Università degli Studi di Ferrara

DE: Department of Engineering Ferrara

Esercitazioni di

FONDAMENTI DI INFORMATICA MODULO B

ESERCITAZIONE 3

Azzolini Damiano: damiano.azzolini@unife.it
Bertagnon Alessandro: alessandro.bertagnon@unife.it

Esercizio 1. Tavola ordinata in memoria centrale

Si deve sviluppare un programma che realizzi una rubrica telefonica. In particolare, ogni elemento della rubrica è caratterizzato dalle seguenti informazioni:

- **Cognome** (campo chiave primaria)
- **Nome** (campo chiave secondaria)
- **Telefono**

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

Il progetto deve essere opportunamente strutturato su più file.

Esercizio 1. Tavola ordinata in memoria centrale

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

- **inserimento**: l'utente vuole inserire un nuovo record nella rubrica. Dati cognome, nome e numero di telefono della persona da inserire, il programma aggiunge un nuovo elemento all'archivio. Poi riordina la rubrica (qsort) in modo da mantenerla ordinata rispetto alla chiave.
- **stampa**: l'utente richiede la stampa dell'intera rubrica.
- **uscita**: l'utente richiede che il programma termini.

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)**

```
Scegli l'operazione:
    1   Inserimento
    2   Stampa
    3   Uscita


Scelta:
```

Esercizio 1. Tavola ordinata in memoria centrale

Gli elementi della tavola dovranno avere la seguente struttura:

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


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

Esercizio 1. Tavola ordinata in memoria centrale

Si richiede, in particolare, di:

- implementare la funzione di **inserimento**. La funzione richiede all'utente l'inserimento dei dati di **un** nuovo elemento, aggiunge all'array il nuovo elemento (in coda) e chiama **qsort** per riordinare l'array:

  int inserimento (rubrica R, int n);

  inserisce un nuovo elemento, letto da tastiera, nella rubrica **R** (contenente **n** elementi) e restituisce il numero di elementi in **R** al termine dell'inserimento.

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

  int compare (const void *e1, const void *e2);

  deve effettuare un confronto sulla chiave (quindi sul cognome e a parità di cognome sul nome).

Esercizio 1. Tavola ordinata in memoria centrale

Si utilizzi l'algoritmo di ordinamento **qsort** all'interno della funzione inserimento.

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

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 funzionalità va aggiunta al menù testuale, già realizzato nella parte 1, in modo che l'utente possa scegliere l'ordine con cui effettuare le richieste e il numero di richieste da fare prima di terminare l'esecuzione.

Esercizio 2. Tavola ordinata in memoria centrale

Implementare la funzione di **individua_binaria** corrispondente alla funzionalità di ricerca richiesta nelle specifiche. La funzione **individua_binaria** DEVE utilizzare come algoritmo di ricerca la **RICERCA BINARIA** essendo la tavola ordinata:

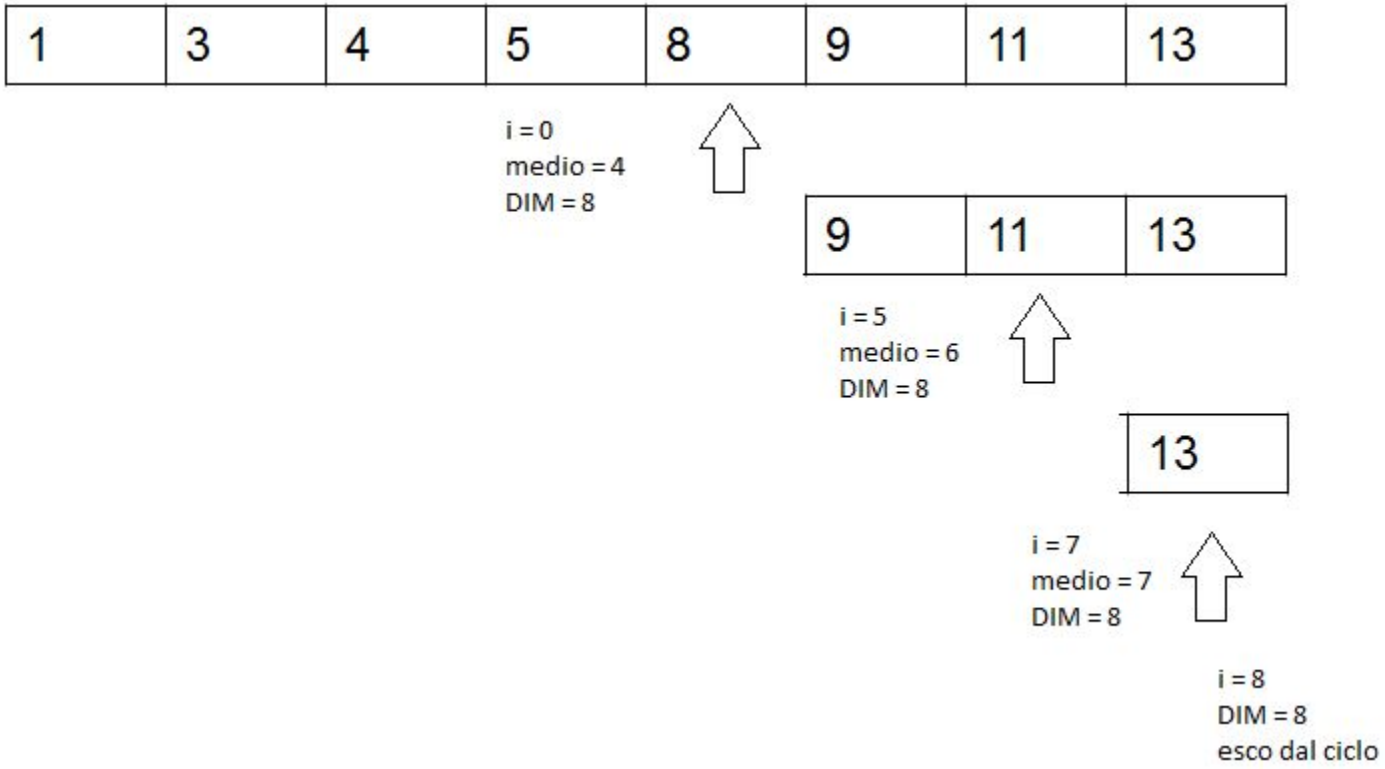
int individua_binaria (rubrica R, int n, elemento e);

cerca l'elemento **e** nella rubrica **R** contenente **n** elementi e restituisce l'indice dove si trova l'elemento **e** se presente (se l'elemento non è presente restituisce **n**)

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 binario

Si supponga che la rubrica sia stata salvata in un file **binario** (passato come **argomento** da linea di comando), 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 (utilizzare l'**inserimento ordinato** e non la funzione qsort)
4. Cercare un elemento all'interno della rubrica (utilizzare la **ricerca binaria**), ovvero dati nome e cognome restituire il numero di telefono.
5. Uscire.

SUGGERIMENTO: usare le funzioni di accesso diretto ai file e in particolare la **fseek** per la ricerca.

Strutturare il programma su più file.

Esercizio 3. 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

Esercizio 3. 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

Per maggiori informazioni sulle funzioni di accesso diretto ai file consultare l'help di Visual Studio.