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

#define SIZE 100 /* dimensione dei buffer di lettura */
#define DIM 100 /* dimensione del vettore nomi */

int main(int argc, char **argv)
{
    FILE *fin = fopen(argv[1], "r");
    FILE *fout = argc<3 ? stdout : fopen(argv[2], "w");
    /*
     * assert termina il programma se il test non è vero
     * con un messaggio del tipo

% ./leggi-passwd file-che-non-esiste
assertion "fin && fout" failed: file "leggi-passwd.c", line 23

     * serve per il debugging non per l'utente finale
     * se definisco la macro NDEBUG il compilatore non considera
     * assert(...)
     */
    assert(fin && fout); /* richiede assert.h */
```

```
24      /*
25       * si noti che SIZE e DIM pur assumendo lo stesso valore
26       * sono logicamente indipendenti
27       */
28      char nome[SIZE], passwd[SIZE], dir[SIZE], shell[SIZE];
29      /* buffer mi serve per leggere una riga intera */
30      char buffer[1000];
31      char *nomi[DIM];
32
33  #if 0
34
35  esempio di file /etc/passwd:
36
37  root:x:0:0:root:/root:/bin/bash
38  bin:x:2:2:bin:/bin:/bin/sh
39  sys:x:3:3:sys:/dev:/bin/sh
40  kermit:x:1000:1000:Kermit the Frog,,,:/home/kermit:/bin/bash
41
42  nome utente:password:identificativo utente:identificativo gruppo:
    descrizione:directory di lavoro:shell utilizzata
43
44  nelle versioni recenti di Unix la password non viene memorizzata (
    in questo file)
45  nelle vecchie era presente ma crittografata (qualcosa come $1$
    oTc1DHLi$m4cqP7MVPu2Plrw/6Ov.v/)
46  #endif
```

```
47      /*
48       * legge al massimo DIM righe oppure fino al termine del file
49       * fgets restituisce NULL se si incontra EOF (end of file)
50       */
51      int n;
52      for(n=0; n<DIM && fgets(buffer, sizeof(buffer), fin); n++) {
53      /*
54       * %[^:]: legge una stringa costituita da un qualunque
         carattere
55       * escluso ':' poi viene letto necessariamente il carattere ':'
56       *
57       * %*d l'asterisco indica che viene letto un dato
58       * (in questo caso un intero) ma non memorizzato
59       */
60          int id; /* è usata solo nel ciclo */
61          if(sscanf(buffer, "%[^:]:%[^:]:%d:%*d:%*[^:]:%[^:]:%s",
62              nome, passwd, &id, dir, shell) != 5) {
63              fprintf(stderr,"Errore nella riga
                %d\n%s\n",n+1,buffer);
64          }
65          fprintf(fout, "%s %s %d %s %s\n",
66              nome, passwd, id, dir, shell);
67          nomi[n] = nome;
68      }
```

```
69  #if __STDC_VERSION__ < 199901L
70      int i;
71      for(i=0; i<n; i++) {
72          printf("%2i %s\n", i+1, nomi[i]);// %i è equivalente a %d
73      }
74  #else
75      for(int i=0; i<n; i++) {
76          sprintf(buffer, "%2i %s.\n", i+1, nomi[i]);
77          fputs(buffer, stdout);
78      }
79  #endif
80      return 0;
81  }
82  #if 0
83  in analogia a sscanf esiste anche sprintf
84  (scrive in un buffer di caratteri - cioè crea una stringa)
85
86  si può sostituire la riga 72 con:
87
88  sprintf(buffer, "%2i %s\n", i+1, nomi[i]);
89  fputs(buffer, stdout);
90
91  lo standard C99 ammette la definizione di variabili in un ciclo for
92  vedi riga 75 (compilare con l'opzione -std=c99)
93  #endif
```

```
 94  /*
 95   * nomi[n] = nome;
 96   * va corretto in:
 97   * nomi[n] = strdup(nome);
 98   *
 99   * char* strdup(char *s) (duplica stringa - richiede string.h) fa
100   * una copia della stringa in una nuova locazione di memoria il cui
101   * indirizzo viene restiuito come risultato della funzione
102   */
103
```