Es. 1

Un dentista tiene traccia dei propri pazienti tramite un insieme di files. In particolare, in un file di testo denominato “pazienti.txt” tiene traccia, in ogni riga del file, dei seguenti dati:

- Codice fiscale, stringa di 16 caratteri terminati da uno spazio
- Cognome del paziente, al più 64 caratteri senza spazi, terminati da uno spazio
- Nome del paziente, al più 64 caratteri senza spazi, terminati da uno spazio
- Nome del file contenente le fatture relative a quel cliente, al più 64 caratteri senza spazi

Es. 1

Poi per ogni paziente, registra in un file di testo (il cui nome è salvato in pazienti.txt) i dati relativi alle fatture, nel seguente modo:

- Codice fiscale, ancora 16 caratteri terminati da spazio
- Costo dell’intervento, un numero (con virgola) rappresentante il costo in euro, terminato da uno spazio
- Data dell’intervento odontoiatrico, nella forma di aaaammgg

I problemi sorgono dal fatto che la segretaria del dentista, un po’ sbadata, ha registrato i dati in maniera poco ortodossa. Il dentista infatti si è accorto che ogni tanto il codice fiscale non è completo (ad esempio, mancano le ultime n lettere), e che a volte, nel file relativo ad un cliente sono registrate le fatture relative ad un altro cliente.

Es. 1

Il dentista ha rinunciato a correggere il problema del codice fiscale parzialmente specificato, e si aspetta che il computer sia in grado di capire che “CHSFRC75H06A445X” è da considerarsi equivalente a “CHSFRC75” (eventuali problemi di omonimia non gli interessano, i suoi pazienti non sono poi così tanti).
Invece il dentista vuole a tutti i costi correggere l’anomalia delle fatture relative ad un cliente ma registrate nel file relativo ad un altro cliente (altrimenti non sa come fare a calcolare gli importi totali e ad incassare i soldi…).
Si deve realizzare un programma che stampi a video, in ordine crescente per l’importo, tutte le fatture registrate nei file sbagliati, con a fianco di ognuna il nome del file sbagliato.
Nello **StartKit** si trovano alcuni file di testo di esempio: usare questi al fine di testare opportunamente le funzioni.

Es. 1

Esercizio 1 – Lettura dei pazienti (patient.h/patient.c)
Al fine di rappresentare correttamente i dati di un paziente, si dichiari in patient.h una struttura patient con i campi opportunamente dimensionati.

Si realizzi la funzione readPatient che, ricevuti in ingresso un puntatore ad un area di memoria sufficientemente grande (puntatore a patient), ed un puntatore a file, legga da tale file una struttura dati di tipo patient e la memorizzi nell’area di memoria passata come parametro. La funzione restituisca un valore intero interpretabile come vero se la lettura è andata a buon fine, o un valore “falso” in caso contrario.

```
int readPatient(patient * thePatient, FILE * f);
```

Si realizzi poi la funzione readAllPatient che, ricevuto in ingresso il nome di un file, lo apra, allochi memoria dinamicamente a sufficienza, legga e restituisca un array di strutture patient (con la sua dimensione) dei pazienti ivi contenuti (e chiuda il file…). Al fine di sapere quanti pazienti sono registrati nel file, si consiglia di fare una prima lettura di tutte le strutture presenti, e poi, dopo aver invocato opportunamente la funzione rewind(…), di procedere ad una seconda lettura memorizzando i dati.

```
patient* readAllPatient(char * fileName, int * dim);
```

Es. 1

Esercizio 2 – Lettura di fatture (patient.h/patient.c)
Dopo aver creato in patient.h una struttura dati di nome fattura, opportunamente definita per rappresentare i dati relativi ad una fattura, si realizzi la funzione readFattura che, ricevuto in ingresso un'area di memoria (puntatore a fattura) allocata opportunamente ed un puntatore a file, legga da tale file una fattura e la memorizzi nell'area di memoria. La funzione restituisca un valore "vero" se la lettura è andata a buon fine, "falso" altrimenti.

```
int readFattura(fattura * p, FILE * f);
```

Suggerimento: Si consiglia di prevedere nella struttura dati fattura un campo nomeFile dove andare a scrivere il nome del file da cui tale fattura è stata letta: questo faciliterà di molto la soluzione dei punti successivi.

Es. 1

Esercizio 3 – Confronto di codici fiscali (patient.h/patient.c)
Si realizzi una funzione compareCF che, dati due codici fiscali, restituisca un intero di valore diverso da 0 se questi sono identici secondo la notazione stravagante usata dalla segretaria, o 0 se sono diversi. Si ipotizzi che la segretaria abbia sempre inserito almeno un carattere del c.f. ...

```
int compareCF(char * cf1, char * cf2);
```

Es. 1

Esercizio 4 – Funzione di filtro e confronto di fatture (patient.h/patient.c)
Si realizzi una funzione compareFatture che, ricevuti in ingresso due fatture, restituisca un valore negativo, pari a 0 o positivo a seconda che la prima fattura preceda (per importo), eguagli o segua la seconda.

```
int compareFatture(fattura f1, fattura f2);
```

Si realizzi poi una funzione filter che, ricevuto in ingresso un array di strutture patient di pazienti, per ognuno di questi apra il file relativo, legga tutte le fatture ivi registrate erroneamente e le memorizzi in un unico, opportuno vettore allocato dinamicamente (di dimensione massima 1000). Si abbia cura di chiudere i files non più in utilizzo. La funzione restituisca un puntatore all'area di memoria allocata, ed un intero (passato se necessario per riferimento) rappresentante la dimensione logica del vettore.

```
fattura * filter(patient * list, int dim, int * logicDim);
```


Es. 1

Esercizio 5 – Ordinamento delle fatture (patient.h/patient.c)
Si scelga uno degli algoritmi di ordinamento visti a lezione durante il corso, e lo si implementi modificandolo al fine di ordinare il vettore di fatture sbagliate in ordine crescente per l'importo.

Esercizio 6 – Main (main.c)
Si scriva un programma main che, tramite la funzione readAllPatient legga l'elenco dei pazienti, e tramite la funzione filter legga tutte le fatture sbagliate. Si abbia cura di stampare a video le fatture (in ordine per l'importo), indicando per ogni fattura il file dove essa è contenuta…