Esercizi di programmazione in linguaggio C – English Dictionary

Il file di testo *wordnet.txt* contiene un certo numero di parole (*word*) e il corrispondente significato (*meaning*) secondo il seguente formato:

```
word1 * meaning1 \n
word2 * meaning2 \n
...
```

Le parole sono sequenze di caratteri albabetici maiuscoli prive di *whitespace*, mentre i significati possono contenere spazi o tabulazioni. Ogni parola è memorizzata in una linea insieme al suo significato e separata da esso con la sequenza " * ". I significati terminano con un carattere di *newline*.

1. **elenco-10** – Scrivere un programma che visualizzi le prime dieci linee del file di testo.
2. **stampa-parole** – Scrivere un programma che stampi tutte le parole contenute nel file senza memorizzarle in un vettore.
3. **vettore-s** – Scrivere un programma che memorizzi le sole parole in un vettore di stringhe allocato staticamente e, successivamente, indichi il conteggio delle voci presenti nel dizionario (si ipotizzi che il dizionario non contenga più di 100.000 parole e che le parole non siano più lunghe di 50). Si riportino infine le dimensioni della tabella (fisica e logica) espresse in byte.
4. **vettore-d** ★ Ripetere l'esercizio precedente utilizzando un vettore a elementi statici allocato dinamicamente.
5. **vettore-dd** ★ Ripetere l'esercizio **vettore-d** utilizzando un vettore a elementi dinamici allocato dinamicamente.
6. **elenco-ordinato** – Scrivere un programma che memorizzi le sole parole in un vettore di stringhe e stampi l'elenco dei vocaboli in ordine alfabetico.
7. **statistica-s** – Scrivere un programma che dichiari una struttura adatta a memorizzare i dati relativi a una parola e memorizzi il contenuto del dizionario in una tabella statica (si ipotizzi che il dizionario non contenga più di 100.000 parole e che le parole e i significati non siano più lunghi rispettivamente di 50 e 1500 caratteri). Successivamente, il programma deve calcolare e visualizzare le seguenti informazioni: a) numero totale di parole; b) lunghezze minima, media e massima delle parole; c) lunghezze minima, media e massima dei significati.
8. **statistica-d** – Ripetere l'esercizio precedente utilizzando una tabella a campi statici allocata dinamicamente.
9. **statistica-dd** – Ripetere l'esercizio precedente utilizzando una tabella a campi dinamici allocata dinamicamente.
10. **ricerca-parola1** – Scrivere un programma che, dopo aver memorizzato il dizionario in un'apposita tabella, chieda all'utente di inserire una parola maiuscola da tastiera, la ricerchi nella tabella e riporti il suo significato (oppure un messaggio d'errore nel caso in cui la parola non sia inclusa nel dizionario). Si ricorda che, all'interno del file, le parole non sono memorizzate in ordine alfabetico.
11. **ricerca-parola2** – Ripetere l'esercizio precedente con le seguenti varianti: la parola immessa è composta anche da caratteri minuscoli; il programma, dopo aver visualizzato un significato, richiede l'immissione di un'altra parola e termina quando l'utente immette una stringa vuota.
12. **ottimizzazione** – Ottimizzare l'esercizio **ricerca-parola2** affinché la ricerca sia efficiente in tempo.

Soluzioni degli esercizi

Esercizio n. 4 (vettore-d)

```
/*
 *        File: vettore-d.c
 *
 *        Memorizza le sole parole in una tabella allocata dinamicamente.
 *        I campi della tabella hanno dimensione prefissata.
 */

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

#define MAX_PAROLA 51

struct record {
       char parola[MAX_PAROLA];
};

int main( void ) {

       FILE *fp;
       struct record *tabella = NULL;      // Equivale a un puntatore
                                           //a vettore di record
       char buffer[MAX_PAROLA];
       int num_rec = 0;
       int dim_fisica, dim_logica;

       if ((fp = fopen("..\\wordnet.txt", "r")) == NULL) {
              printf("Errore durante l'apertura del dizionario.\n");
              return 1;
       }

       while (fscanf(fp, "%s", buffer) > 0) {
              // Estendo di un elemento il vettore di parole
              tabella = (struct record *) realloc(tabella,
                     (num_rec+1) * sizeof(struct record));

              strcpy(tabella[num_rec].parola, buffer);

              while (fgetc(fp) != '\n');      // Legge e scarta i caratteri
                                              // del significato, continuando finche'
                                              // il carattere e' diverso da newline
              num_rec++;
       }
```

```
        fclose(fp);

        printf("Numero di parole presenti nel dizionario: %d\n", num_rec);

        dim_logica = num_rec*MAX_PAROLA*sizeof(char);
        dim_fisica = dim_logica;

        printf("\nDimensioni del vettore\n==================\n");
        printf("Logica: %6d byte\n", dim_logica);
        printf("Fisica: %6d byte\n", dim_fisica);

        printf("Percentuale di utilizzo della memoria: %.1f%%\n",
               100.0 * (float) dim_logica / dim_fisica);

        free(tabella);

        return 0;

}
```

Esercizio n. 5 (vettore-dd)

```
/*
 *        File: vettore-dd.c
 *
 *        Memorizza le sole parole in una tabella allocata dinamicamente.
 *        I campi della tabella sono dinamici (lunghezza variabile).
 */

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

#define MAX_BUFFER 51

struct record {
       char *parola;
};

int main( void ) {

       FILE *fp;
       struct record *tabella = NULL;      // Equivale a un puntatore
                                           // a vettore di record
       char buffer[MAX_BUFFER];
       int num_rec = 0;
       int dim_fisica = 0, dim_logica = 0;

       if ((fp = fopen("..\\wordnet.txt", "r")) == NULL) {
              printf("Errore durante l'apertura del dizionario.\n");
              return 1;
       }

       while (fscanf(fp, "%s", buffer) > 0) {
              // Estendo di un elemento il vettore di parole
              tabella = (struct record *) realloc(tabella,
                     (num_rec+1) * sizeof(struct record));

              // Alloco spazio in cui copiare la parola presente nel buffer
              tabella[num_rec].parola = (char *) malloc(
                     (strlen(buffer)+1) * sizeof(char));

              strcpy(tabella[num_rec].parola, buffer);

              while (fgetc(fp) != '\n');      // Legge e scarta i caratteri del
                                              // significato, continuando finche'
                                              // il carattere e' diverso da newline
              num_rec++;
```

```
            dim_logica += sizeof(struct record) + (strlen(buffer)+1) * sizeof(char);
            dim_fisica += sizeof(struct record) + (strlen(buffer)+1) * sizeof(char);
        }

        fclose(fp);

        printf("Numero di parole presenti nel dizionario: %d\n", num_rec);

        printf("\nDimensioni del vettore\n==================\n");
        printf("Logica: %6d byte\n", dim_logica);
        printf("Fisica: %6d byte\n", dim_fisica);

        printf("Percentuale di utilizzo della memoria: %.1f%%\n",
            100.0 * (float) dim_logica / dim_fisica);

        for (int i = 0; i < num_rec; i++)
            free(tabella[i].parola);

        free(tabella);

        return 0;
}
```