```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#ifndef VERSIONE
#define VERSIONE 0
#endif

// gestione ordinamento
#if 0
// bubblesort non ottimizzato
void bubblesort(char *history[], int n)
{
    int i, j;
    for(j=0; j<n-1; j++) {
        for(i=0; i<n-1; i++) {
            if(strcmp(history[i], history[i+1]) > 0) {
                char *tmp = history[i];
                history[i] = history[i+1];
                history[i+1] = tmp;
            }
        }
    }
    // si noti che altermine del ciclo j
    // gli ultimi elementi j+1 del vettore sono già in posizione
    corretta
```

```
26  }
27  #endif
28
29  #if VERSIONE == 2
30  void bubblesort(char *history[], int n)
31  {
32      int i, j;
33      for(j=n-1; j>0; j--) {
34          for(i=0; i<j; i++) {
35              if(strcmp(history[i], history[i+1]) > 0) {
36                  char *tmp = history[i];
37                  history[i] = history[i+1];
38                  history[i+1] = tmp;
39              }
40          }
41      }
42  }
43  #endif
44
45  #if VERSIONE == 3
46
47  // cmpfunction è un puntatore a funzione
48  typedef int (*cmpfunction)(void*, void*);
49
50  void bubblesort(void *vet[], int n, cmpfunction cmp)
51  {
```

```
52      int i, j, k;
53      for(j=n-1; j>0; j=k) {
54          k=-1;
55          for(i=0; i<j; i++) {
56              if((*cmp)(vet[i], vet[i+1]) > 0) {
57                  void *tmp = vet[i];
58                  vet[i] = vet[i+1];
59                  vet[i+1] = tmp;
60                  k=i;
61              }
62          }
63      }
64  }
65  #endif
66
67  #if VERSIONE == 4
68
69  int cmp(const void *p1, const void *p2)
70  {
71      /* The actual arguments to this function are "pointers to
72        pointers to char", but strcmp arguments are "pointers
73        to char", hence the following cast plus dereference (see: man
          qsort)*/
74
75      return strcmp(* (char **) p1, * (char **) p2);
76  }
```

```
 77
 78  #endif
 79
 80  /*
 81   * ricerco un dato in un vettore ordinato
 82   */
 83  int binary_search(char *history[], int n, char *target)
 84  {
 85      int left = 0; // indice inferiore
 86      int right = n-1; // indice superiore
 87      // oviiamente si deve avere right>=left
 88      while(right>=left) {
 89          int m=(right+left)/2; // indice mediano
 90          int ris = strcmp(history[m], target);
 91          if(ris == 0) return m; // ho trovato
 92          if(ris<0) {
 93              left = m+1; // ricerco nella parte alta del vettore
 94          } else {
 95              right = m-1; // ricerco nella parte bassa del vettore
 96          }
 97      }
 98      return -1;
 99  }
100
101  int test_ordinamento(char *history[], int n)
102  {
```

```
103         for(n--; n>0; n--) {
104             if(strcmp(history[n], history[n-1]) < 0) return 0;
105         }
106         return 1;
107     }
108     // fine gestione ordinamento
109
110     /*
111      * cerco un comando che inizia con target
112      * viene restituito il più recente o NULL
113      */
114     char* search(char *history[], int n, char *target)
115     {
116         int l = strlen(target);
117         // parto dal comando più recente (ultimo memorizzato)
118         for(n--; n>=0; n--) {
119             // confronto al più l caratteri
120             if(strncmp(history[n], target, l) == 0) return history[n];
121         }
122         return NULL;
123     }
124
```

```
125  int main(int argc, char *argv[])
126  {
127      FILE *f = stdin; // sorgente dei dati letti - di default stdin
128      int max = 1000, // numero massimo di comandi memorizzabili
129          n=0, // numero di comandi memorizzati
130          debug=0, // flag per un eventuale debugging
131          ric=-1, // ric>=0 eseguo un test di ricerca
132          i;
133      FILE *fsort = NULL;
134
135      // gestione parametri
136      for(i=1; i<argc; i++) {
137          // il parametro inizia con if=
138          // nome del file da leggere
139          if(strncmp(argv[i], "if=", 3) == 0) {
140              f = fopen(argv[i]+3, "r");
141              if(!f) {
142                  perror(argv[i]);
143                  exit(1);
144              }
145              continue;
146          }
147          // il parametro inizia con sort=
148          // eventuale file in scrittura (vengono memorizzati i
             comandi ordinati)
149          if(strncmp(argv[i], "sort=", 5) == 0) {
```

```
150             fsort = fopen(argv[i]+5, "w");
151             if(!fsort) {
152                 perror(argv[i]);
153                 exit(1);
154             }
155             continue;
156         }
157         // il parametro inizia con max=
158         if(strncmp(argv[i], "max=", 4) == 0) {
159             max=atoi(argv[i]+4);
160             continue;
161         }
162         // il parametro inizia con ric=
163         if(strncmp(argv[i], "ric=", 4) == 0) {
164             ric=atoi(argv[i]+4);
165             continue;
166         }
167         // viene utilizzato il parametro successivo
168         if(strcmp(argv[i], "--ric") == 0) {
169             if(argv[++i]) ric = atoi(argv[i]);
170             else ric = 1000;
171             continue;
172         }
173         // il parametro è un flag
174         if(strcmp(argv[i], "--debug") == 0) {
175             debug=1;
```

```
176             continue;
177         }
178         if(strcmp(argv[i], "--version") == 0) {
179             printf("%s VERSIONE=%d\n", argv[0], VERSIONE);
180             return 0;
181         }
182         if(strcmp(argv[i], "--help") == 0) {
183             printf("uso: %s [--help] [--debug]  [--version]
                [max=(intero)] [if=file] [sort=file]\n", argv[0]);
184             return 0;
185         }
186         // non sono riuscito a interpretare il parametro
187         fprintf(stderr, "parametro non gestito: %s\n", argv[i]);
188         // spesso questa situazione termina l'esame dei parametri
189         // si usa allora un break
190     }
191     // fine gestione parametri
192
193     // allocazione dei vettori utilizzati
194     char buffer[1024];
195     char **history = malloc(max * sizeof(char*));
196     if(!history) {
197         perror("main");
198         max = 0;
199     }
200
```

```
201       //ciclo di lettura
202       while(fgets(buffer, sizeof(buffer), f)) {
203           int l = strlen(buffer);
204           buffer[l-1] = 0; // elimino il carattere di newline
205           // non vengono gestite righe troppo lunghe
206           if(*buffer == '!') {
207               char *trovato = search(history, n, buffer+1);
208               if(trovato) {
209                   printf("input: '%s': eseguo: %s\n", buffer,
                      trovato);
210               } else {
211                   printf("input: '%s': comando non trovato\n",
                      buffer);
212               }
213               continue;
214           }
215   #if VERSIONE > 0
216   // aumento la dimensione del vettore
217           if(n==max) {
218               char ** tmp = realloc(history, max*2 * sizeof(char*));
219               if(tmp) {
220                   history = tmp;
221                   max += max;
222               }
223               if(debug) fprintf(stderr, "riallocato %d!\n", max);
224           }
```

```
225  #endif
226          if(n<max) {
227              history[n] = strdup(buffer);
228              /* codice equivalente
229              l = strlen(buffer);
230              history[n] = malloc((l+1)*sizeof(char));
231              strcpy(history[n], buffer);
232              */
233              if(history[n] == NULL) {
234                  perror(buffer);
235              } else {
236                  n++;
237              }
238          }
239      }
240
241      // operazioni finali se definite
242  #if VERSIONE >= 2
243      if(fsort) {
244  #if VERSIONE == 2
245          bubblesort(history, n);
246  #elif VERSIONE == 3
247          bubblesort((void **) history, n, (cmpfunction) strcmp);
248  #elif VERSIONE == 4
249          /* parametri di qsort:
250          1: vettore da ordinare
```

```
251             2: numero di elementi del vettore
252             3: dimensione in byte del singolo elemento
253             4: funzione di confronto fra gli elementi
254             per maggiori dettagli eseguire: man 3 qsort
255             NB: il vettore viene modificato
256             */
257             qsort(history, n, sizeof(*history), cmp);
258     #endif
259             for(i=0; i<n; i++) fprintf(fsort, "%3d %s\n", i+1, history[
                i]);
260
261             // verifica ordinamento
262             if(!test_ordinamento(history, n)) fprintf(stderr, "Il
                vettore non e' ordinato\n");
263             // test ricerca binaria
264             if(ric>=0) printf("ricerca: %d\n", binary_search(history,
                n, ric<n ? history[ric] : "!!"));
265         }
266     #endif
267
268         return 0;
269     }
270
```