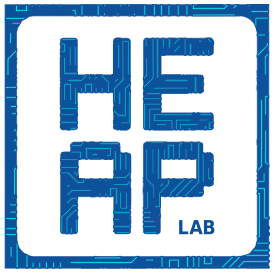
Orario dipendenti in C

Stefano Cherubin*



29/11/2019

[**Informatica A**] Esercitazione #17

corso per Ing. Gestionale a.a. 2019/20

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

Indice

1 Orario dipendenti

Si implementino le seguenti versioni di un sistema di conteggio ore lavorative per i dipendenti di una azienda.

1.1 Versione base

L'azienda dove lavorate ha di recente installato un nuovo sistema di controllo degli accessi nella sua sede principale. Ogni dipendente ha un badge che striscia su un rilevatore di accessi. Settimanalmente il rilevatore fornisce un file di testo che riporta gli accessi rilevati. Ogni riga del file corrisponde ad un passaggio di badge e si presenta come si seguito

```
hh mm id_badge
8 29 xfc052a8
```

Ogni dipendente ha un codice identificativo univoco salvato sul proprio badge; questo codice è composto da una stringa alfanumerica di 8 caratteri e viene registrato dal rilevatore.

Uno dei vostri superiori vi chiede una verifica sul numero di ore effettivamente lavorate nella passata settimana da uno specifico dipendente della vostra azienda (id: `xfc026b7`) perché sospettato di assenteismo.

Scrivete un programma che esegua la suddetta verifica in modo automatico.

1.2 Variante: controllo globale

Il vostro superiore ha realizzato che il programma che avete scritto al punto precedente può tornare utile per controllare le effettive ore lavorate da tutti i dipendenti della sede. Non vi ricordate quanti dipendenti esattamente ha l'azienda ma sapete che quelli che lavorano nella sede principale e utilizzano il nuovo sistema di controllo degli accessi sono in numero compreso tra 500 e 550.

Il brillante lavoro svolto con il controllo del dipendente assenteista vi ha resi automaticamente il massimo esperto dell'ufficio nella gestione del sistema di controllo degli accessi, è vostro compito implementare questo miglioramento.

Suggerimento: per semplificare l'output, si generi un file di testo che elenca, per ogni dipendente, il numero di ore effettivamente lavorate.

1.3 Variante: azienda in crescita

Alcuni giorni dopo aver consegnato il lavoro del punto precedente, sentite il vostro superiore di cui sopra che parla al telefono del nuovo sistema di controllo degli accessi e di come si potrebbe estendere a tutte le sedi dell'azienda il sistema sperimentato con successo nella sede principale.

Sapete già che vi toccherà rimettere mano al programma per portarlo su scala aziendale. Prevedete questa richiesta, e prevedete anche che il vostro superiore vi domanderà di svolgere questo lavoro entro domani mattina. Nel tentativo di non perdere la festa di compleanno del vostro migliore amico che avete in programma per questa serata, avete 40 minuti di tempo per modificare il programma del punto precedente in modo da:

- Gestire un numero imprecisato di dipendenti
- Ammettere che un dipendente possa andare in trasferta in una diversa sede dell'azienda

Suggerimento: si assuma possibile eseguire il programma su diversi file di input e/o accodare uno o più file di testo prima di eseguire il programma.

Si fornisca assieme alla soluzione in linguaggio C, una accurata descrizione delle assunzioni fatte.

1.4 Variante: rivolta sindacale

A seguito delle analisi che avete fornito, diversi dipendenti della vostra azienda, sparsi per le diverse sedi, sono stati raggiunti da provvedimenti disciplinari e, in alcuni casi, da lettere di licenziamento motivato da scarsa presenza sul luogo di lavoro. I sindacati si rivolgono a voi per confrontare le ore di lavoro effettive dei dipendenti raggiunti da provvedimenti con quelle dei loro superiori che hanno deciso di avviare la procedura.

Sapete che le prime 3 cifre del codice identificativo rappresentano il grado del dipendente, e che i responsabili di grado più alto hanno codice che inizia con `xfA`.

Si calcoli il numero di ore effettivamente lavorate da tutti i dipendenti di quel grado, se ne calcoli la media e si visualizzi a schermo inoltre il codice identificativo del dipendente tra questi che ha lavorato di meno.

1.5 Fine della storia

Dalle statistiche che avete scovato al punto precedente emerge che i più sfaticati tra tutti i lavoratori sono appunto i dirigenti che avevano deciso i licenziamenti.

Alla fine i sindacati e i dirigenti dell'azienda trovano un accordo e i provvedimenti disciplinari avviati in precedenza vengono ritirati. L'unico dipendente ad essere licenziato siete voi perché mentre stavate risolvendo questo esercizio vi siete scordati di passare il badge.

1.6 Soluzioni

1.6.1 Versione base

Listato 1: Orario dipendenti (base)

```
1  #include <stdio.h>
2  #include <string.h>
3  #define DIPENDENTE_MOLESTO "xfc026b7"
4
5  const char* nomeFile = "passaggi.txt";
6
7  int main() {
8    FILE* f;
9    int h, m, h_in, m_in, h_exit, m_exit;
10   char id[9];
11   int h_lav, m_lav, trovato;
12
13   f = fopen(nomeFile, "r");
14   if (!f) {
15     printf("Impossibile aprire il file\n");
16     return -1;
17   }
18
19   h_lav = 0;
20   m_lav = 0;
21   /* Per tutto il file */
22   do{
23
24     /* Orario Ingresso */
25     trovato = 0;
26     while (!feof(f) && !trovato) {
27       fscanf(f, "%d %d %s", &h, &m, id);
28
29       if (strcmp(id, DIPENDENTE_MOLESTO) == 0) {
30         h_in = h;
31         m_in = m;
32         trovato = 1;
33       }
34     }
35
36     /* Orario Uscita */
37     trovato = 0;
38     while (!feof(f) && !trovato) {
39       fscanf(f, "%d %d %s", &h, &m, id);
40
41       if (strcmp(id, DIPENDENTE_MOLESTO)==0) {
```

```
42          h_exit = h;
43          m_exit = m;
44          trovato = 1;
45
46          /* calcolo ore lavorative */
47          h_lav += h_exit - h_in;
48          m_lav += m_exit - m_in;
49
50          /* aggiusta conteggio minuti */
51          if (m_exit - m_in < 0) {
52            m_lav += 60;
53            h_lav -= 1;
54          }
55          if (m_lav >= 60) {
56            m_lav -= 60;
57            h_lav += 1;
58          }
59        }
60      }
61    } while (!feof(f));
62
63    printf("Il dipendete %s ha lavorato %d ore e %d
          minuti.\n", DIPENDENTE_MOLESTO, h_lav, m_lav);
64    return 0;
65  }
```

1.6.2 Controllo globale

Listato 2: Orario dipendenti (controllo globale)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #define MAX_EMPLOYEES 550
5  #define ID_LEN 9
6  #define ELEMENT_NOT_FOUND -1
7
8  const char* inputFileName="passaggi.txt";
9  const char* outputFileName="ore_effettive.txt";
10
11 typedef struct employee_s {
12   int hh;
13   int mm;
14   struct employee_s* next;
15 } employee_t;
16
17 typedef struct index_s {
18   char id[ID_LEN];
19   employee_t* entries;
20 } index_t;
21
22 index_t timeTable[MAX_EMPLOYEES];
23 int actual_N = 0;
24
25 int getIndexPosition(char* id);
26 int addEmployee(char* id);
27 int enqueue(int index, int h, int m);
28 void computeEmployee(int index, int* hh, int* mm);
29
30 int main() {
31   FILE* f_in;
32   FILE* f_out;
33   char id[ID_LEN];
34   int i, h, m;
35
36   f_in = fopen(inputFileName, "r");
37   if (!f_in) {
38   printf("Impossibile aprire file di input\n");
39     return -1;
40   }
41
42   fscanf(f_in, "%d %d %s", &h, &m, id);
```

```
43    while (!feof(f_in)) {
44
45      i = getIndexPosition(id);
46
47      if (i == ELEMENT_NOT_FOUND) {
48        i = addEmployee(id);
49      }
50
51      if (i == ELEMENT_NOT_FOUND) {
52        fclose(f_in);
53        return -1;
54      }
55
56      enqueue(i, h, m);
57
58      fscanf(f_in, "%d %d %s", &h, &m, id);
59    }
60    fclose(f_in);
61
62    f_out = fopen(outputFileName, "w");
63    if (!f_out) {
64      printf("Impossibile creare il file di output\n");
65      return -2;
66    }
67
68    for (i = 0; i < actual_N; i++) {
69      computeEmployee(i, &h, &m);
70      fprintf(f_out, "%s %d h %d m\n", timeTable[i].id,
          h, m);
71    }
72
73    fclose(f_out);
74    return 0;
75  }
76
77  int getIndexPosition(char* id) {
78    int i;
79    for (i = 0; i < actual_N; i++) {
80      if (strcmp(id, timeTable[i].id) == 0) {
81        return i;
82      }
83    }
84    return ELEMENT_NOT_FOUND;
85  }
86
87  int addEmployee(char* id) {
```

```
88    if (actual_N >= MAX_EMPLOYEES) {
89      printf("Troppi dipendenti nel file.\n");
90        return ELEMENT_NOT_FOUND;
91    }
92    timeTable[actual_N].entries = NULL;
93    strcpy(timeTable[actual_N].id, id);
94    actual_N++;
95    return actual_N - 1;
96  }
97
98  int enqueue(int index, int h, int m) {
99    employee_t* tmp = (employee_t*) malloc(sizeof(
          employee_t));
100   employee_t* aux = timeTable[index].entries;
101   if (!tmp) {
102     printf("Errore: impossibile allocare memoria\n");
103     return -1;
104   }
105   tmp->hh = h;
106   tmp->mm = m;
107   tmp->next = NULL;
108   if (aux == NULL) {
109     timeTable[index].entries = tmp;
110   } else {
111     while (aux->next != NULL) {
112       aux = aux->next;
113     }
114     aux->next = tmp;
115   }
116   return index;
117 }
118
119 void computeEmployee(int index, int* hh, int* mm) {
120   employee_t *in, *out;
121   *hh = 0;
122   *mm = 0;
123   in = timeTable[index].entries;
124   while (in != NULL) {
125     out = in->next;
126     if (out != NULL) {
127       *hh += out->hh - in->hh;
128       *mm += out->mm - in->mm;
129       /* adjust */
130       if (*mm >= 60) {
131         *mm -= 60;
132         *hh ++;
```

```
133         }
134         if (*mm < 0) {
135           *hh --;
136           *mm += 60;
137         }
138         in = out->next;
139       }
140     } /* end while */
141 }
```

1.6.3 Azienda in crescita

Listato 3: Orario dipendenti (Azienda in crescita)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #define ID_LEN 9
5  #define ELEMENT_NOT_FOUND NULL
6
7  const char* inputFileName = "passaggi.txt";
8  const char* outputFileName = "ore_effettive.txt";
9
10 typedef struct employee_s {
11   int hh;
12   int mm;
13   struct employee_s* next;
14 } employee_t;
15
16 typedef struct index_s {
17   char id[ID_LEN];
18   employee_t* entries;
19   struct index_s* next;
20 } index_t;
21
22 index_t* timeList;
23 index_t* getIndexRecord(char* id);
24 index_t* addEmployee(char* id);
25 index_t* enqueue(index_t* index, int h, int m);
26 void computeEmployee(index_t* index, int* hh, int* mm)
     ;
27
28 int main() {
29   FILE* f_in;
30   FILE* f_out;
31   char id[ID_LEN];
32   int h, m;
33   index_t* tmp;
34   f_in = fopen(inputFileName, "r");
35   if (!f_in) {
36     printf("Impossibile aprire il file di input\n");
37     return -1;
38   }
39   fscanf(f_in, "%d %d %s", &h, &m, id);
40   while (!feof(f_in)) {
41     tmp = getIndexRecord(id);
```

```
42      if (tmp == ELEMENT_NOT_FOUND) {
43        tmp = addEmployee(id);
44      }
45      if (tmp == ELEMENT_NOT_FOUND) {
46        fclose(f_in);
47        return -1;
48      }
49      enqueue(tmp, h, m);
50      fscanf(f_in, "%d %d %s", &h, &m, id);
51    }
52    fclose(f_in);
53    f_out = fopen(outputFileName, "w");
54    if (!f_out) {
55      printf("Impossibile creare il file di output\n");
56      return -2;
57    }
58    for (tmp = timeList; tmp != NULL; tmp = tmp->next) {
59      computeEmployee(tmp, &h, &m);
60      fprintf(f_out, "%s %d h %d m\n", tmp->id, h, m);
61    }
62    fclose(f_out);
63    return 0;
64  }
65
66  index_t* getIndexRecord(char* id) {
67    index_t* tmp;
68    for (tmp = timeList; tmp != NULL; tmp = tmp->next) {
69      if (strcmp(id, tmp->id) == 0) {
70        return tmp;
71      }
72    }
73    return ELEMENT_NOT_FOUND;
74  }
75
76  index_t* addEmployee(char* id) {
77    index_t* tmp;
78    index_t* prev;
79    tmp = (index_t*) malloc(sizeof(index_t));
80    if (!tmp) {
81      printf("Errore: impossibile allocare memoria\n");
82      return ELEMENT_NOT_FOUND;
83    }
84    strcpy(tmp->id, id);
85    tmp->next = NULL;
86    tmp->entries = NULL;
87    prev = timeList;
```

```
88    if (prev) {
89      while (prev->next) {
90        prev = prev->next;
91      }
92      prev->next = tmp;
93    } else {
94      timeList = tmp;
95    }
96    return tmp;
97  }
98
99  index_t* enqueue(index_t* index, int h, int m) {
100   employee_t* tmp = (employee_t*) malloc(sizeof(
        employee_t));
101   employee_t* aux = index->entries;
102   if (!tmp) {
103     printf("Errore: impossibile allocare memoria\n");
104     return ELEMENT_NOT_FOUND;
105   }
106   tmp->hh = h;
107   tmp->mm = m;
108   tmp->next = NULL;
109   if (aux == NULL) {
110     index->entries = tmp;
111   } else {
112     while (aux->next != NULL) {
113       aux = aux->next;
114     }
115     aux->next = tmp;
116   }
117   return index;
118 }
119
120 void computeEmployee(index_t* index, int* hh, int* mm)
      {
121   employee_t *in, *out;
122   *hh = 0;
123   *mm = 0;
124   in = index->entries;
125   while (in != NULL) {
126     out = in->next;
127     if (out != NULL) {
128       *hh += out->hh - in->hh;
129       *mm += out->mm - in->mm;
130       /* adjust */
131       if (*mm >= 60) {
```

```
132           *mm -= 60;
133           *hh ++;
134         }
135         if (*mm < 0) {
136           *hh --;
137           *mm += 60;
138         }
139         in = out->next;
140       }
141     } /* end while */
142   }
```

1.6.4 Rivolta sindacale

Listato 4: Orario dipendenti (Rivolta sindacale)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define ID_LEN 9
#define ELEMENT_NOT_FOUND NULL

const char* inputFileName = "passaggi.txt";
const char* outputFileName = "ore_effettive.txt";
const char* prefix = "xfA";

typedef struct employee_s {
  int hh;
  int mm;
  struct employee_s* next;
} employee_t;

typedef struct index_s {
  char id[ID_LEN];
  employee_t* entries;
  struct index_s* next;
} index_t;

index_t* timeList;
index_t* getIndexRecord(char* id);
index_t* addEmployee(char* id);
index_t* enqueue(index_t* index, int h, int m);
void computeEmployee(index_t* index, int* hh, int* mm)
    ;
int startsWith(const char* baseStr, const char* subStr
    );

int main() {
  FILE* f_in;
  FILE* f_out;
  char id[ID_LEN];
  char id_min[ID_LEN];
  int h, m, count;
  float avg, min;
  index_t* tmp;
  f_in = fopen(inputFileName, "r");
  if (!f_in) {
    printf("Impossibile aprire il file di input\n");
```

```
41      return -1;
42    }
43    fscanf(f_in, "%d %d %s", &h, &m, id);
44    while (!feof(f_in)) {
45      if (startsWith(id, prefix) != 0) {
46        tmp = getIndexRecord(id);
47        if (tmp == ELEMENT_NOT_FOUND) {
48          tmp = addEmployee(id);
49        }
50        if (tmp == ELEMENT_NOT_FOUND) {
51          fclose(f_in);
52          return -1;
53        }
54        enqueue(tmp, h, m);
55      }
56      fscanf(f_in, "%d %d %s", &h, &m, id);
57    }
58    fclose(f_in);
59    f_out = fopen(outputFileName, "w");
60    if (!f_out) {
61      printf("Impossibile creare il file di output\n");
62      return -2;
63    }
64    min = 0;
65    strcpy(id_min, "");
66    avg = 0;
67    count = 0;
68    for (tmp = timeList; tmp != NULL; tmp = tmp->next) {
69      computeEmployee(tmp, &h, &m);
70      fprintf(f_out, "%s %d h %d m\n", tmp->id, h, m);
71        if (h * 60 + m < min) {
72        min = h * 60 + m;
73        strcpy(id_min, id);
74      }
75      count ++;
76    }
77    fclose(f_out);
78    if (count != 0) {
79      avg = avg / count;
80    }
81    printf("%s ha lavorato solo %d ore e %d minuti.\n",
         id_min, (int) min / 60, (int) min % 60);
82    printf("In media hanno lavorato %d ore e %d minuti\n
         ", (int) avg / 60, (int) avg % 60);
83    return 0;
84  }
```

```
85
86  index_t* getIndexRecord(char* id) {
87    index_t* tmp;
88    for (tmp = timeList; tmp != NULL; tmp = tmp->next) {
89      if (strcmp(id, tmp->id) == 0) {
90        return tmp;
91      }
92    }
93    return ELEMENT_NOT_FOUND;
94  }
95
96  index_t* addEmployee(char* id) {
97    index_t* tmp;
98    index_t* prev;
99    tmp = (index_t*) malloc(sizeof(index_t));
100   if (!tmp) {
101     printf("Errore: impossibile allocare memoria\n");
102     return ELEMENT_NOT_FOUND;
103   }
104   strcpy(tmp->id, id);
105   tmp->next = NULL;
106   tmp->entries = NULL;
107   prev = timeList;
108   if (prev) {
109     while (prev->next) {
110       prev = prev->next;
111     }
112     prev->next = tmp;
113   } else {
114     timeList = tmp;
115   }
116   return tmp;
117 }
118
119 index_t* enqueue(index_t* index, int h, int m) {
120   employee_t* tmp = (employee_t*) malloc(sizeof(
        employee_t));
121   employee_t* aux = index->entries;
122   if (!tmp) {
123     printf("Errore: impossibile allocare memoria\n");
124     return ELEMENT_NOT_FOUND;
125   }
126   tmp->hh = h;
127   tmp->mm = m;
128   tmp->next = NULL;
129   if (aux == NULL) {
```

```
130       index->entries = tmp;
131     } else {
132       while (aux->next != NULL) {
133         aux = aux->next;
134       }
135       aux->next = tmp;
136     }
137     return index;
138   }
139
140   void computeEmployee(index_t* index, int* hh, int* mm)
          {
141     employee_t *in, *out;
142     *hh = 0;
143     *mm = 0;
144     in = index->entries;
145     while (in != NULL) {
146       out = in->next;
147       if (out != NULL) {
148         *hh += out->hh - in->hh;
149         *mm += out->mm - in->mm;
150         /* adjust */
151         if (*mm >= 60) {
152           *mm -= 60;
153           *hh ++;
154         }
155         if (*mm < 0) {
156           *hh --;
157           *mm += 60;
158         }
159         in = out->next;
160       }
161     } /* end while */
162   }
163
164   int startsWith(const char* baseStr, const char* subStr
         ) {
165     int i, len;
166     len = strlen(subStr);
167     if (strlen(baseStr) < len) {
168       return 0;
169     }
170     for (i = 0; i < len; i++) {
171       if (baseStr[i] != subStr[i]) {
172         return 0;
173       }
```

```
174     }
175     return 1;
176  }
```

Licenza e crediti

Licenza beerware[1]

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.

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