp;

    tpqueue<string> pq(rcomp);
    tpqueue<string> pq2;
    string w;

    CTimer timer;

    timer.Start();
    while (cin >> w) {
        pq.insert(w);
        pq2.insert(w);
    }
    timer.Stop();
    cout << pq.size() << "\tread/store " << timer.ElapsedTime() << endl;

    printall(pq);
    cout << "----------" << endl;
    printall(pq2);

    return 0;
}
```

```
#ifndef _PQ_H
#define _PQ_H

// templated priority queue class
// written and modified many times
// 11/27/95
//          to use vector class rather than array type
// 12/30/96
//          changed to use two templated classes, one for
//          item, one for priority (old version had int
//          for priority)
//          also added DeleteMin in two ways: one has
//          parameters to return values when deleting
//
// 11/99    changed to tvector class
//
// 2/00     changed to single template parameter
// operations:
//
// tpqueue()
//     construct pqueue using standard < operator
//
// tpqueue(const Comparer& comp)
//     construct pqueue using comparer object comp, see comparer.h
//
// void insert(const Kind& )
//     insert Kind object
//
// int size()    const
//     returns the number of elements in the priority queue
//
// bool isEmpty() const
//     returns true if pqueue is empty, otherwise returns false
//
// void deletemin()
//     remove minimal element from queue
//
// void deletemin(Kind &)
//     remove minimal element and return it
//
// Kind&  getmin()
//     return minimal element
//
// Complexity:
//
// a heap-based implementation is used, so minelt is O(1)
// and insert/deletemin are O(log n) where n is # of items in pqueue
// (insert is O(1) average case)
//

#include "tvector.h"
#include "comparer.h"


template <class Kind>
class tpqueue
{
  public:
    tpqueue();
    tpqueue(const Comparer<Kind>& cmp);
    ~tpqueue();
    void insert(const Kind & kind);      // insert kind
    int size()     const;                // return # of elements
    bool isEmpty() const;                // return true iff empty

    void deletemin();                    // remove min elt from
    void deletemin(Kind &);              // remove/return min elt
    Kind& getmin( );                     // return min elt
```



```
    void dump() const;

  private:
    void init (int n);                   // reserve space for n elements

    const Comparer<Kind>& myComp;

    static Comparer<Kind> ourComp;

    void heapify(int vroot);             // percolate down function
    tvector<Kind> myList;                // array of prioritized stuff
    int myNumElts;                       // # elements in priority queue
};

// uncomment line below if including source code
#include "tpq.cpp"

#endif
```