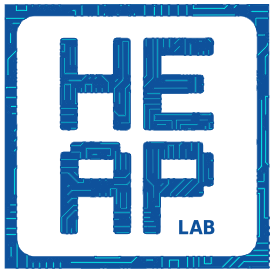
Esercizi su files di testo in C

Stefano Cherubin*



17/11/2017

[**Informatica A**] Esercitazione #13

corso per Ing. Gestionale a.a. 2017/18

*<nome>.<cognome>@polimi.it

Indice

1 Giralettere

Si desidera sviluppare un programma in linguaggio C per la modifica di un file di testo. La modifica consiste nel sostituire (scambiandoli tra loro) due caratteri alfabetici dati. In particolare, tutte le occorrenze del primo carattere dovranno essere sostituite dal secondo e viceversa.

La sostituzione deve avvenire mantenendo la forma (maiuscola o minuscola) della lettera originaria.

La libreria `ctype.h` contiene alcune funzioni utili che possono essere usate, quali:

- `char toupper(char)`
- `char tolower(char)`
- `int isupper(char)`
- `int islower(char)`

1.1 Approccio alla soluzione

1.1.1 Riga per riga vs carattere per carattere

È possibile acquisire una riga alla volta del file di input ed effettuare operazioni di sostituzione lettere in memoria prima di scrivere sul file di output. Questo tuttavia non è possibile perché non viene fornita alcuna indicazione sulla dimensione massima delle righe del file. Non potendo seguire l'approccio riga per riga in questo caso, si ricorre all'approccio carattere per carattere.

È comunque possibile leggere un buffer sufficientemente grande di caratteri dal file di input, elaborarne i caratteri, scriverlo i output e poi riprendere a leggere dal punto di interruzione. Questa soluzione non viene però presentata per brevità; le viene preferito un approccio carattere per carattere, il quale è più semplice e di intuitiva applicazione al problema presentato.

1.2 Soluzione C

Listato 1: Giralettere

```
1  #include <stdio.h>
2  #include <ctype.h>
3  #define FILEIN "input.txt"
4  #define FILEOUT "output.txt"
5  int main() {
6    FILE* fin;
7    FILE* fout;
8    char ch, c1, c2;
9    printf("Inserisci i due caratteri che vuoi scambiare
         , senza separarli da spazio\n");
10   scanf("%c%c", &c1, &c2);
11   fin = fopen(FILEIN, "r");
12   if (!fin) {
13     printf("\nErrore apertura file.");
14     return -1;
15   }
16   fout = fopen(FILEOUT, "w");
17   if (!fout) {
18     printf("\nErrore apertura file.");
19     fclose(fin);
20     return -1;
21   }
22   c1 = tolower(c1);
23   c2 = tolower(c2);
24   while ((ch = fgetc(fin)) != EOF ) {
25     if (tolower(ch) == c1 ){
26       if (isupper(ch))
27         ch = toupper(c2);
28       else
29         ch = c2;
30     } else if (tolower(ch) == c2) {
31       if (isupper(ch) )
32         ch = toupper(c1);
33       else
34         ch = c1;
35     }
36     fputc(ch, fout);
37   }
38   fclose(fin);
39   fclose(fout);
40   return 0;
41 }
```

2 WC

Si consideri il comando unix `wc`. Esso fornisce delle statistiche relative ad un file di testo, tra cui numero di caratteri, numero di parole, numero di linee contenute nel file di testo. Si codifichi in linguaggio C una funzione che prenda in input il nome di un file di testo e restituisca in output le statistiche sopra descritte.

2.1 Soluzione C

Listato 2: word count

```c
#include <stdio.h>

typedef struct ret_s {
  int linecount, wordcount, charcount;
} ret_t;

ret_t wc (const char* filename) {
  FILE *fp;
  char ch;
  ret_t ret;
  int twospaces = 1; // flag for multiple spaces

  ret.linecount = 0;
  ret.wordcount = 0;
  ret.charcount = 0;

  fp = fopen(filename, "r");
  if (!fp) {
    return ret;
  }

  ch = getc(fp);
  while (!feof(fp)) {
    if (ch != ' ' && ch != '\n') {
      ret.charcount++;
      twospaces = 0;
    }
    if ((ch == ' ' || ch == '\n') && !twospaces) {
      ret.wordcount++;
      twospaces = 1;
    }
    if (ch == '\n') {
      ret.linecount++;
    }
    ch = getc(fp);
  }
  fclose(fp);
  return ret;
}
```

3 Cesare

Il cifrario di Cesare è uno dei più antichi algoritmi crittografici conosciuti. Prende il suo nome da Giulio Cesare, utilizzatore di questo strumento nella sua corrispondenza privata.

Esso converte un testo in chiaro (*plain message*) in un testo cifrato (*encoded message*). Il messaggio cifrato si ottiene tramite uno spostamento applicato lettera per lettera sul testo originale di 3 posizioni nell'ordine alfabetico. Si veda il seguente schema di cifratura.

plain	encoded
A	X
B	Y
C	Z
D	A
E	B
F	C
G	D
H	E
I	F
J	G
K	H
L	I
M	J
N	K
O	L
P	M
Q	N
R	O
S	P
T	Q
U	R
V	S
W	T
X	U
Y	V
Z	W

Si sciva una funzione C che applichi la cifratura di Cesare ad un file di testo, producendo un secondo file di testo contenente il testo cifrato.

3.1 Soluzione C

Listato 3: cifrario di cesare

```
1  #include <stdio.h>
2
3  #define BUF_LEN 1024
4
5  int cesare (const char* plain, const char* encoded) {
6    int i;
7    FILE* fin;
8    FILE* fout;
9    char buffer[BUF_LEN];
10
11   if (!(fin = fopen(plain, "r"))) {
12     printf("Error opening input file\n");
13     return 1;
14   }
15
16   if (!(fout = fopen(encoded, "w"))) {
17     fclose(fin);
18     printf("Error opening output file\n");
19     return 1;
20   }
21
22   fgets(buffer, BUF_LEN, fin);
23   while(!feof(fin)) {
24     for (i = 0; buffer[i] != '\0'; i++) {
25       // if it is a printable character
26       if ((buffer[i] >= 'a' && buffer[i] <= 'z') || (
             buffer[i] >= 'A' && buffer[i] <= 'Z')) {
27         buffer[i] = buffer[i] + 3;
28         // if it is no more a printable character
29         if (!((buffer[i] >= 'a' && buffer[i] <= 'z')
               || (buffer[i] >= 'A' && buffer[i] <= 'Z')))
30           buffer[i] = buffer[i] + 'a' - 'z';
31       }
32     }
33     fputs(buffer, fout);
34     fgets(buffer, BUF_LEN, fin);
35   }
36   fclose(fin);
37   fclose(fout);
38   return 0;
39 }
```

4 Election simulator 2017

Si consideri la seguente versione semplificata dalla legge elettorale italiana «*Italicum*» per la Camera dei Deputati, in vigore da luglio 2016

- Gli elettori eleggono 630 deputati alla Camera dei Deputati, unica camera i cui membri sono scelti con elezione diretta
- 13 di questi deputati, in rappresentanza delle regioni a statuto speciale Valle D'Aosta e Trentino-Alto Adige, vengono scelti attraverso un sistema elettorale dedicato, in modo non diverso da come avveniva con la precedente legge elettorale
- Le liste che non superano la soglia di sbarramento posta al 3% dei voti, non ottengono seggi
- La lista che da sola ottiene più del 40% dei voti ottiene il premio di maggioranza
- Se nessuna singola lista ottiene più del 40% dei voti, il premio di maggioranza si decide al ballottaggio tra le due liste che hanno ottenuto più voti (è questo il caso di nessuna lista sopra il 40% o due liste sopra il 40%)
- Il premio di maggioranza è costituito da 340 seggi
- I restanti 277 seggi saranno assegnati con il metodo proporzionale di Hare-Niemer a tutte le altre liste che superano la soglia di sbarramento
- Il metodo proporzionale di Hare-Niemer prevede l'assegnazione dei seggi in base alle percentuali di voti ottenute e si sviluppa in due fasi:
  - Fase dei quozienti
    - Sia $Q = {}^{V}/_{N}$ numero intero detto quoziente di Hare, con V numero di voti validi e N numero di seggi da assegnare
    - Data una lista l, sia V_l il numero di voti che essa ha ottenuto
    - I seggi assegnati alla lista l si ottengono come arrotondamento per difetto della divisione $N_l = {}^{V_l}/_{Q}$
  - Fase dei resti
    - Sia $R_l = V_l - (N_l \cdot Q)$ numero intero associato alla lista l
    - Si ordino tutte le liste l in ordine decrescente in base al rispettivo valore R_l
    - Si assegni un ulteriore seggio a ciascuna lista, partendo da quella con R_l maggiore, fino a esaurimento dei seggi disponibili

1. Scrivere un programma C che, ricevuto in input il nome di un file contentente i voti di tutti gli elettori al primo turno[1], dica quale lista ha vinto le elezioni o, nel caso di ballottaggio, quali liste andranno al ballottaggio.

   - Il file in input si compone di un elenco dove ogni voce corrisponde al nome di una lista
   - Supporre che non ci siano più di 10 liste indipendenti in corsa
   - Supporre che il nome del file sia dato da una `#define`

2. Ampliare il programma del punto precedente per indicare quali liste otterranno dei seggi

3. Ampliare il programma del punto precedente per indicare quanti seggi otterrà ciascuna lista

4.1 Approccio alla soluzione

4.1.1 Gestire un numero imprecisato di liste

Non viene fornito nel testo dell'esercizio il numero esatto né il nome delle liste che partecipano alle elezioni. È un dato che è possibile ricavare dal file di input.

Per ogni riga letta in input dal file, si può verificare se tale voce è già nota o se essa è nuova e necessita di essere aggiunta alle liste note.

L'informazione sul numero massimo di liste consente la dichiarazione di un array per contenere i nomi delle liste e i rispettivi contatori.

```
#define MAXLISTE 10
#define MAXLEN 35 /* nomi delle liste */
int numListe = 0;
/* vettori i cui valori sono validi da 0 a numListe -
   per ogni nuova lista trovata si incrementa numListe
    */
int voti[MAXLISTE] = {0};
char nomi[MAXLISTE][MAXLEN];
```

[1] ad eccezione di Valle D'Aosta e Trentino-Alto Adige

4.1.2 Struct vs vettori paralleli

È possibile definire un tipo strutturato che contenga al suo interno tutte le informazioni necessarie a descrivere una lista. Il vettore sarà quindi unico e definito sul tipo strutturato. Di seguito si riporta l'esempio del punto 4.1.1, gestito con un tipo strutturato.

```
#define MAXLISTE 10
#define MAXLEN 35 /* nomi delle liste */

typedef struct {
  char nome[MAXLEN];
  int voti;
} t_lista;

int numListe = 0;

/* vettore i cui valori sono validi da 0 a numListe -
   per ogni nuova lista trovata si incrementa numListe
    */
t_lista risultati[MAXLISTE];
```

La soluzione è del tutto equivalente alla precedente, la quale faceva uso di due distinti vettori utilizzati in parallelo. Si usa dire che due vettori vengono utilizzati in parallelo quando sono definiti indipendentemente uno dall'altro e sono connessi solo logicamente.

In questo caso la connessione logica è data dal fatto che nella posizione `i` del primo vettore ci sia il nome della lista i cui voti sono nella posizione `i` dell'altro vettore. Questo è deducibile solo dall'uso che viene fatto di questi vettori nella logica del programma e non è esplicitato in nessun costrutto. L'utilizzo di vettori paralleli ha come svantaggio il fatto di dover gestire due vettori anziché uno solo: per ogni modifica (ad esempio ordinamento) a ciascun vettore, è necessario riflettere le modifiche sull'altro vettore per mantenere l'accoppiamento.

Questo problema si presta maggiormente ad essere risolto tramite la definizione e l'utilizzo di un tipo strutturato.

4.2 Soluzione

4.2.1 Vincitore delle elezioni

Listato 4: Election Simulator - punto 1

```
1  #include <stdio.h>
2  #include <string.h>
3
4  #define MAXLISTE 10
5  #define MAXLEN 35 /* nomi delle liste */
6  #define FILENAME "Elezioni.txt"
7  #define KEY_NOT_FOUND -1
8
9  typedef struct {
10   char nome[MAXLEN];
11   int voti;
12 } t_lista;
13
14 /* restituisce l'indice della stringa k in res, se non
       lo trova restituisce KEY_NOT_FOUND */
15 int find(char *k, t_lista res[], int nListe);
16
17 /* ordina res con ordinamento non crescente sul campo
      voti */
18 void sort(t_lista res[], int nListe);
19
20 int main() {
21   FILE* fin;
22   t_lista res[MAXLISTE];
23   int i, nListe, l, len, numVoti;
24   char bufin[MAXLEN];
25
26   fin = fopen(FILENAME, "r");
27   if (!fin) {
28     printf("\nErrore apertura file.");
29     return -1;
30   }
31
32   nListe = 0;
33   while (!feof(fin)) {
34       fgets(bufin, MAXLEN, fin);
35       /* fgets copia in bufin anche '\n' */
36       len = strlen(bufin);
37       if (len > 0 && bufin[len - 1] == '\n') {
38         bufin[len - 1] = '\0';
39       }
```

```
40        l = find(bufin, res, nListe);
41        if (l != KEY_NOT_FOUND) {
42          res[l].voti++;
43        } else {
44          res[nListe].voti = 1;
45          strcpy(res[nListe].nome, bufin);
46          nListe++;
47        }
48    }
49    fclose(fin);
50
51    sort(res, nListe);
52    numVoti = 0;
53    for (i = 0; i < nListe; i++) {
54      numVoti += res[i].voti;
55    }
56
57    if (nListe < 1) {
58      printf("Nessun voto trovato!\n");
59      return -2;
60    }
61    if (nListe < 2) {
62      printf("Allerta dittatura! Ha partecipato solo %s
            !\n", res[0].nome);
63    } else if ((float) res[0].voti / numVoti < 0.4f || (
        float) res[1].voti / numVoti >= 0.4) {
64      printf("Si andrà al ballotaggio tra %s e %s\n",
            res[0].nome, res[1].nome);
65    } else {
66      printf("Vince le elezioni al primo turno la lista
            %s\n", res[0].nome);
67    }
68    return 0;
69  }
70
71  int find(char *k, t_lista res[], int nListe) {
72    int i;
73    for (i = 0; i < nListe; i++) {
74      if (strcmp(res[i].nome, k) == 0) {
75        return i;
76      }
77    }
78    return KEY_NOT_FOUND;
79  }
80
81  void sort(t_lista res[], int nListe) {
```

```
82    t_lista tmp;
83    int i, j;
84    for (i = 0; i < nListe - 1; i++) {
85      for (j = i + 1; j < nListe; j++) {
86        if (res[i].voti < res[j].voti) {
87          tmp = res[i];
88          res[i] = res[j];
89          res[j] = tmp;
90        }
91      }
92    }
93  }
```

4.2.2 Soluzione punti 2 e 3

Di seguito viene riportato il programma modificato, limitato alle sole parti soggette a modifiche rispetto alla versione precedente. Sono quindi omesse le funzioni precedentemente definite.

Listato 5: Election Simulator - punto 2, punto3

```
1  #include <stdio.h>
2  #include <string.h>
3
4  #define MAXLISTE 10
5  #define MAXLEN 35 /* nomi delle liste */
6  #define KEY_NOT_FOUND -1
7  #define FILENAME "Elezioni.txt"
8  #define SEGGI_PROPORZ 277
9  #define SEGGI_MAGGIOR 340
10
11 typedef struct {
12   char nome[MAXLEN];
13   int voti;
14 } t_lista;
15
16 /* restituisce l'indice della stringa k in res, se non
       lo trova restituisce KEY_NOT_FOUND */
17 int find(char *k, t_lista res[], int nListe);
18
19 /* ordina res con ordinamento non crescente sul campo
     voti */
20 void sort(t_lista res[], int nListe);
21
22 /* trova la posizione dell'elemento massimo in un
     vettore di interi */
23 int pmax(int v[], int n);
24
```

```
25 int main() {
26   FILE* fin;
27   char bufin[MAXLEN];
28   t_lista res[MAXLISTE];
29   int i, nListe, l, len, nVoti, ballottaggio;
30   int seggi[MAXLISTE] = {0};
31   int resti[MAXLISTE] = {0};
32   int Q, seggiAssegnati, over3, votiMin;
33
34   fin = fopen(FILENAME, "r");
35   if (!fin) {
36     printf("\nErrore apertura file.");
37     return -1;
38   }
39
40   nListe = 0;
41   while (!feof(fin)) {
42       fgets(bufin, MAXLEN, fin);
43       /* fgets copia in bufin anche '\n' */
44       len = strlen(bufin);
45       if (len > 0 && bufin[len - 1] == '\n') {
46         bufin[len - 1] = '\0';
47       }
48       l = find(bufin, res, nListe);
49       if (l != KEY_NOT_FOUND) {
50         res[l].voti++;
51       } else {
52         res[nListe].voti = 1;
53         strcpy(res[nListe].nome, bufin);
54         nListe++;
55       }
56   }
57   fclose(fin);
58
59   sort(res, nListe);
60   nVoti = 0;
61   for (i = 0; i < nListe; i++) {
62     nVoti += res[i].voti;
63   }
64
65   if (nListe < 1) {
66     printf("Nessun voto trovato!\n");
67     return -2;
68   }
69
70   ballottaggio = 0;
```

```
71    if (nListe < 2) {
72      printf("Allerta dittatura! Ha partecipato solo %s
           !\n", res[0].nome);
73    } else if ((float) res[0].voti / nVoti < 0.4f || (
         float) res[1].voti / nVoti >= 0.4) {
74      printf("Si andrà al ballottaggio tra %s e %s\n",
           res[0].nome, res[1].nome);
75      ballottaggio = 1;
76    } else {
77      printf("Vince le elezioni al primo turno la lista
           %s\n", res[0].nome);
78    }
79
80    over3 = 0; /* Ottengono più del 3% */
81    votiMin = 0; /* Voti alla minoranza */
82    printf("Ottengono seggi le seguenti liste:\n");
83    for (i = 0; i < nListe; i++) {
84      if (res[i].voti >= 0.03 * nVoti) {
85        printf("%s\n", res[i].nome);
86        over3++;
87        votiMin += res[i].voti;
88      }
89    }
90
91    /* calcola il numero dei seggi solo in caso di esito
         certo delle elezioni */
92    if (!ballottaggio) {
93      /* quota maggioritaria */
94      seggi[0] = SEGGI_MAGGIOR;
95      /* la maggioranza non deve ottenere seggi nella
           fase dei resti */
96      resti[0] = -1;
97
98      /* quota proporzionale - fase quozienti */
99      seggiAssegnati = 0;
100     votiMin -= res[0].voti;
101     /* proporzione sui voti della minoranza */
102     Q = votiMin / SEGGI_PROPORZ;
103     for (i = 1; i < over3; i++) {
104       seggi[i] = res[i].voti / Q;
105       resti[i] = res[i].voti % Q;
106       seggiAssegnati += seggi[i];
107     }
108
109     /* quota proporzionale - fase resti */
110     while (seggiAssegnati < SEGGI_PROPORZ) {
```

```
111         /* trova resto max */
112         i = pmax(resti, over3);
113         /* passa al resto massimo immediatamente più
                piccolo */
114         resti[i] -= Q;
115         /* la maggioranza non deve ottenere seggi nella
                fase dei resti */
116         resti[0] -= Q;
117         /* assegna il seggio */
118         ++seggi[i];
119         ++seggiAssegnati;
120       }
121
122       printf("L'assegnazione del numero di seggi è la
             seguente:\n");
123       for (i = 0; i < nListe; ++i) {
124         printf("Lista %s: %d seggi\n", res[i].nome,
              seggi[i]);
125       }
126     }
127     return 0;
128   }
129
130   int pmax(int v[], int n) {
131     int pmax, i;
132     pmax = 0;
133     for (i = 1; i < n; ++i) {
134       if (v[i] > v[pmax]) {
135         pmax = i;
136       }
137     }
138     return pmax;
139   }
```

Licenza e crediti

Crediti

Quest'opera contiene elementi tratti da materiale di Gerardo Pelosi redatto per il corso di Fondamenti di Informatica per Ingegneria dell'Automazione a.a. 2014/15.

Licenza beerware[2]

Quest'opera è stata redatta da Stefano Cherubin. Mantenendo questa nota, puoi fare quello che vuoi con quest'opera. Se ci dovessimo incontrare e tu ritenessi che quest'opera lo valga, in cambio puoi offrirmi una birra.

[2] http://people.freebsd.org/~phk/