SETI 25/26 - Programmazione Sistemi e Reti - Laboratorio 2

Esercizio 1 - Programmare echo

1. Scrivere un programma `myecho` chi stampa sul terminale il testo che riceve in argomento. Questo programma riceve un numero arbitro di string separate da spazi e le deve stampare su `stdout`. La funzione `main` da inserire nel file `myecho.c` avrà quindi la seguente forma: `int main(int argc, char* argv[])` dove il primo argomento corrisponde alla lunghezza della array di string `argv` che corrisponde agli argomenti dati al programma quando chiamata dal shell (NB: la prima string è il nome stesso del programma).

2. Verificare che il vostro programma si comporta come il commando `echo`. Cosa succede se eseguite `./myecho *` e `./myecho abc | wc -c`?

3. Scrivere una funzione `int streq(char *s1,char *s2)` che ritorna `1` se le due string `s1` e `s2` sono uguale e `0` altrimenti (è chiesto di riprogrammare questa funzione e di non usare la funzione `strcmp` vista a lezione).

4. Modificare il vostro programma in modo che accetta le opzione `-s` e `-n`. Se queste opzione sono presente, devono essere inserite prima degli altri argomenti.

   - `-s` indica che non vogliamo separare le string stampata con degli spazi
   - `-n` indica che non vogliamo andare a capo dopo avere stampato le string.

   Testare il vostro programma con le due commando: `./myecho -l -n s i n g > test.txt` e `./myecho le line >> test.txt`. Si tutto funziona dovete avere nel file `test.txt` il testo `single line`.

Esercizio 2 - Cifratura di Cesare

Scrivere il programma seguente `rot13.c` e testarlo (questo programma aspetta un testo su stdin).

```c
#include <stdio.h>
#include <string.h>

int main() {
  unsigned int i=0;
  char s[64];
  fgets(s, 64, stdin);
  for(i = 0; i < strlen(s); i++) {
    if (97 <= s[i] && s[i] <= 122)
      s[i] = (((s[i] - 97) + 13) % 26) + 97;
  }
  fputs(s, stdout);
  return 0;
}
```

1. Cosa ha le chiavi ma non riesce ad aprire le serrature ? Ecco la risposta:

   hab fcnegvgb

2. Trovare nella pagina man `ascii` della sezione 7, i codici ASCII in decimale per le lettere `'a'` e `'z'`.

3. Capire cosa fa il programma esattamente.

4. Cambiare il testo per fermare il ciclo `for` perché non usa la funzione `strlen`.

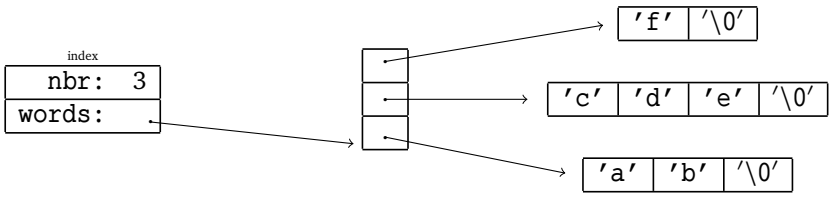
5. Migliorare il programma perché possa anche essere usato con maiuscole.

6. Cosa succede se entriamo un testo di più di 64 caratteri? Migliorare il programma perché funziona con dei testi di lunghezza arbitraria (**Suggerimento:** Guardare il valore di ritorno di `fgets`.)

7. Produrre una versione cifrata della "GNU General Public License" (`http://www.gnu.org/licenses/gpl.txt`).

Esercizio 3 - Estrarre le parole

In questo esercizio, consideriamo delle sequenze di parole (composte da caratteri alfanumerici) e separate da spazi (ci possono essere più spazi fra due parole e ci possono essere spazi prima della prima parola e dopo l'ultima parola e una sequenza può contenere solo spazi). Vogliamo scrivere una funzione che estrae le parole di una tale sequenza. Per questo, useremo il seguente tipo:




```
typedef struct {
  unsigned nbr;
  char **words;
} w_index;
```

Il tipo `w_index` contiene un array di string (allocate) di dimensione `nbr`. Chiamerò *indice* ogni valore di tipo `w_index`. La figura sopra è un esempio di indice che contiene tre parole `ab`, `cde` e `f`.

1. Scrivere una funzione `void print_index(w_index *pi)` che stampa il contenuto dell'indice di indirizzo `pi`.
2. Scrivere una funzione `unsigned size_words(w_index *pi)` che ritorna il numero di caratteri nelle parole contenute nell'indice di indirizzo `pi`.
3. Scrivere una funzione `char *concat_words(w_index *pi)` che ritorna una string (allocata) che contiene le parole concatenate dell'indice di indirizzo `pi`.
4. Scrivere una funzione `w_index *cons_index(const char *s)` che ritorna un indice che contiene le parole della string `s` (si suppone che `s` è una sequenza di parole come descritto all'inizio).