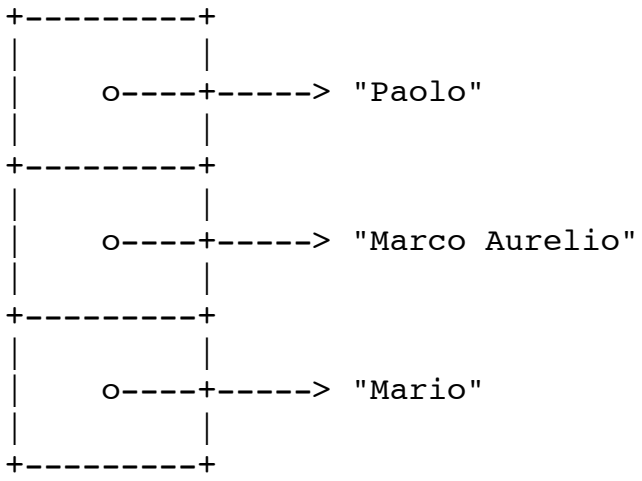
1. lettura da file e gestione liste

Implementare le due funzioni con prototipo

```
char** leggi_lista(FILE* file_in, int* nof_elements);
void stampa_lista(char** mio_ar, int n_elems);
```

La funzione *leggi_lista* ha due argomenti: un puntatore a file, e un puntatore a intero. Il puntatore a file deve essere utilizzato per leggere l'elenco di nomi contenuto nel file *lista_nomi.txt*. Il secondo argomento sarà assegnato dalla funzione *leggi_lista* con il numero di nomi contenuti nel file *lista_nomi.txt* . La funzione *leggi_lista* effettua le seguenti operazioni:

- crea un array di puntatori a stringhe opportunamente dimensionato in base al numero di nomi presenti in *lista_nomi*;
- assegna al secondo parametro della funzione stessa il numero di elementi contenuti nell'array;
- legge i nome presenti nella *lista_nomi* e li memorizza ciascuno in una stringa, riferita da un elemento dell'array;



- restituisce l'array di puntatori precedentemente allocato e assegnato.

La funzione *stampa_lista* stampa l'array restituito da *leggi_lista*; ha due argomenti: l'array di puntatori che si vuole stampare e l'intero *n_elems*, che contiene il numero di nomi contenuti nel file.

Il *main* deve contenere la chiamata a entrambe le funzioni.

```
//
// ------------ IMPLEMENTAZIONE -------------
//
int main(int argc, char** argv) {
     // CHIAMATA alla funzione leggi_lista()

     // CHIAMATA alla funzione stampa_lista()

  return 0;
}
//
// PRE: il numero di righe del file corrisponde
//      esattamente al numero degli elementi
//
char** leggi_lista(FILE* file_in, int* nof_elements) {

}
//
void stampa_lista(char** mio_ar, int n_elems) {

}
```

2. Gestione di input da tastiera (funzioni *gets* e *fgets*)

Scrivere un programma per la lettura da standard input di coppie *<nome,cognome>*. Nome e cognome possono essere separati da uno o più spazi bianchi e/o tabulazioni. La stringa letta in input si assume sempre composta di una coppia *<nome,cognome>*.

Si implementi la funzione *parse_nome* per l'estrazione del *nome*. La funzione ha il seguente prototipo

```
char* parse_nome(char* stringa_completa);
```

La funzione *parse_nome* prende in input la stringa contenente nome e cognome, e restituisce la stringa contenente solo il nome immesso dall'utente.

Il *main* deve contenere la chiamata alla funzione implementata e un meccanismo elementare di gestione dell'interazione che richieda all'utente di inserire una coppia composta di nome e cognome, e che termini nel momento in cui l'utente preme invio senza avere digitato altri caratteri.

3. ricorsione e manipolazione di stringhe (libreria <*ctype.h*>)

Leggere da tastiera una stringa composta di più parole, per esempio 'Paolo Rossi'. Implementare una funzione ricorsiva per la stampa a video della stringa ricevuta in input convertita in caratteri maiuscoli. per esempio, data la precedente, vorremmo ottenere 'PAOLO ROSSI'.

Il prototipo della funzione da implementare è

```
void recur_to_up(char* in_str)
```

e la funzione deve convertire i soli caratteri alfabetici minuscoli, lasciando inalterati gli altri, le cifre e gli spazi.

4. manipolazione di stringhe

Scrivere un programma che legga in input da tastiera una frase composta da almeno 5 parole e richiami una funzione per la stampa di tale stringa in caratteri maiuscoli e al contrario. Per esempio, data la frase

buongiorno a tutti da daniele

il programma deve stampare

ELEINAD AD ITTUT A ONROIGNOUB

La funzione per la stampa deve essere ricorsiva. Per lo svolgimento dell'esercizio è possibile utilizzare le funzioni di libreria *fgets()* e *toupper()*.

5. cifrario di Cesare: il *decoder*

Scrivere un programma che prenda in input la stringa riportata qui sotto e che la decodifichi utilizzando il cifrario di Cesare, ossia decrementando di 2 unità il valore intero associato a ciascun carattere. Per esempio, se in input avessimo la stringa "ekcq", il programma dovrebbe essere in grado di risalire alla stringa originale "ciao".

la stringa da decifrare è la seguente (copiarla e inserirla direttamente nel programma):

char *encoded = "Kn\"eqtuq\"hqtpkueg\"wp)kpvtqfw|kqpg\"cn\"nkpiwciikq\"fk
\"rtqitcooc|kqpg\"E\"g\"cnnc\"rtqitcooc|kqpg\"korgtcvkxc0\"Wp\"pwogtq
\"eqpukuvgpvg\"fk\"qtg\"gb\"wvknk||cvq\"rgt\"uxqnigtg\"gugtekvc|kqpk\"kp

\"ncdqtcvqtkq\"hkpcnk||cvg\"c\"rtgpfgtg\"eqphkfgp|c\"eqp\"kn\"nkpiwciikq
\"vtcokvg\"nc\"tgcnk||c|kqpg\"fk\"rtqitcook\"kp\"nkpiwciikq\"E0";

6. ricerca di sottostringhe

Scrivere un programma per la lettura e manipolazione del file *complexity.txt*.

Il programma dovrà effettuare le seguenti operazioni:

- visualizzare il contenuto del file, senza modifiche;
- visualizzare solo le righe che contengono la stringa "computational", e contarne il numero totale di occorrenze, tenendo conto del fatto che la stringa occorre al massimo una volta per riga.

Per la ricerca di una stringa all'interno di un'altra è possibile utilizzare la funzione *strstr()*, il cui prototipo è contenuto nella libreria <*string.h*>, definita come segue:

Declaration:

```
char *strstr(const char *str1, const char *str2);
```

Finds the first occurrence of the entire string *str2* (not including the terminating *null* character) which appears in the string *str1*.

Returns a pointer to the first occurrence of *str2* in *str1*. If no match was found, then a null pointer is returned. If *str2* points to a string of zero length, then the argument *str1* is returned.

7. l'environment dei processi

Aiutandosi con il manuale (*man environ*) scrivere un'istruzione che stampi l'environment della propria shell, selezionando le righe che contengono informazioni sull'utente (LOGNAME), sulla sua home (HOME), e sulla variabile d'ambiente PATH.

8. re-implementazione di printenv

Utilizzando la variabile globale *environ* scrivere un programma che scorra *environ* stampandone il contenuto, e producendo lo stesso output del comando *printenv*.