Esercitazione 2

Stringhe e libreria standard di IO, creazione di librerie statiche, uso di `valgrind`.

Esercizio 1: tokenizer

Scrivere un programma che data una stringa come argomento, stampa a video (`stdout`) le parole contenute nella stringa in input una per ogni linea. Usare la funzione di libreria 'strtok' per estrarre i token dalla stringa. Per realizzare il programma implementare la funzione 'tokenizer' con la seguente segnatura:

```
void tokenizer(char *stringa, FILE *out);
```

Testare il programma con i seguenti casi (supponiamo che l'eseguibile si chiami es1):

```
./es1 "ciao mondo"
./es1
./es1 "questa e' una stringa di alcune parole!"
./es1 questa è un\'altra stringa
```

Esercizio 2: tokenizer_r

Scrivere una seconda versione del programma precedente che implementa la funzione 'tokenizer_r' che ha la stessa identica interfaccia, ma che utilizza al suo interno `strtok_r` invece di `strtok`. Quali sono le principali differenze tra 'tokenizer' e 'tokenizer_r' ? (Suggerimento: leggere con attenzione il `man` di `strtok_r`).

NOTA: se si utilizza l'opzione `-std=c99`, per evitare i warnings del tipo `implicit declaration of function 'strtok_r'`... aggiungere la seguente opzione di compilazione `-D_POSIX_C_SOURCE=200112L`. La funzione strtok_r fa parte dello standard POSIX-2001.

Esercizio 3: tokenizer_rfile

Scrivere un programma C che prende in input 3 argomenti, i primi due argomenti sono obbligatori, il terzo e' opzionale. Il primo argomento è il nome di un file di input in formato testuale contenente stringhe di parole. Ogni stringa ha una lunghezza massima pari a 1024 caratteri. Il secondo argomento corrisponde ad un file testuale di output che dovra' contenere tutte le parole del primo file, scritte al contrario una per ogni riga. Se il file di input non esiste, il programma dovrà ritornare un messaggio di errore opportuno sullo standard error (stderr). Se il file di output esiste questo dovra' essere sovrascritto se il terzo argomento del programma non è stato specificato, oppure se tale argomento è diverso dal carattere 'a'. Se il terzo argomento è uguale al carattere 'a' l'output del programma dovrà essere appeso in fondo al file.

Esercizio 4: valgrind

Verificare la correttezza degli accessi in memoria utilizzando `valgrind` del programma realizzato nell'esercizio 3. Verificare che non ci siano memory leaks all'uscita del programma.

Valgrind permette, fra l'altro, di capire se le variabili sono inizializzate prima del loro uso, se accediamo a memoria gia' deallocata o mai allocata o a zone non inizializzate.

Passi:

- compilare il file da verificare con opzione `-g` per includere le informazioni di debugging (anche se non è strettamente necessario).
- eseguire il programma con `valgrind` nel modo seguente :

```
bash$ valgrind ./prova
```

in questo modo, a schermo verranno riportare le infrazioni rilevate. Ad esempio, *invalid read* o *invalid write* sono accessi in lettura o scrittura a memoria non allocata o gia' deallocata.

Se si specifica l'opzione `–leak-check=full` (attenzione al doppio trattino), valgrind fornirà dettagli per ogni blocco di memoria che non è più raggiungibile o che pur essendo raggiungibile non è stato liberato, dando anche l'informazione di dove il blocco è stato allocato.