Laboratorio di programmazione — Corso avanzato

24 gennaio 2008

Rubrica del telefono

Scopo di questo esercizio è implementare una rudimentale rubrica del telefono.

Il programma deve leggere da *standard input* un elenco di coppie cognome/numero di telefono e in seguito, dato un cognome (come parametro sulla linea di comando), riportare il numero di telefono corrispondente.

Il formato del file di testo è esemplificato dalla seguente rubrica

```
3
Santini 0250316259
Rossi 023344567
BusH 0015552349876
```

Come notate, sulla prima riga c'è un numero intero, che rappresenta il numero di righe seguenti, ciascuna delle quali contiene un cognome, uno spazio e quindi un numero di telefono.

Osservate che non c'è un limite prefissato al numero di righe della rubrica; ovviamente, qualora tale numero fosse troppo grande per la memoria a disposizione, il programma dovrà terminare segnalando un errore. Potete invece fare affidamento sul fatto che cognomi e numeri di telefono non siano mai più lunghi di 256 caratteri.

Supponendo che tale rubrica sia nel file `rubrica.txt` e che il programma che avete sviluppato si chiami `rubrica`, un esempio di esecuzione è il seguente

```
$./rubrica Santini < rubrica.txt
Il numero di Santini è 0250316259
$./rubrica Paperino < rubrica.txt
Il numero di Paperino non è presente nella rubrica
$./rubrica Bush < rubrica.txt
il numero di BusH è 0015552349876
```

Osservate che il programma deve memorizzare i cognomi esattamente come scritti nella rubrica, ma effettuare le ricerche a meno di maisucole/minuscole.

Suggerimenti

Rappresentate le coppie cognome/numero di telefono con una struttura (che denominerete come `struct coppia`) con due membri stringa (di nome `cognome` e `numero`).

Scrivete una funzione che legga la rubrica in un array di tali strutture, attenzione: dovrete usare `malloc` per allocare in modo dinamico la memoria necessaria, a seconda del numero di righe indicato come prima riga della rubrica.

Dopo aver letto la rubrica, ordinatela alfabeticamente con la funzione `qsort` e utilizzate la funzione `bsearch` per effettuare la ricerca (usate il comando `man` per scoprire quali sono gli argomenti di entrambe).

Procedete per passi: dopo aver scritto la parte di ordinamento, per verificare il corretto funzionamento di quanto avete

implementato, create qualche rubrica di esempio e provate a leggerla, ordinarla e stamaprla. Passate ad implementare la parte di ricerca solo dopo che vi siete accertati che il codice per la parte di ordinamento è corretto.

Le funzioni di comparazione. Per svolgere l'esercizio dovete implementare due funzioni di comparazione, richieste da `qsort` e `bsearch`.

Si tratta di un compito non banale, che richiede buona dimestichezza coi puntatori. Se non riuscite a cavarvela da soli, seguite i suggerimenti che seguono. Come al solito sono "incrementali": leggete il suggerimento, se non vi chiarisce le idee leggete il codice, studiatelo e cercate di capirlo, quindi integratelo nel vostro codice.

L'ingrediente fondamentale è una funzione che compari le stringhe a meno di maiuscole/minuscole. Potete implementarla usando la funzione `toupper` per eliminare la distinzione tra maiuscole e minuscole, come nell'esempio seguente.

```
int compara( char *x, char *y ) {
        while ( *x && *y && toupper( *x ) == toupper( *y ) ) {
                x++;
                y++;
        }

    return toupper( *x ) - toupper( *y );
}
```

Usando la funzione `compara` potete ora scrivere le due funzioni di comparazione che si attendono come parametro sia `qsort` che `bsearch` (le chiameremo rispettivamente `qsort_compara` e `bsearch_compara`). Fate attenzione: sebbene il prototipo sia uguale nei due casi, la funzione necessaria all'ordinamento deve confrontare due strutture, mentre quella necessaria alla ricerca deve confrontare una stringa (quello che nel manuale è chiamato *key*) e una struttura.

Se non siete in grado di implementare le due funzioni, trovate di seguito una possibile soluzione:

```
int qsort_compara( const void *px, const void *py ) {
        struct coppia *x, *y;

        x = (struct coppia *)px;
        y = (struct coppia *)py;

        return str_compara( x->cognome, y->cognome );
}

int bsearch_compara( const void *px, const void *py ) {
        char *x;
        struct coppia *y;

        x = (char *)px;
        y = (struct coppia *)py;

        return str_compara( x, y->cognome );
}
```