```
#include <iostream>
#include <fstream>
#include <map>
#include <set>
#include <string>
#include <sstream>
#include <cstdlib>     // for exit
using namespace std;

/**
 * read a file, print each word and line numbers
 * on which word occurs, words printed in alphabetical order
 * @author Owen Astrachan
 */

typedef map<string, set<int> > linemap;   // easier to use linemap in code

ostream& operator <<(ostream& out,const pair<string, set<int> >& p)
// post: string/set printed all on line line
{
    out << p.first << "\t";

    set<int>::iterator it  = p.second.begin();  // print set on one line
    for(; it != p.second.end(); it++) {
        out << *it << " ";
    }
    return out;
}

int main(int argc, char *argv[])
{
    string filename,line, w;
    if (argc == 1)
    {
        cerr << "usage: " << argv[0] << " filename" << endl;
        exit(1);
    }
    filename = argv[1];
    ifstream input(filename.c_str());

    linemap info;
    int linecount = 0;
    while (getline(input,line)) {

        linecount++;
        istringstream iline(line);
        while (iline >> w) {
            linemap::iterator it = info.find(w);
            if (it == info.end()) {
                set<int> si;
                si.insert(linecount);
                info.insert(make_pair(w,si));
            }
            else {
                it->second.insert(linecount);
            }
        }
    }
    linemap::iterator it;
    for(it = info.begin(); it != info.end(); it++) {
        cout << *it << endl;
    }

    return 0;
}
```

# CPS 108, Spring 2004, Run, Right, Fast

This assignment uses wordlines.cpp as a starting point.

The assignment has three parts. For each part you'll write a program that will read a file and determine the words that occur most frequently in the file. Words are delimited by whitespace and should not have leading or trailing punctuation. All characters should be converted to lowercase equivalents.

The program should print the *n* words that occur most frequently, where *n* defaults to 20, but is otherwise a parameter to the program. If no filename is specified, the program should read from standard input (cin). The examples below show how the program can be used.

## Usage

```
wordcount

wordcount -f filename -c wordcount

wordcount --file[=filename] --count[=wordcount]
```

For example, the following are all valid uses.

```
wordcount < ~/data/poe.txt

wordcount -c < ~/data/poe.txt

wordcount -file=/u/ola/data/poe.txt --count=30

wordcount -f ~/data/poe.txt -c 30
```

## Versions

1. Version 1.0 should work correctly reading from standard input and determining the 20 words that occur most frequently (break ties alphabetically). You can proces command-line options, but you don't need to. The grade will be based only on correctness, but comments on design will be given.
2. Version 2.0 should process command-line options and be designed using classes to facilitate alternative implementations with minimal rewriting. The grade will be based 50% on design and 50% on conforming to specifications.
3. Version 3.0 should be as fast as possible. It will be graded 80% on speed and 20% on design. Programs that produce incorrect output will receive little credit.

---

*Owen L. Astrachan*
Last modified: Wed Jan 7 12:57:20 EST 2004