Università di Pisa
Laboratorio di Programmazione di Sistema
AA 2004-2005

Implementazione di una rubrica telefonica in linguaggio Ansi/Iso C

Autore: Cristiano Anselmi
email: anselmi@cli.di.unipi.it
matricola: 226847

0 - Introduzione

Il progetto di Laboratorio di programmazione di sistema è un programma scritto in linguaggio Ansi/Iso C che realizza una rubrica telefonica.

I **file principali** sono:
librub.c, librubs.c, rubserver.c, rubclient.c.

1 - Convenzioni utilizzate nella relazione

Per **record** intendo una riga del file di testo che contiene una voce della rubrica
Per **contatto** una struttura di tipo voce.
Per **lista** intendo una lista (implementata con una struct) di elementi di tipo voce.

La relazione è impostata per essere consultata e non per essere necessariamente letta dall'inizio alla fine.
Ogni funzione è spiegata in maniera indipendente dalle altre.
E' possibile consultare una sezione apposita riguardo le **variabili globali** e i **files** utilizzati.

2 – LIBRUB.C

Il file librub.c contiene una collezione di funzioni (libreria) per operare sui record della rubrica in termini di: ricerca, formattazione, trasformazioni dati.

voce*rectovoce(charr[]);

Trasforma la stringa r[] in uno studente.

Algoritmo

- dichiaro e inizializzo studente con una chiamata alla funzione malloc gestendo

l'errore. Dichiaro un puntatore a caratteri **char*tmp** e lo inizializzo facendolo puntare a **r**.
- copio dentro il campo cognome di studente i primi 40 caratteri.
- aggiungo il carattere terminatore /0.
- incremento tmp di 40 caratteri.
- ripeto la stessa operazione per gli altri campi variando opportunamente l'incremento di tmp a seconda della grandezza del campo.
- restituisco studente

int vocetorec(charp[], voce*v);

La funzione trasforma lo **studentev** in una stringa formato **recordp**.

Algoritmo

- copio il campo cognome di studente in p con una strncpy.
- aggiungo gli altri campi con una strncat.
- ritorno il valore 0.

int formatvoce(charc[], voce*v);

Formatto lo **studentev** in una **stringac** per la stampa au schermo.

Algoritmo

- Copio la stringa "Cognome:" dentro c.
- elimino gli spazi da studente con la funzione **char* formatta_key(char* record).**
- aggiungo il campo formattato in c.
- aggiungo lo /n finale.
- ripeto la stessa operazione per gli altri campi.
- ritorno 0.

int matchvoce(charpattern[],voce*v);

Cercala stringa**pattern**dentrooginipossibilesottostringasequenzialedi **v**.
Restituisce1 in casodi successo,0 in casodi nessnriscontro,-1 in casodi errore.

Algoritmo

- dichiaradue stringhe
char campo_corrent1[41];
char campo_corrent2[21];

che conterrannoi campidellostudente.

- copiodentrocampo_corrente1il camponomedi studente.
- richiamolafunzione **int match(char*campo_corrente,charpattern[])** che verificala presenzadel patterndentrooginisottostringasequenzialedi campo_correntee restitusce un intero.
- Se l'interorestituitoè 1 terminaconsuccesso.
- Se l'interorestituitoè -1 terminaconinsuccesso.
- ripetolastessaoperazioneper gli altri campi.
- ritorna0 allafine. Segnochenonhatrovatonessunmatching.

NOTE

Hopreferitoutilizzaredue array:
char campo_corrent1[41];
char campo_corrent2[21];

anzichéun unicastringaallocatadinamicamenteper possibiliproblemiin run-time.

int match(char*campo_corrente,charpattern[]);

L'algoritmoconfrontaogni possibilesottostringasequenzialedel campo_correntedi lunghezza ugualealla lunghezzadi pattern. Restituisce1 se hatrovatoriscontro,0 se nonhatrovato riscontro,-1 in casodi errore.

ALGORITMO

- elimino gli spazi da campo corrente utilizzando **char* formatta_key(char* record).**
- controllo se la lunghezza del campo_corrente è minore del pattern. In questo caso termino restituendo 0.
- alloco la memoria corrispondente alla grandezza del pattern dentro la stringa **sub**.
- inizia un ciclo.
- copio dentro sub i primi caratteri del campo corrente.
- controllo l'uguaglianza con pattern. Se il riscontro è positivo termino.
- altrimenti mi sposto di un carattere nel campo corrente. Se il carattere successivo è /0 non ci sono piu' sottostringhe da processare e termino.
- Al termina libero la memoria allocata per sub e termino restituendo 0, segno che la funzione non ha trovato riscontro.

char* formatta_key(char* record);

Elimina gli spazi finali dalla stringa **record**
Funziona anche per i nomi composti.

ALGORITMO

- Dichiaro una stringa **corrente** allocata dinamicamente che conterrà un carattere.
  Dichiaro un contatore i =0;
- Copia dentro corrente l'i-esimo carattere del record.
- Se questo carattere è uno spazio, ripete la stessa operazione per il carattere successivo.
  Se il carattere successivo è uno spazio allora copia i primi i caratteri, escludendo i due spazi, dentro una stringa r allocata dinamicamente e la restituisce.

2 – LIBRUBS.C

Una collezione di funzioni che realizzano operazioni sulla rubrica in termini d'inserzione, cancellazione, ricerca, salvataggio su file, caricamento in memoria e stampa su schermo.
Viene dichiarata all'inizio una variabile globale conta. Si veda la sezione corrispondente per un approfondimento.

int push(elem**db, voce*v)

Inserisce lo studente **v** dentro la lista **db** seguendo il modello LIFO.

ALGORITMO

- viene allocata la memoria per inizializzare **t**, un elemento della lista. In caso di errore termina.
- assegno al campo voce di t lo studente v e al campo successivo il resto della lista db.
- assegna alla testa della lista t e restituisce un valore che simboleggia la riuscita dell'operazione.

int cancella(char key[], elem**db)

Cancella gli studenti con pattern **key (**che contengono la key in una loro sottostringa sequenziale) dalla lista **db.**
Utilizza la funzione **int matchvoce(charb[],voce*v)** per il matching.

ALGORITMO

L'algoritmo è molto simile alla procedura per la cancellazione di un elemento da una lista.
Vengono dichiarati due **contatori**:
int cont=0 – che conta il numero degli elementi trovati.
int conta_processati= 0 – che conta il numero degli elementi processati ma non cancellati.
Viene utilizzata la variabile globale **int conta** che decrementa quando viene cancellato un elemento.
Viene dichiarato un **elem* prec** ed inizializzato con una **malloc**. Questi conterrà l'elemento precedente nella lista rispetto all'elemento corrente.
Viene dichiarato un puntatore: **elem* t= **db** che scorre la lista.
A questo punto parte un **ciclo** infinito dove:

- una struttura if controlla se conta = 0 e se il numero degli elementi processati è diverso da conta. In tal caso lo studente corrente e il pattern vengono passati alla funzione **int matchvoce (charb[],voce*v)** per il riscontro.

Se il riscontro è positivo allora si esegue la cancellazione:
- il campo next dell'elemento precedente punta al campo next dell'elemento successivo all'elemento corrente.
- l'elemento a cui punta t viene eliminato.
- viene decrementato conta e incrementato cont.
- t punta all'elemento successivo.

Se il riscontro è negativo:
- t e prec puntano al proprio elemento successivo.
- viene incrementato conta_processati.

- Se conta = 0 non ci sono piu' elementi nella lista e il ciclo termina. Se conta_processati è uguale a conta vuol dire che tutti gli elementi sono stati processati. In tal caso termina restituendo cont.

int cerca(char key[], elem *db, elem **r)

Ricerca gli elementi che contengono la chiave **key** all'interno della lista **db** e restituisce il numero degli elementi trovati. La lista **r** conterrà gli elementi trovati.

ALGORITMO

- parte un ciclo infinito dove:
- viene richiamata la funzione **int matchvoce(char b[],voce *v)** che controlla se la chiave **b** è presente all'interno dello studente corrente **v**.
- Se il riscontro è positivo:
  - viene allocata memoria per inizializzare un nuovo elemento della lista.
  - poi mi comporto come nella funzione **push, inserzione LIFO nela lista r**:
    - assegno lo studente al campo voce dell'elemento allocato
    - l'elemento successivo all'elemento allocato è il resto della lista r.
    - la lista **r** punta all'elemento allocato
  - viene incrementato il numero degli elementi trovati.
- Sia in caso positivo che in caso negativo si passa all'elemento successivo della **lista db**.
- Il ciclo termina se il numero degli elementi processati è uguale a conta, variabile globale che conta il numero degli elementi nella lista..

int stampaDatabase(FILE*uscita, elem*db)

Stampa il contenuto della lista **db** nel **FILE uscita**.

ALGORITMO

E' molto semplice. I passi:

- Scorre la lista.
- per ogni elemento richiama la funzione **int formatvoce(charc[], voce*v)** che trasforma lo studente v in una stringa formattata.
- Richiama la funzione fprintf(FILE*uscita,char* formato,char [] r) che stampa la stringa r nel FILE uscita.
- passa all'elemento successivo.

int leggiArchivio(FILE*ingresso, elem**db)

Legge dal file d'ingresso una serie di record e inserisce il tutto in una lista di elementi i cui campi sono: studente e puntatore all'elemento successivo.

ALGORITMO

Dichiara la stringa char r[143].
Riazzera la variabile **int conta** che conta il numero degli elementi nella lista
In un ciclo while:

- richiama la funzione **s = fgets(r,143,ingresso)** che preleva 143 caratteri dal file ingresso, li copia dentro la stringa r e restituisce un puntatore a caratteri s.
- richiama la funzione **voce*rectovoce(charr[])**, che trasforma la stringa r in uno studente
- esegue la funzione **int push(elem**db, voce*v)** che inserisce un nuovo studente nella lista.
- incrementa conta.
- restituisce conta.

int registraArchivio(FILE*uscita, elem*db)

Salva i conttatti della rubrica cotenuti nella lista **db** nel file **uscita**

ALGORITMO

Ho utilizzato un algoritmo che è risultato essere una conseguenza dell'utilizzo del metodo a pila LIFO per l'inserzione.
Quando viene caricata la lista degli elementi l'ordine, rispetto al file, risulta invertito: l'ultimo record è il primo elemento della lista e così via.
Quando vengono registrati gli elementi sul file si utilizza il metodo FIFO a coda, dunque s'inverte l'ordine.
Per rispettare l'ordine dei contatti rispetto al file di origine:

- utilizzo un file temporaneo FILE* tmp
- scrivo su questo file i record della rubrica.
- chiudo il file temporaneo
- riapro il file temporaneo
- ricarico i record in una lista. In questo modo ho ripristinato l'ordine originale degli elementi rispetto al file. Utilizzo una nuova lista. Algoritmo praticamente uguale alla funzione **int leggiArchivio(FILE*ingresso,elem**db).**
- riscrivo tutti gli elementi nel file vero e proprio aggiornando opportunamente la variabile globale conta.

3 – RUBSERVER.C

Risponde alle richieste dell'utente (attraverso rubclient.c) gestendo la rubrica telefonica in termini di cancellazione, inserzione e ricerca contatti.
Contiene un main.

Contiene tre funzioni principali:

- **int insert(char buf[])**
- **int delete(char buf[])**
- **int cerca_rec(char buf[])**

e tre funzioni di utilità:

- **void rispondi(char*, int, char*)**
- **void gestore_INT(int)**

Variabili Globali

Si utilizzano come variabili globali le seguenti perchè richiamate da piu' funzioni in rubserver.c: .

- **elem* db :** è la lista. Utilizzata da tutte le funzioni principali.
- **char pipe_server[124]** la pipe del server. Utilizzata nel **main** e in **void gestore_INT(int)**
- **char* file**: è la stringa che contiene il PATH del file dei record. Utilizzata da tutte le funzioni principali e in **void gestore_INT(int).**
- **char log_[1500]** è una stringa che tiene conto delle ultime operazioni fatte. Utilizzata nel **main** e in **void gestore_INT(int).**

int main(int argc, char* argv[])

- assegna alla variabile globale **char* file** l'argomento della line adi comendo corrispondente al nome del file di testo.
- controlla se il numero di argomenti è diverso da due. In questo caso termina
- controlla se il formato del file dei record è idoneo alle specifiche:
  - prova ad aprire il file con **fopen(file,"r")**. Se ci sono errori termina.
  - richiama la funzione **int leggiArchivio(FILE*ingresso,elem**db)** che restituisce -1 in caso di errore. Se ci sono errori termina.
- crea la pipe con nome.
- gestisce il segnale di terminazione con una procedura standard.
- parte un ciclo infinito dove
  - apre la pipe in lettura e prova a leggere (chiamata bloccante) 161 caratteri.
  - separo gli argomenti del buffer di 161 caratteri:
    - estraggo i primi 21 caratteri e li copio dentro la stringa **pipe_client** che conterrà il nome della pipe del client eliminando gli spazi finali.
    - copio i restanti caratteri dentro l'array **record**

  - a lettura avvenuta, controlla il primo carattere per stabilire il tipo di richiesta e richiama la funzione corrispondente.
    Nel caso la funzione sia **int delete(char*, char*)** o **int cerca_rec(char*, char*)** vengono eliminati i paddings attraverso la funzione **char* formatta_key(char*)**.

La stringa **char log[1500]** registra di volta in volta l'operazione scelta.

int insert(char* pipe_client, char *rec)

Inserisce il testa alla lista un nuovo contatto.
Il parametro **char* pipe client** è la stringa che contiene il nome della pipe del client.
Il parametro **char* rec** è il contatto da aggiungere in formato record.

NOTA

Nella funzione utilizzo questa istruzione **v = rectovoce(record)** che trasforma il record, visto come un array, in un contatto.
L'array è stato dichiarato ed inizializzato nel main.
Visto che in C gli array vengono passati per riferimento e non per valore, ossia viene passato il puntatore al primo elemento, ho dovuto risolvere il problema di passare alla funzione **voce* rectovoce(record)** un array anziché un puntatore.
Ci sono due soluzioni:

- quella che ho scelto: dichiarare all'interno di int insert(char* , char *) un array e copiarci i riferimenti del puntatore
- utilizzare la **voce* rectovoce(record)** all'interno del main e trasformare la funzione **int insert (char* , char*)** in **int insert(char* , voce*)**. In questo caso avrei dovuto importare il file **librub.h**, contenente la definizione della struttura voce, all'interno del file **rubserver.h**

ALGORITMO

- utilizza la funzione **voce* rectovoce(record)** per trasformare il record in un contatto
- utilizza la funzione **int push(elem**db, voce*v)** per aggiungere in testa alla lista il contatto
- incrementa il valore della variabile globale **conta**, definita in **librubs.c**, che aggiorna il numero dei contatti in rubrica
- controlla l'esito dell'operazione e risponde al client.

int delete(char* pipe_client, char* r)

Cancella un contatto della rubrica contenente la stringa r.
Il parametro **char* pipe client** è la stringa che contiene il nome della pipe del client.
Il parametro **char* r** è il pattern da ricercare.

ALGORITMO

- Controlla se la lista non sia vuota. In questo caso richiama la funzione **int cancella(charkey[], elem**db)** restituendo il numero di contatti cancellati.
- Sposta la lista di n posizioni avanti se sono state cancellate le prime n posizioni.
- risponde al client a seconda dei casi.

int cerca_rec(char* pipe_client, char* r)

Ricerca all'interno della lista i contatti con pattern r.
Il parametro **char* pipe client** è la stringa che contiene il nome della pipe del client.
Il parametro **char* r** è il pattern da ricercare.

Tra le variabili locali dichiaro:
- **elem* er**, lista che conterrà gli elementi trovati.
- **char messaggio[141]**, che conterrà il record da inviare al client preso dalla lista er.
- **char messaggio_fin[141]** messaggio finale per chiudere la comunicazione col client.

ALGORITMO

- cerca gli elementi utilizzando la funzione **int cerca(charkey[], elem*db, elem**r)**
- risponde al client in presenza di errore o di ricerca nulla
- Altrimenti:
    - apre la pipe del client in scrittura
    - avvio un ciclo che terminerà quando l'elemento successivo di er è nullo, nel quale:
        - trasformo il contatto trovato in una stringa con int vocetorec(charr[], voce* v).
        - aggiungo /0 alla fine
        - aggiungo "A" in testa, attraverso il puntatore a caratteri pi, che punta a messaggio
        - scrivo la pipe
        - mi sposto di una posizione nella lista

- invio il messaggio di chiusura aggiungendoi paddings.

void rispondi(char* pipe, int status, char* type)

Scrive nella **pipe**, a seconda dello **status**, una stringa **type** e spedisce tutto al client.
Dichiaro la variabile char messaggio[141] che conterrà il messaggio da spedire al client.
A seconda del valore di status genera il messaggio e aggiunge i paddings.
A questo punto scrive tutto al client.
Funzione di utilità utilizzata da tutte le funzioni principali.

void gestore_INT(int)

Funzione richiamata se viene recepito un segnale di tipo SIGTERM.

ALGORITMO

- Stampa il contenuto della stringa log, che tiene conto delle operazioni fatte.
- apre il file dei record
- richiama la funzione **int registraArchivio(FILE*uscita, elem*db)** per salvare i contatti nel file
- elimina la propria pipe.
- esce con codice -1

4 – RUBCLIENT.C

E' un comando per richiedere al server della rubrica le operazioni di:
- inserzione di un nuovo contatto.
- cancellazione di un contatto.
- ricerca di uno o piu' contatti.

Vengono dichiarate la seguente variabile globale:
- char richiesta[161]: la richiesta da inviare al Server. Utilizzata dalla funzione **void insert(char[])** e **void delete(char[], char[]).**

int main(int argc, char* argv[])

Nel main:

- creo la pipe della forma **"./tmp/pid_client"** fino ad un massimo di 20 caratteri. **Ubuntu** raggiunge i 5 e forse piu' numeri per il pid del processo nella tabella dei processi. Quindi la stringa può raggiungere un massimo di 11 o 12 caratteri. Ho lasciato come grandezza venti per una questione di sicurezza.
- Viene interpretato il comando. Prima però controlla se il o numero degli argomenti sia maggiore o uguale a 2 o a 3 a seconda dei casi.
- A questo punto il controllo passa alla funzione prescelta.

void insert()

Chiede al Server di inserire un nuovo contatto nella rubrica.
Variabili principali:
- **char buf[141]**: conterrà il record da inviare al Server.

ALGORITMO

- formatto il buffer principale (buf) con RECLEN spazi e aggiungo alla fine il carattere terminatore.
  Per ogni campo:
  - lancio la funzione insert_campo(char* campo) che restituisce il valore del campo inserito dall'utente della forma: campo + \n + \0
  - mi sposto di 40 caratteri
  - copio il contenuto del valore del campo in buf.

- alla fine preparo il messaggio da inviare al Server secondo i requisiti richiesti
- scrivo al Server e attendo la risposta
- chiudo il descrittore della pipe e la elimino e il programma termina.

Per quanto riguardo il formato di buf:
-l'insert chiede d'inserire prima il nome e dopo il cognome. Mentre, nel file dei record viene salvato prima il cognome e poi il nome attraverso la rectovoce.
Per fare in modo che questo accada

- copio il primo campo (Nome) nello spazio compreso tra buf[39] e buf[79].
- In seguito faccio puntare p (il puntatore a buf) all'inizio dell'array e copio il secondo campo nelle prime 40 posizioni.
- Dopo mi sposto di 80 posizioni.

char* insert_campo(char* campo)

Chiede all'utente il valore del campo corrente e lo restituisce.
La variabile principale è **char tmp[40]** che memorizza il valore del campo.
Un altra variabile importante è **char formato_stampa[13]** che stampa sullo STDOUT la stringa campo+ " -> ".

ALGORITMO

- formatto il buffer temporaneo con 40 spazi e formato_stampa con 12 caratteri.
- leggo dallo standard input il valore del campo e lo inserisce dentro t.
- stampo ciò che ho inserito
- controllo la presenza di caratteri invalidi.

void delete(char key[]);

Richiede la cancellazione di un record dalla rubrica

ALGORITMO

- controlla se la chiave è minore di 40 caratteri
- scrive al server e aspetta la risposta che genera una stampa a seconda dei casi.

void search(char key[])

Richiede la cancellazione di un record dalla rubrica
Due buffer:

- char richiesta[RECLEN+100]: il buffer dove viene copiato il risultato generato dal server
- char messaggio[141]: buffer temporaneo dove viene copiato il buffer richiesta.

ALGORITMO

- controlla se la chiave è minore di 40 caratteri
- scrive al server e aspetta la risposta.
- inizia un ciclo infinito
- legge dalla pipe
- se il messaggio è ENOCo EERR termina chiudendo descrittore ed eliminando la pipe.
- altrimenti dichiara la variabile locale **char messaggio[141]** e li copia dentro il buffer della read, viene trasformato in un contatto, formattato e stampato.

void scrivi_pipe_server(char key[], char command[])

Scrive nella pipe del server la stringa:
command + pipe_client + paddings + key + paddings

E' utilizzata dalle funzioni **void delete(char[], char[])** e **void search(char[], char[])**.

int controllo_no_numeri(char*);

Controlla se nella stringa passata come parametro sono presenti dei numeri
e risponde con un messaggio intero opportuno.
Viene utilizzata dalla **void insert(char[])**.

int controllo_no_lettere(char*);

Controlla se nella stringa passata come parametro sono presenti dei numeri
e risponde con un messaggio intero opportuno.

Viene utilizzata dalla **void insert(char[])**.

STRUTTURE DATI PRINCIPALI

struttura dati principale: la struttura dati principale è

```
typedef struct{
 char nome[LNOME+1];
 char cognome[LCOGN+1];
 char citta[LCITT+1];
 char telefono[LTELE+1];
 char commento[LCOMM+1];
} voce;
```

che contiene un record della rubrica telefonica e

```
typedef struct elem{
  voce* ptvoce;
  struct elem* next;
} elem;
```

che implementa una lista di elementi di tipo voce.

VARIABILI GLOBALI

La principale variabile globale è:
int conta;
risiede in **librubs.c** e conta il numero di record caricati durante la sessione. In particolare:

- viene utilizzata dalla funzione **int leggiArchivio(FILE*ingresso,elem**db)** in **librubs.c** che la inizializza al valore zero, e incrementa il suo valore di uno per ogni record letto. Al termina della funzione conta conterrà il numero di record della rubrica.
- viene utilizzata dalla funzione **int registraArchivio(FILE*uscita,elem*db)** in **librubs.c** in **while(i!=conta)** come guardia in un ciclo che registra i record nel file . prorub.dat: ad ogni ciclo viene registrato un record nel file e incrementato il valore di **i** inizializzato a 0 . Raggiunto un valore uguale a conta l'espressione booleana della guardia cambia il proprio valore in **false** e il ciclo while termina registrando tutti

i record.

- viene utilizzata dalla funzione **int cerca(char key[], elem *db, elem **r)** in **librubs.c** in un ciclo **do-while(i!=conta)**: ad ogni ciclo viene processato un record della rubrica **db** per verificarne l'uguaglianza con il pattern **key** e incrementato il contatore i inizializzato a 0 all'inizio della funzione Raggiunto un valore uguale a conta l'espressione booleana della guardia cambia il proprio valore in **false** e il ciclo while termina dopo aver processato tutti i record.
- viene utilizzata dalla funzione **int cancella(char key[], elem **db)** in **librubs.c** all'interno del ciclo **do-while(1=1)**. Ad ogni ciclo viene processato un record che puo' essere cancellato o ignorato, se viene cancellato viene decrementato il valore di conta, se viene ignorato viene incrementato il valore della variabile **conta_processati** inizializzata a 0. La struttura di controllo **if((conta!= 0)&&(conta!= conta_processati))** verifica due guardie: se conta è uguale a 0 (tutti gli elementi sono stati cancellati), o se conta è uguale a conta_processati (tutti gli elementi sono stati processati e uno, qualcuno o tutti cancellati) il ciclo termina restituendo il numero di elementi cancellati.
- viene utilizzata nella funzione **int insert(char buf[], char* file)** in **rubserver.c** che incrementa il suo valore in seguito all'inserimento di un nuovo elemento nella rubrica.
- viene utilizzata nella funzione **int delete(char buf[], char* file)** in **rubserver.c** come guardia nel ciclo

```
if (conta!= 0){
if(db->ptvoce->nome!=NULL){
status= cancella(r,&db);
 }
}
```

che verifica se il contatore di record non sia nullo, in tal caso richiama la funzione cancella per eliminare un record. e subito dopo nel ciclo

```
if (conta!= 0){
while(bool== 0){
if(db->ptvoce->nome==NULL)
db=db->next;
else
bool= 1 ;
}
                                                    }
```

che verifica se il contatore di record non sia nullo, in tal caso sposta il db di n posizioni avanti se sono stati cancellati i primi n records.

FILES

Per memorizzare i dati viene utilizzato un file di testo: /home/nome_utente/.prorub.dat
che:

- viene richiamato in lettura dalla funzione **int leggiArchivio(FILE*ingresso,elem**db)** nel file **librubs.c** che carica i record nella lista di voci **elem**
- viene richiamato in scrittura dalla funzione **int registraArchivio(FILE*uscita,elem*db)** nel file **librubs.c** che memorizza la rubrica caricata precedentemente nella lista di voci **elem** e modificata (aggiunta di nuovi record e cancellazioni) nel file.