Introduzione

Il progetto si articola in 3 file sorgenti, tra cui swordx.c, ovvero il programma principale, un makefile e 2 moduli contenenti le strutture dati utilizzate.
I file sorgenti sono contenuti nella cartella src e i moduli delle strutture dati sono contenuti nella cartella lib.

I file list.c e parLog.c sono due simili implementazioni della lista concatenata, ovvero la struttura dati dinamica formata da una sequenza di nodi, in cui ciascuno di essi, a partire dal primo, contiene campi arbitrari ed un riferimento al nodo successivo.

Il file list.c contiene l'implementazione dei nodi della lista, la struttura dati in cui ciascun nodo comprende tutte le parole lette dai file con relativa occorrenza.

Il file parLog.c implementa una lista concatenata come struttura usata per costruire la lista dei file letti e delle parole lette e ignorate, con la relativa tempistica di esecuzione per ciascun file.

Tutti i moduli sopracitati hanno un corrispettivo header file che funge da interfaccia verso il programma principale e contenente le definizioni delle strutture dati e le dichiarazioni di funzioni accessibili dall'esterno.

Il programma principale è swordx.c, non dotato di un header file proprio e le cui dichiarazioni di funzioni sono localizzate nel file stesso prima delle loro definizioni. Il programma fa un conteggio delle parole contenute nei file passati in input, scorrendo per ogni parola la lista per vedere se è uguale a una di quelle presenti ed incrementarne l'occorrenza, o se aggiungerla alla fine. Inoltre il programma offre una serie di funzioni aggiuntive osservabili tramite il comando -help, --help o -h. Il programma riconosce come parole le stringhe composte da al massimo 100 caratteri dell'alfabeto inglese più le 10 cifre arabe, codificate secondo il codice ASCII, separate da uno o più spazi bianchi o invii. L’applicazione non è sensibile alle maiuscole, quindi una stessa parola con caratteri maiuscoli verrà contata come uguale all’altra. Non vengono riconosciute le stringhe con lettere accentate o caratteri non alfanumerici, infatti verrano riconosciute come una stringa vuota (se si fanno più stringhe composte da tali caratteri uguali o diverse verranno considerate uguali, incrementando la stringa vuota di un'occorrenza). Se si pongono caratteri non alfanumerici all'inizio di stringhe alfanumeriche, queste verranno considerate vuote, se invece questi caratteri si trovano in mezzo alla stringa, la stringa viene troncata. Chiaramente se si trovano alla fine della stringa alfanumerica, viene considerata tutta la stringa. Inoltre il programma prende anche cartelle in input per analizzare i file o le sottocartelle se l’opzione necessaria è attiva. Il risultato viene riportato nel file di output chiamato “swordx.out”, tuttavia è possibile impostare un altro file di output tramite un opzione. Le opzioni, con relativa descrizione e implementazione, sono descritte nella descrizione del file swordx.c. Il programma è anche in grado di analizzare i file anche se non presenti nella cartella swordx, purché venga specificato il percorso per quel file. L'esecuzione avviene in single-threaded e, a causa dell'impiego della lista concatenata, risulta essere piuttosto lenta nel caso di file di grandi dimensioni, visto che la complessità della lista nel caso peggiore è O(n^2). Nel caso ottimo invece la complessità è di molto minore (O(n) rispetto a quella del caso pessimo, e ciò si evince anche dal tempo di esecuzione richiesto. Una migliore implementazione del programma potrebbe essere fatta implementando la struttura dati albero e magari, svolgere operazioni di lettura di più file assegnando ciascuno ad un thread in modo da sfruttare il parallelismo. Per quanto riguarda le espressioni regolari sono stati fatti dei test con l’espressione regolare *, grazie all’impiego del metodo isRegular. Una volta compilato il programma con il comanda make (facendo make clean se si è già compilato il programma prima), l’esecuzione si effettua indicando i file, cartelle e opzioni scelte precedute da ./swordx .

Librerie

Tutte le librerie utilizzate fanno parte della libreria standard di C implementata da GNU, la glibc. Nei sistemi GNU/Linux non è pertanto necessaria alcuna installazione aggiuntiva, mentre per la portabilità del programma su altri sistemi, come Windows, potrebbe essere necessario importare nel programma il codice sorgente di Gnulib, la libreria GNU per il porting su altri sistemi.

Le librerie ISO utilizzate sono ctype.h, stdio.h, stdlib.h e string.h.
-La libreria ctype.h contiene funzioni utili a stabilire di che tipo sia un certo carattere e viene usata nella lettura dei caratteri nel file.
-La libreria stdio.h fornisce procedure, costanti e dichiarazioni di funzioni e tipi usati nelle operazioni di input/output da file, come fopen, perror, fclose, FILE e la macro EOF.
-La libreria stdlib.h che viene usata per diverse funzioni per gestire la memoria, come malloc, calloc, free, realloc ecc.
-La libreria string.h che viene usata per manipolare le "stringhe", ovvero array di caratteri.

Le librerie POSIX utilizzate sono dirent.h, sys/stat.h, sys/types.h, unistd.h, time.h.
-La libreria dirent.h permette la gestione delle cartelle; viene utilizzata per scandire file e le eventuali sottocartelle delle directory passate input;
-la libreria sys/types.h che serve per dichiarare i vari tipi di dati usati comunemente;
-la libreria sys/stat.h viene inclusa per sfruttare la funzione lstat e distinguere, quindi, i tipi di file tra file regolare, cartella e link simbolico;
-la libreria unistd.h consente l'accesso alle API di sistema dello standard POSIX come read, write o fork;
-la libreria time.h che comprende funzioni utili a manipolare date e il tempo come clock usata per calcolare il tempo.

La libreria getopt.h, presente in glibc, definisce le funzioni getopt_long, getopt_long_only e la struttura dati struct option, utilizzate per la lettura di opzioni e i loro argomenti in input a linea di comando.

Makefile

La gestione della compilazione è affidata a GNU Make, la struttura delle directories è la seguente:
- in src si trovano i file sorgente;
- in lib si trovano gli header files;
- in obj troviamo i codici oggetto generati.

Nel makefile creo una variabile CC per il compilatore, OBJD per la folder dei file .o, LIBD per la folder della libreria, CFLAGS per i parametri di compilazione di tutti i moduli, infine OBJ che comprende la lista dei file.o da generare. Attraverso i vpath ho indicato un percorso virtuali che i file .c che sono in src ed uno per i file.h che si trovano all'indirizzo della variabile LIBD dichiarata prima. Poi per ottenere il file swordx deve eseguire tutti i file.o, ma per eseguire i file.o deve compilare tutti i file.c corrispondenti.
Infine c è un phony target per l'opzione clean, onde evitare ambiguità con l'eventuale presenza di un file di nome clean. L'argomento -I$(LIBD) specifica la cartella dove sono inseriti gli header files.

list.c

Questo modulo implementa la struttura List come una lista concatenata avente un campo parola di tipo char* ed un campo occorrenza di tipo int per ogni nodo, che indicano rispettivamente la stringa riconosciuta nella lettura del file in cui si trova, e quante volte quella stringa è presente nel file. Ogni nodo inoltre ha un puntatore al nodo successivo, di nome next, in modo tale da scorrere la lista verso il basso. Questa struttura viene dichiarata nel file list.h, dove vengono dichiarati anche i metodi definiti in list.c. Il primo nodo di ogni lista creata è un nodo con i campi impostati a NULL per poter definire una lista nel programma main e per comodità di creazione degli altri nodi. Chiaramente del primo nodo non ne viene considerato il campo parola (non ci sarà un nodo con campo NULL sul file dove si stampa il risultato dell'esecuzione), e non figurerà nel file di log.

createListElement
Il metodo createListElement serve a creare il primo nodo di una lista di tipo List, allocando spazio di memoria per un campo parola e uno per occorrenza con malloc, e assegnargli valore NULL.

createListElement2
Il metodo createListElement2 serve a creare un nodo della lista, passata come parametro, con campo parola uguale a quella passata come parametro con la rispettiva occorrenza impostata ad 1. La funzione alloca spazio in memoria per il nodo con gli attributi definiti come detto, scorre la lista fino a quando il puntatore al nodo successivo è NULL, quindi fino alla fine della lista, e pone il nodo creato alla fine della stessa.

createListElement1
Il metodo createListElement1 crea un nodo della lista passata in input con gli attributi anch'essi passati come parametri.
Si alloca lo spazio con malloc per il nodo e l'attributo parola di quel nodo, si copia la stringa passata come parametro nello spazio di indirizzamento della parola del nodo creato e l'occorrenza data in input, e si scorre la lista fino alla fine per aggiungere il nodo. Questo metodo viene utilizzato per creare la lista della precedente esecuzione dell'opzione update.

contains
Il metodo contains permette di controllare se la parola letta dal file che si sta leggendo, è già presente nella lista o no. Se la lista è nulla crea il primo nodo della lista (in realtà è il secondo nodo visto che il primo nodo di ogni lista è nullo), altrimenti si fa la comparazione della stringa oggetto di studio e ciascun campo parola dei nodi che compongono la lista. Se la stringa è già presente in uno dei campi parola si incrementa l'occorrenza di quel nodo con il metodo incrementOcc, invece se la stringa è nuova si crea un nodo nuovo con il metodo createListElement2.

isIgnored
Il metodo isIgnored prende in input una stringa e un puntatore ad una serie di puntatori che "puntano" a stringhe. Sostanzialmente si compara le stringhe dell'array con la stringa data in input tramite il metodo strcmp e si restituisce il valore 1 se quest'ultima è uguale ad una delle stringhe dell'array o si restituisce il valore 0 altrimenti.

parlog.c

Questo modulo implementa la struttura dati Parlog, anch'essa è una lista concatenata e quindi molto simile alla struttura List. Questa struttura dati ha un campo name che include il nome del file da analizzare, il numero di parole considerate del file nel campo cw, il numero delle parole ignorate o scartate da altre opzioni eventualmente attive del file nel campo iw e il campo time che specifica il tempo necessario all'esecuzione del programma per quel file. Ciascun nodo della struttura include un puntatore al nodo successivo chiamato next. Questa struttura viene utilizzata quando è attiva l'opzione log che stampa su un file il contenuto dei campi di ciascun nodo della struttura. Se un dato file da esaminare è contenuto in una directory, il campo name conterrà il path che porta al file. Le parole considerate dei file letti non coincide con l'esatto numero di parole presenti nel file, esclude quelle parole non richieste da opzioni attive che finiscono nel campo delle parole ignorate. Il tempo del campo time di ciascun nodo è espresso in secondi e, come detto, si riferisce al tempo di CPU necessario per eseguire il task.

createLogFile1
Questo metodo viene utilizzato per creare il primo nodo per motivi di semplicità.
Il metodo createLogFile1 alloca spazio di indirizzamento per il primo nodo della struttura Parlog con il metodo sizeof e imposta il campo name a NULL e gli altri campi a 0 del nodo stesso.

createLogFile
Questo metodo crea gli altri nodi di una struttura Parlog a partire dal primo nodo NULL con i campi impostati ai rispettivi valori passati come parametri.
Si alloca lo spazio in memoria con malloc e sizeof, si duplica il nome del file nel campo name e si impostano gli altri valori, e infine si scorre la struttura per porre il nodo creato alla fine della stessa.

swordx.c

Questo modulo comprende tutte le funzioni necessarie ad eseguire le funzionalità richieste dall'utente, tra cui il programma main. All’inizio del programma sono stati definiti dei flag, sono stati dichiarati metodi che successivamente verrano descritti e le seguenti variabili per le opzioni corrispondenti:

```
static char* wordToIgnore= NULL;
static char *fileToExclude = NULL;
static int numMin = 0;
static char **wordsToIgnore = NULL;
static char *logFile= NULL;
static Parlog* firstLogFile = NULL;
static char *updateFile;
char* outputFile = "swordx.out" ;
```

Le funzionalità delle opzioni sono state implementate in vari modi a seconda della loro tipologia:
-le funzionalità che non richiedono argomenti sono implementate per mezzo di flag, ossia variabili statiche intere che possono assumere valore 0 o 1 se vengono rilevate le opzioni rispettive. Queste funzionalità sono la ricorsione tramite il recursive_flag e la rispettiva voce recursive all’interno

della struct long_options, per analizzare le sottocartelle delle eventuali cartelle in input; l'opzione follow tramite il follow_flag e con la rispettiva voce nella struttura delle option, in modo tale che il programma analizzi anche i link ad altri file testuali, e quindi, contare anche le parole in quei file; l'opzione alpha tramite l' alpha_flag e con la rispettiva voce nella struttura option, in modo che il programma conti le sole stringhe composte da soli caratteri alfabetici; l'opzione sortbyoccurrency tramite il sort_flag e con la propria voce nella struct option, per ordinare in crescenza le stringhe in base alla propria occorrenza.
-la funzionalità help che stampa nel terminale le istruzioni per eseguire il programma che, sebbene non richieda argomento, non ha un suo flag, ma figura come una short option;
-le restanti funzionalità figurano come short option, ossia opzioni non dotate di flag che si indicizzano mediante un valore che, nel caso di questo programma, si basa sul primo carattere del loro nome, e dotate anche di un argomento che le segue e che viene posto nella variabile optarg. Queste funzionalità sono la funzione explude che esclude un file dal conteggio delle parole (si usa all'interno della directory per scegliere quali file contare o no), la funzione min per escludere quelle parole di lunghezza inferiore a quella del parametro di tale opzione, la funzione ignore che ignora delle parole che sono contenute nei file, se queste sono contenute nel file passato come argomento, la funzione log per ottenere informazioni riguardo ai file elaborati su un file passato come argomento, la funzione output per cambiare il file di output dove vengono scritti i risultati e la funzione update, che fa la differenza tra la lista di parole precedentemente elaborate con quella che viene richiesta al momento dell'ultima esecuzione del programma.
In quest'applicazione è stata usata getopt_long_only, la quale si avvale della struttura struct option chiamata long_options, che include un opzione per ogni funzionalità aggiuntiva richiesta dall'utente al momento dell'esecuzione. Ogni elemento della struttura ha 4 campi:
-const char* name, ossia il campo che indica il nome dell'opzione a cui si riferisce quell'elemento;
-int has_arg, ovvero il campo che indica se è richiesto o meno un argomento per eseguire quella funzionalità ;
- int *flag, che indica se quell'elemento della struttura imposta un flag per attivare la funzionalità a cui si riferisce;
-int val, che identifica l'opzione a cui si riferisce quell'elemento.
Fondamentalmente se il flag è un null pointer (ad esempio con valore 0), il campo val identificherà quell'opzione altrimenti, se non lo è, il flag indicherà l'indirizzo di memoria in cui risiede il valore della variabile flag assegnata a quell'option. Il campo val in tal caso indicherà il valore che la variabile assumerà quando, all'esecuzione del programma, quella data opzione viene riconosciuta.
Il metodo getopt_long_only ha come parametri argc, la lunghezza dell'array argv, ossia l'array che custodisce le opzioni che si sono presentate all'esecuzione dell'app, il parametro char *shortopts, in cui si dichiarano le short option, ossia quelle che hanno il parametro del flag nullo(riconoscibili dal loro campo val nella struct e che presentano il carattere ':' se richiedono un argomento), il parametro *longopts che include quelle con il flag e il parametro indexptr che custodisce l'indice della long option registrata nell'option struct.
Infine segue uno switch dove comparare il risultato del metodo descritto con i valori delle short option riconosciute dai case , il valore 0 per le option che hanno il flag non nullo e un case di default che ferma l'esecuzione del programma. In questo programma le short option sono considerate come le opzioni che hanno un flag nullo, mentre le altre sono long option.
Quando getopt_long_only riscontra una short option ritorna il carattere che codifica l'opzione, si richiama una funzione (se definita dal case dello switch) e mette l'argomento di essa in optarg.
Quando la funzione incontra una long option agisce in base al valore del flag e a quello di val: se il flag è null, la funzione ritorna il contenuto di val per indicare l'opzione riscontrata mentre, se il flag non è null bisogna mettere l'indirizzo del flag nel campo flag e in val il valore 1 e la funzione restituirà il valore 0. In questo caso il while si ferma con break poichè viene impostato un flag. Per ciascuna long option, getopt_long_only ne pone l'indice nella struttura longopts in modo da poter prendere il nome dell'opzione quando si vuole e se quella long option ha un argomento, quest'ultimo viene posto in optarg. La funzione utilizzata differisce da getopt_long, per il semplice fatto che

quest'ultima non riconosce le opzioni dell'utente a meno che non siamo precedute dal carattere '--'. La funzione scelta accetta anche il carattere '-'.

lettura/controlloStringhe, readIgnore. Implementazione option alpha, min e ignore
La funzionalità della lettura dei caratteri dai file viene svolta attraverso la funzione lettura, la quale legge dal file di input tutti gli insiemi di caratteri, separati da spazi bianchi o invii, fino alla fine del file. Per far ciò utilizza un array di caratteri di dimensione 100 per le stringhe lette, una struttura List per contare quante parole registrate e quante ignorate per la funzionalità di log dei file, una macro FILE per utilizzare fopen per aprire il file da leggere e un while per leggere tutte le parole fino alla fine dello stesso, in modo da fare il controllo delle stringhe con le opzioni rilevate all'esecuzione tramite il metodo controlloStringhe.
La funzione readIgnore serve per leggere dal file passato come argomento le parole da saltare nel conteggio delle stringhe, creando un puntatore ad una lista di stringhe.
La funzione controlloStringhe ha come parametri la lista da gestire, un puntatore alla stringa letta dal metodo lettura, l'alpha flag, il limite di lunghezza numMin e il puntatore ad un array di stringhe per le stringhe ignorate wordsToIgnore. Viene allocato in memoria uno spazio per ogni carattere della stringa letta, si controlla se un carattere è numerico e, se alpha_flag è attivo, si ritorna null, altrimenti si copia carattere per carattere la stringa dentro allo spazio appositamente creato prima. Successivamente si controlla si controlla se la stringa rispetta il requisito della lunghezza (se non specificato il limite inferiore è 0, quindi tutte le stringhe sono ben accette) e se la stringa fa parte delle stringhe ignorate. Se i requisiti sono rispettati si fa ricorso alla funzione contains per vedere se un nodo della lista ha già registrato questa stringa e se ne incrementa l'occorrenza, altrimenti se ne crea uno nuovo. Se invece i limiti non sono rispettati la funzione restituisce null.

StampaFile, funzione per stampare le liste di stringhe e occorrenze sul file indicato
La funzione per stampare le liste di nodi è StampaFile che stampa sul file indicato, che di default è swordx.out, le strutture di tipo List. Apre il file con una macro FILE e la chiamata fopen e, finché non si raggiunge la fine della lista scorrendola con un ciclo while, si stampa la stringa del nodo, uno spazio, l'occorrenza e un carriage return. Attraverso l'opzione o oppure output è possibile cambiare il file di output di default, con quello indicato come parametro.

Opzione sortbyoccurency, implementata tramite il metodo sort
Se il sort_flag è attivo, si crea un array di caratteri di grandezza pari al doppio dei caratteri del nome dell'output file, e alla lunghezza della stringa " sort -n -o" dove mettere il comando " sort -k 2 -n -o ", a cui concatenare due volte il nome del file di output, separati da uno spazio. Questo array viene passato come argomento alla funzione system che esegue l'ordine delle stringhe in base al numero dopo lo spazio delle parole, ossia le occorrenze. Se il flag non è attivo è previsto un' altra chiamata di system che ha come argomento un insieme di stringhe che non impone l'ordine delle occorrenze.

checkName e filneInDir, implementazione opzioni recursive, follow e explude
La funzione checkName prende in input il file/cartella da leggere e una lista su cui creare i nodi a partire dal primo nodo nullo. Si controlla se il parametro è una directory: se non lo è si verifica se l'opzione del file d log è attiva e, in tal caso si passa al metodo che la implementa, altrimenti si fa la semplice lettura descrittta precedentemente; se il parametro è una directory si passa al metodo fileInDir. Il metodo fileInDir sfrutta la struttura di tipo dirent e la macro DIR per aprire la directory, escludere i file indicati dall'opzione explude (se la variabile fileToExclude è diversa da NULL e facendo la comparazione tra i file della directory e i file specificati dall'opzione), seguire i link tramite il follow_flag (se attivo e se il nome del parametro è un link), analizzare le sottodirectory se viene passata una directory come parametro e se il recursive_flag è attivo, oppure se non si ha a che fare con una sottodirectory o un link e il logFile é NULL, si fa la semplice lettura. In ciascun caso si eseguirà il metodo UpdateListLog per le informazioni sui file.

WriteLog, UpdateListLog e counter. Implementazione opzione log
Il metodo UpdateListLog serve per calcolare il tempo di esecuzione della lettura del file in secondi, creare il nodo della struttura Parlog di riferimento con i campi appropriati (il metodo counter permette di contare le parole registrate e quelle ignorate per ogni file, utilizzando un array con due posizioni da incrementare di 1 a seconda del caso), fare la lettura, dato che se in checkName la variabile logFile fosse diversa da NULL, la lettura non si farebbe per quel parametro e resettare i campi dell'array count utilizzato da counter. Il metodo WriteLog scrive sul file indicato come parametro la struttura Parlog utilizzato con lo stesso sistema di StampaFile.

ConfrontaOccorrenza, confronto, copy. Implementazione opzione update.
Il metodo copy si occupa di copiare dal file di output della precedente esecuzione del programma tramite due array di caratteri rappresentanti la parola e l'occorrenza del nodo, convertendo la stringa dell'occorrenza in numero e creando un nodo della lista da confrontare con quella nuova, ossia della successiva esecuzione del programma. Infine il metodo confronto prende le due liste in input e fa il confronto nodo per nodo tra la lista precedente e quella successiva: se le parole di due nodi delle liste sono uguali, si fa la differenza con il metodo confrontaOccorrenza e le occorrenze vengono poste uguali a 0 e, se la differenza tra le occorrenze è minore di 0 si stampa il segno -, altrimenti il segno + prima della differenza (salvata in una variabile intera) ; se ci sono parole della vecchia lista queste vengono scritte sul file precedute dal segno - (poiché non presenti nella lista nuova); se invece, una volta controllata la lista precedente, rimangono alcune parole della lista nuova queste vengono stampate sul file come viene fatto per i nodi confrontati con segno + (per evitare di stampare le parole precedentemente analizzate, si scrivono solo quelle che non hanno occorrenza 0, saltando così quelle precedentemente confrontate della nuova lista).

Il programma main
Il main contiene la struttura option long_options con tutte le opzioni già descritte, si confrontano le opzioni riconosciute con lo switch, che tratta le short options con un case per ognuna salvando il proprio parametro in optarg (se presente), invocando metodi opportuni e un break. Presenta il caso in cui non sia un short option un case 0 che indica tutte le altre long options che hanno attivato un flag, e un case di default che ferma l'esecuzione del programma in caso di opzione non riconosciuta, ma esegue quella o quelle riconosciute. Successivamente salva tutti gli argomenti delle opzioni in un puntatore ad una serie di stringhe, che fa il checkName per ogni di questi e, se il logFile è diverso da NULL, scrive la struttura dell'opzione log sul file indicato; stessa cosa per l'opzione update, ossia se la variabile non è NULL, si copia la lista di stringhe dal file su cui c è stata la precedente esecuzione e si fa il confronto con la nuova. Si fa il sort, sia che il flag sia attivo o no e, se non ci sono input files viene stampata una stringa di errore sul terminale. Per riconoscere se i parametri siano stati link o cartelle sono stati utilizzati i metodi isDirectory e isLink.

Test
Per quanto riguarda i test ho usato dei file chiamati test, test1, test2 e words_italian.win.txt che pesa 560 kb. Ho anche utilizzato una cartella chiamata tests che racchiude un link a una sottocartella, una sottocartella, un link che si riferisce ad un file esterno e atri due file. E' inoltre presente il file di output swordx.out ma anche un altro file per testare il funzionamento del parametro output per definire un altro output e il file logFile per testare i log. I test fatti dimostrano l'effettivo funzionamento del programma, sebbene l'adozione di questo tipo di struttura lista comporti un maggior tempo nell'analisi dei file più grandi.
Eseguendo questa richiesta ./swordx tests -r -f -a -s -o swordx1.out al programma, essa restituirà sul file indicato dal parametro o il risultato

gatto 2

maio 2
cane 3

da cartella tests che, considerando i link e le sottocartelle, racchiude in tutto
cane1 1
gigino1 1
gatto 2
maio 2
cane 3

se invece si esegue questa richiesta ./swordx -r -f -a -s tests -e testdir -o swordx1.out -h -i wordToIgnore -l logFile

cane 1
maio 1

ossia si tolgono le parole con caratteri numerici e due cane poichè si è escluso il file testdir che ne conteneva due al proprio interno; se si aggiunge il parametro -m 5 non stamperebbe nulla poichè cane e maio hanno meno di 4 caratteri.
Se volessi usare ./swordx -r -f -a -s tests -e testdir -i wordToIgnore -l logFile -m 5 -update swordx.out su swordx.out mi ritornerebbe

cane (-1)
maio (-1)