sample application: telephone directory

```
1 tinf2$ tel fred "x2356 02-6124441" # insert data for fred
2 tinf2$ tel jane 092-6124441 # insert data for jane
3 tinf2$ tel fred # retrieve data for fred
fred    x2356 02-6124441
4 tinf2$ tel fred ""  # delete fred
5 tinf2$ tel fred
"fred" not found
6 tinf2$ tel # show all data in the file
jane 092-6124441
```

tel.h

```
1  #ifndef TEL_H
2  #define TEL_H
3  // $Id: structuur2.tex,v 1.26 2008/09/02 11:51:15 dvermeir Exp
4  #include <string>
5  #include <iostream>
6
7  // associates a string info_ with a string name_
8  class TelRecord {
9    public:
10     TelRecord(const std::string& name="",
11       const std::string& info=""): name_(name), info_(info) {}
12     TelRecord(const TelRecord& r):
13       name_(r.name_), info_(r.info_) {}
14     // two records are the same if their names match
15     bool operator==(const TelRecord& r) const {
16       return name_ == r.name_;
17     }
```

```
18      // needed for std::set<TelRecord>
19      bool operator<(const TelRecord& r) const {
20        return name_ < r.name_;
21      }
22      friend std::ostream& operator<<(std::ostream& os,
23        const TelRecord& r);
24      friend std::istream& operator>>(std::istream& os,
25        TelRecord& r);
26    private:
27      // name_ is a unique key: no 2 records can
28      // have the same name_, name_ may not
29      // contain white space
30      std::string name_;
31      std::string info_;
32  };
33  #endif
```

tel.C

```
1  // $Id: structuur2.tex,v 1.26 2008/09/02 11:51:15 dvermeir Exp
2  // usage:
3  //  tel name       -- query: retrieve info for name
4  //                    from database
5  //  tel name info -- insert: store/replace info for
6  //                    name in database
7  //  tel name ""    -- delete: remove name from database
8  //  tel            -- list whole database
9  //
10 // error return codes:
11 //
12 //    1       -- query: not found
13 //    2       -- insert: cannot write database file
14 //    3       -- delete: not found
15 //    4       -- too many arguments
16 //
17 // only 1 info string can be associated with a name
18 // in the database
```

```
19 #include <iterator>
20 #include <fstream>
21 #include <set>
22 #include "tel.h"
23
24 std::ostream&
25 operator<<(std::ostream& os, const TelRecord& r) {
26   os << r.name_ << '\t' << r.info_ << std::endl;
27   return os;
28 }
29
30 std::istream&
31 operator>>(std::istream& is, TelRecord& r) {
32   is >> r.name_ ; // line starts with name
33   is.ignore(); // ignore \t following name
34   std::getline(is, r.info_); // rest of line is info
35   return is;
36 }
```

```
37 // note that to use set<T>, T needs a default and a
38 //   copy ctor as well as an operator<
39 typedef std::set<TelRecord> record_set;
40 typedef std::istream_iterator<TelRecord> record_input_it;
41
42 const char* const database_name = "tel.data";
43
44 int
45 main (int argc, char *argv[]) {
46   record_set dir;
47   { // Load the database into a record_set
48     std::ifstream data_file(database_name);
49     if (data_file)
50       std::copy(
51         record_input_it(data_file),
52         record_input_it(),
53         std::inserter(dir, dir.begin())
54         );
55   }
56   // data_files is automatically closed by the ifstream destru
```

```
57    switch (argc) {
58      // no arguments: just show all records on std::cout
59      case 1:
60        copy( dir.begin(), dir.end(),
61              std::ostream_iterator<TelRecord>(std::cout)
62            );
63        return 0;
```

```
64      // 1 argument: retrieve info associated with argv[1]
65      case 2:
66        { // use the find() algo with dummy TelRecord with name_
67          std::string key(argv[1]);
68          record_set::iterator r = dir.find(TelRecord(key, ""));
69
70          if (r!=dir.end()) {
71            // success: iterator points to found record
72            std::cout << *r; // output it
73            return 0;
74          }
75          else {
76            std::cerr << "\"" <<  key
77              << "\" not found" << std::endl;
78            return 1;
79          }
80        }
81        break;
```

```
82    // 2 arguments, insert, replace or delete
83    case 3: {
84      std::string key(argv[1]);
85      std::string info(argv[2]);
86      TelRecord r(key, info);
87
88      if (info.size()==0) { // delete
89        if (dir.erase(r)!=1) {
90          // set::erase() returns number of deleted elements
91          std::cerr << "\"" << key
92            << "\" not found; cannot erase" << std::endl;
93          return 3;
94        }
95      }
96      else { // insert or replace
97        std::pair<record_set::iterator, bool>  pair =
98            dir.insert(r);
99        if (!pair.second) { // insert failed: duplicate key
100         dir.erase(pair.first); // erase,
101         dir.insert(r); // then insert again
102       }
103     }
```

```
104       // save to file after update
105       std::ofstream data_file(database_name);
106       if (!data_file) {
107         std::cerr << "Cannot open \"" << database_name
108           << "\" for writing" << std::endl;
109         return 2;
110       }
111       std::copy(dir.begin(), dir.end(),
112            std::ostream_iterator<TelRecord>(data_file));
113       return 0;
114       }
115       break;
116     default:
117       std::cerr << "Usage: " << argv[0]
118         << " [name [info|\"\"]] " << std::endl;
119       return 4;
120       break;
121     }
122     return 0;
123   }
```