Università di Ferrara
Dipartimento di Ingegneria

Esercitazioni di

FONDAMENTI DI INFORMATICA MODULO B

ESERCITAZIONE 3

Tutor

Arnaud Nguembang Fadja: ngmrnd@unife.it
Damiano Azzolini: damiano.azzolini@student.unife.it

Esercizio 1. Tavola ordinata in memoria centrale

Si deve sviluppare un programma che realizzi una rubrica telefonica, implementata come tavola in memoria centrale, **strutturata su più file**. In particolare, ogni elemento della tavola è caratterizzato dalle seguenti informazioni:

- Cognome (campo chiave primaria)
- Nome (campo chiave secondario)
- Telefono

Il programma deve essere in grado di gestire varie richieste dell'utente:

- **inserimento ordinato**: l'utente vuole inserire un nuovo record nell'archivio. Dati cognome, nome e numero di telefono della persona da inserire, il programma aggiunge un nuovo elemento all'archivio. Ogni volta che inserisco un elemento devo riordinare la tavola con la funzione **qsort**.
- **stampa**: l'utente richiede la stampa dell'intera rubrica.
- **uscita**: l'utente richiede che il programma termini.

L'utente può scegliere sia l'ordine che il numero delle richieste da effettuare, prima di terminare l'esecuzione.

Esercizio 1. Tavola ordinata in memoria centrale

L'interazione tra l'utente e il programma avviene in modo ciclico: l'utente può sottoporre una richiesta ad ogni ciclo ed il programma, sfruttando un meccanismo di selezione (switch), reagisce nel modo richiesto. L'esecuzione del programma si conclude quando l'utente richiede l'uscita. **(Creazione di un menù testuale)**

La rubrica telefonica viene presentata come una tavola inizializzata in memoria centrale ordinata sulla chiave (array ordinato in base al cognome e, in caso di uguaglianza, al nome).

Esercizio 1. Tavola ordinata in memoria centrale

Si utilizzi l'algoritmo di ordinamento **qsort** per effettuare l'inserimento ordinato.

void qsort (void ****base*, size_t** *num*, **size_t** *width*, **int** (****compare***)(const void ****elem1***, const void ****elem2** **));**

- **base** = puntatore ad un vettore di elementi
- **num** = numero di elementi del vettore
- **width** = dimensione di ciascun elemento del vettore
- **compare** = nome o identificativo di una funzione (o riferimento ad una funzione) che effettua il confronto tra due elementi del vettore considerato; come risultato del confronto deve restituire un valore intero; come parametri di ingresso deve ricevere il riferimento dei due elementi da confrontare.

Consultare l'help di Visual Studio per ulteriori informazioni.

Esercizio 1. Tavola ordinata in memoria centrale

- Gli elementi della tavola dovranno avere la seguente struttura:

```
typedef struct {
        char cognome[20];
        char nome[20];
        char tel[16];
} elemento;
// Definizione della struttura di un elemento della tavola

typedef elemento rubrica[DIM];
// Definizione del tipo rubrica
// che equivale ad un array di dati di tipo elemento
```

- Implementare tutte le funzioni necessarie al corretto funzionamento del programma.

Esercizio 1. Tavola ordinata in memoria centrale

Si richiede, in particolare, di:

- implementare la funzione di ***inserimento ordinato*** per la funzionalità di inserimento presente nelle specifiche. La funzione richiede all'utente l'inserimento dei dati, aggiunge all'array i nuovi dati e richiama **qsort** per riordinare l'array:

```
int inserimento_ord(rubrica R, int n);
//n=dimensione logica di R
```

- definire una opportuna funzione di confronto **compare** per utilizzare la funzione **qsort**:

```
int compare (const void *e1, const void *e2);
//deve effettuare il confronto sulla chiave
```

Esercizio 1. Tavola ordinata in memoria centrale

Inizializzare la rubrica con il seguente array di persone:

```
{
                    "Grissom","Cyrus","555 01071967",
                "Krueger","Frederick","555 14051945",
                 "Lecter","Hannibal","555 02081950",
                  "Malfoi","Draco","555 23111980",
                  "Malfoi","Lucius","555 12051965",
                   "Price","Elijah","555 30091970",
                 "Voorhees","Jason","555 13061946",
                "Voorhees","Pamela","555 22081930"
}
```

Esercizio 2. Tavola ordinata in memoria centrale

Partendo dal programma precedente, si aggiunga la seguente funzione:

- **ricerca**: dato in ingresso cognome e nome di una persona presente in archivio, si richiede la visualizzazione del numero di telefono relativo alla persona data. Questa funzione **DEVE** sfruttare la ricerca binaria.

Come per la parte 1, l'utente può scegliere l'ordine con cui effettuare le richieste e il numero di richieste da fare prima di terminare l'esecuzione.

Strutturare il programma in più file.

Esercizio 2. Tavola ordinata in memoria centrale

- Ricordiamo che gli elementi della tavola presentano la seguente struttura:

```
typedef struct {
        char cognome[20];
        char nome[20];
        char tel[16];
} elemento;
// Definizione della struttura di un elemento della tavola

typedef elemento rubrica[DIM];
// Definizione del tipo rubrica
// che equivale ad un array di dati di tipo elemento
```

- Implementare tutte le funzioni necessarie al corretto funzionamento della ricerca.

Esercizio 2. Tavola ordinata in memoria centrale

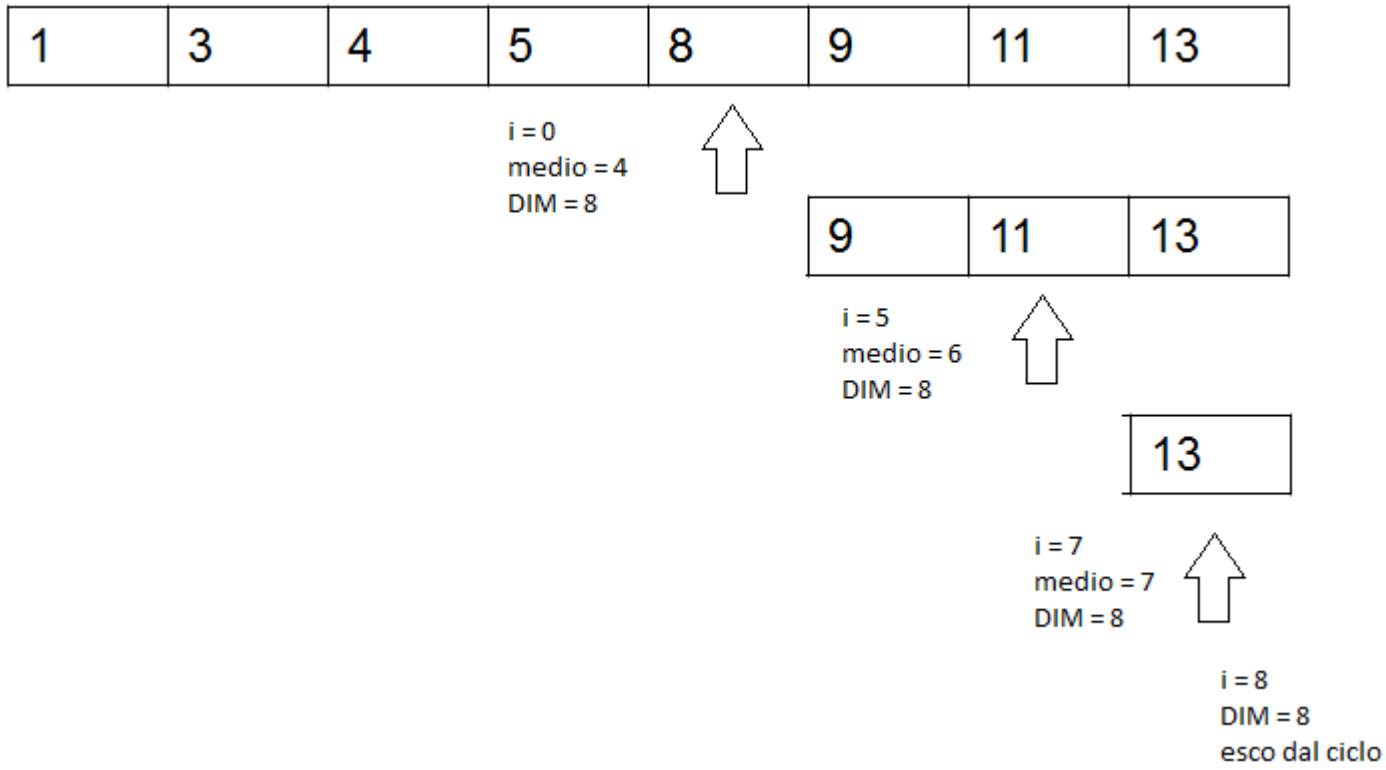
Si richiede, in particolare, di implementare la funzione di **ricerca binaria** per la funzionalità di ricerca presente nelle specifiche, essendo la tavola ordinata:

```
int individua_binaria (rubrica R, int n, elemento e);
//cerca l'elemento e nella rubrica contenente n elementi
```

Implementata la funzione di ricerca, come si può modificare la funzione di inserimento per evitare di inserire due volte la stessa entry? Fornirne un'implementazione.

Esercizio 2. Tavola ordinata in memoria centrale

Esempio: cerchiamo il valore 14 nel seguente array utilizzando il metodo della ricerca binaria.



Esercizio 3. Tavola ordinata su file di testo

Creare una seconda versione dell'esercizio precedente nella quale la rubrica è gestita tramite file (il programma dovrà quindi gestire la tavola mediante l'utilizzo di un file). Il programma, oltre alle funzionalità già viste, dovrà:

1. All'avvio, inizializzare la tavola leggendola da file (passare il nome del file come **ARGOMENTO**)
2. Poter azzerare la rubrica creando un nuovo file
3. Alla chiusura, salvare la tavola nel file
4. Salvare, tramite funzione nel menù, lo stato della tavola

NB: notare la differenza tra i punti 3 e 4. Col punto 3, si vuole implementare una funzione che, quando si decide di terminare l'applicazione, salvi lo stato della rubrica. Col punto 4, invece, si vuole aver la possibilità di salvare lo stato della rubrica in un punto qualsiasi del programma, non necessariamente alla fine.

Strutturare il programma **in più file**.

Esercizio 3. Tavola ordinata su file di testo

Creare una seconda versione dell'esercizio precedente (sempre **strutturato su più file**) nella quale la rubrica è gestita tramite file (il programma dovrà quindi gestire la tavola mediante l'utilizzo di un file). Il programma, oltre alle funzionalità già viste, dovrà:

1. All'avvio, inizializzare la tavola leggendola da file (passare il nome del file come **ARGOMENTO**)
2. Poter azzerare la rubrica creando un nuovo file
3. Alla chiusura, salvare la tavola nel file
4. Salvare, tramite funzione nel menù, lo stato della tavola

NB: notare la differenza tra i punti 3 e 4. Col punto 3, si vuole implementare una funzione che, quando si decide di terminare l'applicazione, salvi lo stato della rubrica. Col punto 4, invece, si vuole aver la possibilità di salvare lo stato della rubrica in un punto qualsiasi del programma, non necessariamente alla fine.

Suggerimento: una volta letta da file, la tavola si trova in memoria centrale, quindi è possibile riutilizzare le funzioni già implementate.

Esercizio 4. Tavola ordinata su file di binario

Si supponga che la rubrica sia stata salvata in un file **binario** (passato come **ARGOMENTO**), il cui contenuto è troppo grande per poter essere mantenuto in memoria centrale. Il programma dovrà quindi gestire la tavola operando **direttamente sul file** (operazioni di inserimento ordinato e ricerca binaria), in particolare dovrà:

1. Poter azzerare la rubrica creando un nuovo file
2. Stampare l'intera rubrica.
3. Inserire in maniera ordinata un nuovo elemento.
4. Cercare un elemento all'interno della rubrica **(usare ricerca binaria)**.
5. Uscire.

NB: non deve essere utilizzata la funzione qsort ad ogni inserimento.

SUGGERIMENTO: usare la funzione **fseek** per la ricerca.

Strutturare il programma in più file fin da subito.

ACCESSO DIRETTO AI FILE

```
int fseek(FILE *f, long offset, int origin)
```

Sposta la testina del file di *offset* byte a partire da *origin*

Nel secondo parametro può essere utile usare la dimensione dei record per spostarsi (**`sizeof(record)`**)

Ad esempio per spostarsi all'inizio del terzo record:

```
fseek(f,sizeof(record)*2,SEEK_SET)
```

origin può valere:

0 (SEEK_SET): inizio file

1 (SEEK_CUR): posizione corrente

2 (SEEK_END): fine file

fseek restituisce **0** se lo spostamento ha **successo**, altrimenti un numero diverso da 0

ACCESSO DIRETTO AI FILE

```
void rewind(FILE *f)
```

Posiziona la testina all'inizio del file

```
long ftell(FILE *f)
```

Restituisce la posizione del byte su cui si trova la testina,

-1 in caso di errore

Tutte queste funzioni possono essere usate ***sia per i file binari che per i file di testo*** !