Esercizio 13 (Strumenti: Lex)

Si scriva un file .lex per:

- riconoscimento delle parole del linguaggio di esercizio 1
- riconoscimento delle parole del linguaggio di esercizio 3
- riconoscimento delle parole del linguaggio di esercizio 5
- la lettura da file di frasi contenenti parole del linguaggio di esercizio 6. Deve generare un file contenente:
    1. il numero totale di parole corrette trovate.
    2. il numero totale di caratteri incontrati.
    3. il numero totale di caratteri ignorati perche' non formanti parole.
    4. la sequenza di parole riconosciute

Esercizio13a

Lex e' uno strumento progettato per manipolare stringhe e individuare sottostringhe di esse che soddisfino vincoli esprimibili mediante espressioni regolari. Pertanto puo' essere utilizzato, come noi facciamo, per "studiare" grammatiche regolari ma non e' concepito per tale scopo e il suo uso non richiede specifiche competenze. Questa premessa e' obbligatoria per capire come talora possa accadere che soluzioni formalmente corrette, complete e efficenti non trovino piena esprimibilita' in Lex, o altrimenti risultino piu' inefficenti di soluzioni che dovrebbero avere costi maggiori, o infine scritture scorrette si rivelino in Lex soluzioni lecite e complete. Di fatto Lex nasconde alcuni meccanismi (quali la complementazione) non esprimibili nelle grammatiche regolari.
Pertanto daremo due soluzioni Lex.
La prima e' una grammatica regolare che costruisce le frasi del linguaggio osservando che:

- possiamo ordinare il nostro alfabeto
- data una qualunque parola w del linguaggio, sia "a" la lettera di valore minore che occorre nalla parola, ed n il numero di occorrenze di "a" in w. Allora, w=w0.aw1. ... . awn, dove w0 e wn possono essere vuote, e w1,...,wn-1 sono parole del linguaggio non contenente "a".
- Ad ogni lettera dell'alfabeto associamo il sottolinguaggio delle parole che contengono tale lettera e che sono strutturate come w sopra.

Notiamo che in questo modo ad ogni lettera possiamo far corrispondere una categoria grammaticale e tale categoria puo' essere espressa facendo ricorso alle categorie grammaticali di lettere di valore maggiore: concludiamo che le categorie grammaticali sono totalmente ordinabili e la gramatica e' quindi regolare.
La seconda soluzione si basa sull'uso del meccanismo di complementazione. A tale soluzione non corrisponde tuttavia, alcuna grammatica regolare.

IN LEX

```
%{
int count1=0;
int count2=0;
int tot=0;
%}


special ["@"|"#"|"$"|"%"|"^"|"&"|"*"|"("|")"|"_"|"-"|"+"]
char [a-zA-Z]
digit [1-9]
separatore [ \t\n]

doppie
{char}*[aa|bb|cc|dd|ee|ff|gg|hh|ii|ll|mm|nn|oo|pp|qq|rr|ss|tt|uu|vv|zz]{char}*

ide {char}*

%%

{doppie} {++count1;printf("<IDE2,%s>",yytext);tot=tot+yyleng;}
{separatore} {++tot;}
{special} {++tot;}
{ide} {++count2;printf("<IDE1,%s>",yytext);tot+=yyleng;}
. {++tot;printf("");}

%%

int yywrap(void){
printf("\nPAROLE SU [a-z] CONTENENTI DOPPIE TROVATE: %d", count1);
printf("\nTOTALE PAROLE TROVATE: %d", count1+count2);
printf("\nLETTI %d CARATTERI", tot);
return 1;
}
```