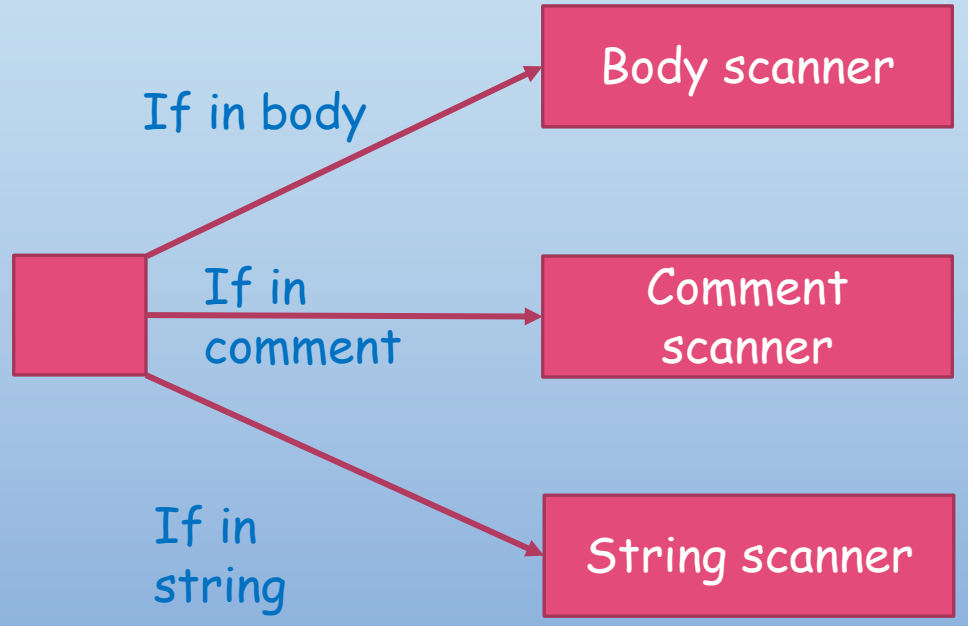
# START CONDITIONS

# START CONDITIONS

Gli scanner sono automi. Quando lo scanner deve compiere compiti diversi a seconda di diverse situazioni, può essere utile permettergli di scegliere "quale" scanner deve attivare a seconda delle situazioni. Le start conditions permettono appunto questa flessibilità.

Le start conditions permettono di combinare piccoli parser in uno più grande



# START CONDITIONS

Flex fornisce un meccanismo di attivare regole in modo condizionato.

Le condizioni start sono **dichiarate** nella sezione delle dichiarazioni con **`%s`** o **`%x`** (condizioni inclusive o esclusive) seguite dal nome assegnato (per esempio, n_start_cond).

Una condizione START è **attivata** all'interno di un'azione usando **`BEGIN`** `nome_start_cond`. Da quel momento lo scanner si trova nella condizione START indicata fino all'attivazione di una nuova start condition;

- Se la condizione è inclusiva le regole che non si trovano in nessuna delle condizioni START saranno attive
- Se la condizione è esclusiva le sole regole ad essere attive saranno quelle qualificate con la condizione START attivata.

# START CONDITIONS

L'automa attivato dalla start condition, ossia le azioni che l'analizzatore lessicale deve svolgere quando si attiva la start condition

`n_start_cond`, viene preceduto dal nome della start condition circondata da parentesi angolari

**`<nome_start_cond>  {istruzioni}`**

# ESEMPIO

Copiare l'input nell'output sostituendo la parola "magic" con la parola "first" in tutte le linee che cominciano per a, con la parola "second" in quelle che cominciano per b e in "third" in quelle che cominciano per c

```
%s aa bb cc
%%
^a {ECHO; BEGIN aa;}
^b {ECHO; BEGIN bb;}
^c {ECHO; BEGIN cc;}
\n {ECHO; BEGIN 0;}
<aa> "magic" {printf("first");}
<bb> "magic"{printf("second");}
<cc> "magic" {printf("third");}
. {ECHO};
```

Dichiarazione delle start conditions (inclusive) aa,bb e cc

Nelle linee che iniziano per a (b,c resp) si attiva la start condition aa (bb,cc, resp.)

Definizione delle azioni a seconda del verificarsi delle start conditions <aa> <bb> <cc>

Ogni volta che si passa a una nuova linea si annullano tutte le start conditions attivate (begin 0 corrisponde ad annullare le start conditions attivate

# PROBLEMA

Scanner che riconosce e elimina quello che è contenuto dentro i commenti mantenendo un contatore delle linee.

Dichiarazione di start condition esclusiva

Nel caso in cui trova /* attiva la start condition

rimuove tutto ciò che non è '*' o newline

```
%{
int line_num = 1;
%}
%x comment
%%
"/*" BEGIN(comment);
<comment>[^*\n]* ;
<comment>"*"+[^/\n]* ;
<comment,INITIAL>\n ++line_num;
<comment>"*"+"/" BEGIN(INITIAL);
```

BEGIN(0) o BEGIN(INITIAL) sono equivalenti e riportano lo scanner allo stato originale in cui nessuna condizione start è attiva.

Incrementa il numero delle linee ogni volta che si trova un newline, sia in un commento che fuori

Rimuovi tutte le '*' non seguite da '/' o newline

# Programma completo

```
%{
int line_num = 1;
%}
%x comment
%option noyywrap
%%
"/*" BEGIN(comment);
<comment>[^*\n]* ;
<comment>"*"+[^*/\n]* ;
<comment,INITIAL>\n {++line_num; ECHO;}
<comment>"*"+"/" BEGIN(INITIAL);
. {ECHO;}
%%
int main(int argc,char *argv[])
{
--argc; /* skip over program name */
if ( argc > 0 )
yyin = fopen( argv[1], "r" );
else yyin = stdin;
yylex();
printf("\n\n");
printf("il numero di linee del testo e' %d", line_num);
printf("\n\n");
return 0;
}
```

# USO DI EOF IN FLEX

Flex ha il pattern <<EOF>> che ha un match solo con la fine del file. La regola speciale <<EOF>> indica azioni che sono eseguite quando si incontra un end-of-file e yywrap() restituisce un valore non nullo, ovvero indica che non ci sono altri file da processare. Le azioni devono terminare preferibilmente facendo una delle seguenti cose:

1. Assegnare yyin a un nuovo file in input.
2. Eseguire un'azione di return
3. Eseguire yyterminate().

Nel caso di input con file multipli, si deve gestire la funzione yywrap.

La regola <<EOF>> non dovrebbe essere usata con altri pattern. Dovrebbe solo essere qualificata con una lista di start conditions. Se non è qualificata da nessuna start condition allora si applica a tutte le start conditions.

# PROBLEMA

Con l'ausilio di flex scrivere un analizzatore lessicale che effettui la tokenizzazione di un testo scritto nel linguaggio UNO. Tale linguaggio è case-sensitive e prevede solo l'utilizzo di caratteri alfanumerici, spazi, tabulazioni e newline. Ogni altro carattere è considerato fuori dal lessico.

Il lessico del linguaggio prevede quattro categorie di token per le quali l'analizzatore assegna un attributo come di seguito descritto:

Token di tipo 1: tutte le parole costituite da lettere maiuscole e cifre e che iniziano per vocale. L'attributo assegnato a tali parole è il massimo numero di cifre pari consecutive. Per esempio, l'attributo di AHUSY43T24627T88I9 è 4.

Token di tipo 2: tutte le parole costituite da lettere (minuscole o maiuscole) che iniziano e finiscono per consonante. Per tali parole l'attributo è la differenza tra il numero di consonanti e il numero di vocali. Per esempio, l'attributo di bgduHspAretiAOUiLgg è 5.

Token di tipo 3: tutte le parole che iniziano con una cifra pari e sono costituite da vocali minuscole e cifre. Per tali parole l'attributo è il numero massimo di vocali comprese tra due cifre dispari consecutive. Per esempio, l'attributo di 6aioeuie25eaiu7ioeauei9uuueioae443526iou è 7.

Token di tipo 4: tutte le parole che contengono lettere maiuscole e cifre e iniziano con una consonante e non contengono 0 consecutivi. Per tali parole l'attributo è sempre 0. Per esempio la parola D3GRGH4U5CIT090DG ha attributo 0

Tutte le altre parole rientrano nella categoria token di Tipo 5 per le quali non è definito nessun attributo.

Il programma deve produrre un file in output contenente una tabella in cui sono elencati TOKEN, ATTRIBUTO e LESSEMA di tutte le parole che rientrano nelle categorie dei token da 1 a 4.

Alla fine il file in output deve contenere anche il numero di righe del testo e il numero dei token di tipo 5.

Per esempio se il file in input è:

AHUSY43T24627T88I9 bgduHsparetisouilG

6aioeuie5eaiu7ioeauei9uuueioae443526iou

QYU900HSJ650HH TYU90HSJ650HH

hjdkkaudAe67738

Il file in output è:

| TOKEN | LESSEMA | ATTRIBUTO |
|---|---|---|
| T1 | AHUSY43T24627T88I9 | 4 |
| T2 | bgduHsparetisouilG | 5 |
| T3 | 6aioeuie5eaiu7ioeauei9uuueioae443526iou | 7 |
| T4 | TYU90HSJ650HH | 0 |

Numero righe: 4

T5: 2

Token di tipo 1: tutte le parole costituite da lettere maiuscole e cifre e che iniziano per vocale. L'attributo assegnato a tali parole è il massimo numero di cifre pari consecutive. Per esempio, l'attributo di AHUSY43T24627T88I9 è 4.

```
%{
 int righe=1, token5=0, attr=0;
%}
VOC     [AEIOU]
voc     [aeiou]
CONS    [B-DF-HJ-NP-TV-Z]
cons    [b-df-hj-np-tv-z]
%option noyywrap
%x T1  T2  T3  T4
%%
{VOC}/[A-Z0-9]+                 {PRINTF("T1 \t \t %s",yytext); begin(T1);}
…
<T1>[02468]+                    {echo; if (yyleng>attr) attr=yyleng;}
…
<T1,T2,T3,T4>[ \t]+             {printf("\t\t%d\n",attr);attr=0;begin(0);}
<T1,T2,T3,T4>\n                 {righe++;
                                printf("\t\t%d\n",attr);attr=0;begin(0);}
<T1,T2,T3,T4><<EOF>>            {printf("\t\t%d\n",attr);
                                attr=0;begin(0);
                                yyterminate();}
```

Token di tipo 2: tutte le parole costituite da lettere (minuscole o maiuscole) che iniziano e finiscono per consonante. Per tali parole l'attributo è la differenza tra il numero di consonanti e il numero di vocali. Per esempio, l'attributo di bgduHsparetisouilgG è 5.

```
%{
 int righe=1, token5=0, attr=0;
%}
VOC      [AEIOU]
voc       [aeiou]
CONS      [B-DF-HJ-NP-TV-Z]
cons      [b-df-hj-np-tv-z]
%option noyywrap
%x T1  T2  T3  T4
%%
...
({cons}|{CONS})/([A-Za-z]*({CONS}|{cons})) printf("T2 \t \t %s",yytext); attr++; begin (T2);}
...
<T2>{cons}|{CONS}{echo; attr++;}
<T2> {voc}|{VOC} {echo; attr--;}
<T1,T2,T3,T4>[ \t]+        {printf("\t\t%d\n",attr);attr=0;begin(0);}
<T1,T2,T3,T4>\n            {righe++; printf("\t\t%d\n",attr);attr=0;begin(0);}
<T1,T2,T3,T4><<eof>>       {printf("\t\t%d\n",attr);attr=0;begin(0);yyterminate();}
```

Token di tipo 3: tutte le parole che iniziano con una cifra pari e sono costituite da vocali minuscole e cifre. Per tali parole l'attributo è il numero massimo di vocali comprese tra due cifre dispari. Per esempio, l'attributo di 6aioeuie25eaiu7ioeauei9uuueioae443526iou è 7.

```
%{
 int righe=1, token5=0, attr=0;
%}
VOC   [AEIOU]
voc      [aeiou]
CONS     [B-DF-HJ-NP-TV-Z]
cons     [b-df-hj-np-tv-z]
%option noyywrap
%x T1   T2   T3   T4
%%
…
[02468]/[aeiou0-9]+                  {printf("T3 \t \t %s",yytext); BEGIN (T3);}
…
<T3>[13579][aeiou]+/[13579]          {ECHO; if (yyleng-1>attr) attr=yyleng-1;}
<T1,T2,T3,T4>[ \t]+                  {printf("\t\t%d\n",attr);attr=0;BEGIN(0);}
<T1,T2,T3,T4>\n                      {righe++;
                                     printf("\t\t%d\n",attr);attr=0;BEGIN(0);}
<T1,T2,T3,T4><<EOF>>{printf("\t\t%d\n",attr);attr=0;BEGIN(0);yyterminate();}
```

Token di tipo 4: tutte le parole che contengono lettere maiuscole e cifre e iniziano con una consonante e non contengono 0 consecutivi. Per tali parole l'attributo è sempre 0. Per esempio la parola D3GRGH4U5CIT090DG ha attributo 0

```
%{
 int righe=1, token5=0, attr=0;
%}
VOC   [AEIOU]
voc      [aeiou]
CONS     [B-DF-HJ-NP-TV-Z]
cons     [b-df-hj-np-tv-z]
%option noyywrap
%x T1   T2   T3   T4
%%
…
{CONS}(0?[A-Z1-9]+)*0?      {printf("T4 \t \t %s\n",yytext); BEGIN (T4);}
…
<T1,T2,T3,T4>[ \t]+         {printf("\t\t%d\n",attr);attr=0;BEGIN(0);}
<T1,T2,T3,T4>\n             {righe++;
                            printf("\t\t%d\n",attr);attr=0;BEGIN(0);}
<T1,T2,T3,T4><<EOF>>{printf("\t\t%d\n",attr);attr=0;BEGIN(0);yyterminate();}
```

Tutte le altre parole rientrano nella categoria token di Tipo 5 per le quali non è definito nessun attributo.

```
[\n]           {righe++;}
[ \t]          ;
[a-zA-Z0-9]+   {token5++;}
%%
```

```
%{
 int righe=1, token5=0, attr=0;
%}
VOC         [AEIOU]
voc         [aeiou]
CONS        [B-DF-HJ-NP-TV-Z]
cons        [b-df-hj-np-tv-z]
%option noyywrap
%x T1   T2   T3   T4
%%
{VOC}/[A-Z0-9]+                              {printf("T1 \t \t %s",yytext); BEGIN (T1);}
({CONS}|{cons})/([a-zA-Z]*({CONS}|{cons}))   {printf("T2 \t \t %s",yytext); attr++; BEGIN (T2);}
[02468]/[aeiou0-9]+                          {printf("T3 \t \t %s",yytext); BEGIN (T3);}
{CONS}(0?[A-Z1-9]+)*0?                       {printf("T4 \t \t %s\n",yytext); BEGIN (T4);}
<T1>[02468]+                        {ECHO; if (yyleng>attr) attr=yyleng;}
<T2>{CONS}|{cons}                   {ECHO; attr++;}
<T2> {VOC}|{voc}                    {ECHO; attr--;}
<T3>[13579][aeiou]+/[13579]         {ECHO; if (yyleng-1>attr) attr=yyleng-1;}
<T1,T2,T3,T4>[ \t]+                 {printf("\t\t%d\n",attr);attr=0;BEGIN(0);}
<T1,T2,T3,T4>\n                     {righe++; printf("\t\t%d\n",attr);attr=0;BEGIN(0);}
<T1,T2,T3,T4><<EOF>>                {printf("\t\t%d\n",attr);attr=0;BEGIN(0);yyterminate();}
[\n]                                        {righe++;}
[ \t]                                       ;
[a-zA-Z0-9]+      {token5++;}
%%
int main(void){
printf("TOKEN \t \t LESSEMA \t \t ATTRIBUTO\n");
yylex();
printf("Numero di righe: %d\n",righe);
printf("Token T5: %d\n", token5);
return 0;
}
```

Dichiarazione start conditions esclusive

Attivazione della start condition T1 quando si trova una vocale solo se seguita da una stringa di lettere maiuscole o cifre

Stampa a video il nome del token (T1,T2,T3,T4) e (dopo i tab) il lessema riconosciuto

Nel caso di nuova riga, incrementare il numero delle righe in ogni condizione e fuori dalle condizioni

Una volta terminato di leggere la parola (si riconoscono spazi, tab o nuova riga o end of file) e calcolato l'attributo, si inserisce nella tabella il valore assegnato all'attributo. Questo viene riinizializzato al valore 0 e si azzerano le precedenti start conditions (begin(0))

# TOKENIZZAZIONE PER C

Scrivere in flex un tokenizzatore per il linguaggio C che riconosca i seguenti lessemi:

Identificatori (che cominciano con lettera), con token id.
Costanti intere con token intconst

| | | |
|---|---|---|
| Main | { | ; |
| Int | } | = |
| Void | ( | + |
| Return | ) | * |

Il programma deve stampare a video in una tabella l'elenco dei token, i relativi attributi (ovvero nel caso degli identificatori, il nome dell'identificatore, nel caso delle costanti, il numero) e il numero di riga in cui il token compare.

I commenti devono essere riconosciuti e devono essere contate le righe che contengono commenti, sia quelli racchiusi tra /* e */ che il commento preceduto da //.

Queste linee di programma devono produrre la tabella a fianco

```
int main(void)
{int x;
x = 19;
x = x * x;
/* Commento
su due righe*/
return 0; }
```

```
Token           lessema  lineno
Tok_int                  1
Tok_main                 1
Tok_lparen               1
Tok_void                 1
Tok_rparen               1
Tok_lbrace               2
Tok_int                  2
ID              x        2
tok_semicolon            2
ID              x        3
tok_equal                3
INTCONST        19       3
tok_semicolon            3
ID              x        4
tok_equal                4
ID              x        4
tok_mult                 4
ID              x        4
tok_semicolon            4
tok_rbrace               4
tok_return               7
INTCONST        0        7
tok_semicolon            7
tok_rbrace               7

2 righe contengono commenti.
```

```
%{
 int n_riga=1; int nr_commenti=1;
%}
%x comment
%option noyywrap
%%
"/*" BEGIN(comment);
<comment>\n {++nr_commenti;}
<comment>. ;
<comment>"*"+"/" BEGIN(INITIAL);
(main|MAIN)                     {printf("MAIN \t \t main \t\t%d\n", n_riga);}
(int|INT)                       {printf("INT \t \t int \t \t%d\n", n_riga);}
(void|VOID)                     {printf("VOID \t \t void\t \t%d\n", n_riga);}
(return|RETURN)                 {printf("RETURN \t \t return \t \t%d\n", n_riga);}
[A-Za-z][A-Za-z0-9]*            {printf("ID \t \t %s\t \t%d\n",yytext, n_riga);}
0|[1-9][0-9]*                   {printf("INTCONST \t %s \t \t%d\n", yytext, n_riga);}
"{"                             {printf("L_BRACE \t { \t \t%d\n", n_riga);}
"}"                             {printf("R_BRACE \t } \t \t%d\n", n_riga);}
"("                             {printf("L_PAR \t \t ( \t \t%d\n", n_riga);}
")"                             {printf("R_PAR \t \t ) \t \t%d\n", n_riga);}
";"                             {printf("SEMICOLON \t ; \t \t%d\n", n_riga);}
"="                             {printf("EQUAL \t \t = \t \t%d\n", n_riga);}
"+"                             {printf("PLUS \t \t + \t \t%d\n", n_riga);}
"*"                             {printf("TIMES \t \t * \t \t%d\n", n_riga);}
[\n]                            {++n_riga;}
[ \t]                           ;
%%
int main(void){
printf("TOKEN \t \t LESSEMA \t RIGA\n");
yylex();
printf("Numero di righe che contengono commenti: %d\n", nr_commenti);
return 0;
}
```

# ESERCIZIO

Con l'ausilio di flex scrivere un programma che preso in input un file di testo ne produca un altro che risulti di dimensione minore, attuando alcune strategie di compressione lossy ad ogni parola del testo (si definisce parola una sequenza di lettere e cifre). In particolare tale programma agisce sulle parole che cominciano per consonante maiuscola nel seguente modo:

1. ogni parola costituita solo da lettere maiuscole viene trasformata in una parola in cui le occorrenze di vocali uguali consecutive vengono compresse in un'unica vocale (uguale) minuscola seguita dal numero di occorrenze.
   - Esempio: ZZEEEEABBAAAAAZZZQRRR viene trasformata in ZZe4a1BBa5ZZZQRRR
2. Ogni parola costituita da una unica consonante maiuscola seguita solo da lettere minuscole viene trasformata secondo la seguente regola.
   - Tutte le occorrenze di vocali consecutive crescenti vengono trasformate nella prima vocale (trasformata in maiuscolo) seguita dall'intero che rappresenta il numero delle vocali soppresse. Per esempio, 'eio' viene trasformata in 'E2'.
   - Esempio: Fsdeiosdaeiodfhiond viene trasformata in FsdE3sdA4fhI2nd

# ESERCIZIO(CONTINUA)

3. Per tutte le parole che iniziano per consonante maiuscola seguita solo da cifre, se tali cifre rappresentano un numero pari esso viene sostituito con l'intero ottenuto dividendo per la massima potenza di 2 per cui il numero è divisibile, concatenato con 'b' e poi concatenato con l'esponente della suddetta potenza.
   - Esempio C120 è sostituito con C15b3 poiché $120=2^3\times15$.
4. Per le parole che cominciano per consonante maiuscola seguita sia da lettere che da cifre: tutte le cifre consecutive vengono sostituite con l'intero che rappresenta la somma di tali cifre. Le lettere devono essere cambiate da maiuscolo in minuscolo e viceversa.
   - Esempio: la parola FdfjdS1293DsFAe32403kOAA è trasformata in fDFJDs15dSfaE12Koaa

Tutte le altre parole devono essere lasciate invariate.

Alla fine della scansione del file il programma deve visualizzare il numero delle sostituzioni di parole effettuate per ciascuna delle tipologie sopra elencate.

# PRIMO CASO

Ogni parola costituita solo da lettere maiuscole viene trasformata in una parola in cui le occorrenze di vocali uguali consecutive vengono compresse in un'unica vocale (uguale) minuscola seguita dal numero di occorrenze.

Esempio: ZZEEEEABBAAAAAZZZQRRR viene trasformata in Ze4a1BBa5ZZZQRRR

```
%{
#include <math.h>
int i, somma, pot;
%}
MAIUSC [A-Z]
vocali [aeiou]
VOCALI [AEIOU]
CONS [A-Z]-[AEIOU]
VOC_CRESC {[A|a]?[E|e]?[I|i]?[O|o]?[U|u]?}
%array
%option main
%x UNO DUE TRE QUATTRO
%%
```

```
MAIUSC/MAIUSC*          {BEGIN(UNO);}
…
<UNO>A+          {printf("a%d", yyleng);}
<UNO>E+          {printf("e%d", yyleng);}
<UNO>I+          {printf("i%d", yyleng);}
<UNO>O+          {printf("o%d", yyleng);}
<UNO>U+          {printf("u%d", yyleng);}
<UNO>.           {ECHO;}
```

# SECONDO CASO

Ogni parola costituita da una unica consonante maiuscola seguita solo da lettere minuscole viene trasformata secondo la seguente regola.

- Tutte le occorrenze di vocali consecutive crescenti vengono trasformate nella prima vocale (trasformata in maiuscolo) seguita dall'intero che rappresenta il numero delle vocali soppresse. Per esempio, 'eio' viene trasformata in 'E2'.
- Esempio: Fsdeiosdaeiodfhiond viene trasformata in FsdE3sdA4fhI2nd

```
%{
#include <math.h>
int i, somma, pot;
%}
MAIUSC [A-Z]
vocali [aeiou]
VOCALI [AEIOU]
CONS [A-Z]-[AEIOU]
VOC_CRESC {[A|a]?[E|e]?[I|i]?[O|o]?[U|u]?}
%array
%option main
%x UNO DUE TRE QUATTRO
%%
```

```
…
[A-Z]/[a-z]*              {BEGIN(DUE);}
…

<DUE> a(e?i?o?u?)  {printf("A%d", yyleng);}
<DUE> e(i?o?u?)    {printf("E%d", yyleng);}
<DUE> i(o?u?)      {printf("I%d", yyleng);}
<DUE> o(u?)        {printf("O%d", yyleng);}
<DUE> u            {printf("U%d", 0);}
```

# TERZO CASO

Per tutte le parole che iniziano per consonante maiuscola seguita solo da cifre, se tali cifre rappresentano un numero pari esso viene sostituito con l'intero ottenuto dividendo per la massima potenza di 2 per cui il numero è divisibile, concatenato con 'b' e poi concatenato con l'esponente della suddetta potenza.

Esempio C120 è sostituito con C15b3 poiché $120=2^3\times15$.

```
%{
#include <math.h>
int i, somma, pot;
%}
MAIUSC [A-Z]
vocali [aeiou]
VOCALI [AEIOU]
CONS [A-Z]-[AEIOU]
VOC_CRESC {[A|a]?[E|e]?[I|i]?[O|o]?[U|u]?}
%array
%option main
%x UNO DUE TRE QUATTRO
%%
```

```
…
{CONS}/[0-9]*          {BEGIN(TRE);}
…
<TRE>[0-9]* {if (atoi(yytext)%2==0)
                        {i=0; pot=1;
                       while(atoi(yytext)<=pot)
                        {i=i+1; pot=2*pot;}
printf("%db%d", atoi(yytext)-pot/2, i-1);};}
```

# CASO QUATTRO

Per le parole che cominciano per consonante maiuscola seguita sia da lettere che da cifre: tutte le cifre consecutive vengono sostituite con l'intero che rappresenta la somma di tali cifre. Le lettere devono essere cambiate da maiuscolo in minuscolo e viceversa.

Esempio: la parola FdfjdS1293DsFAe32403kOAA è trasformata in fDFJDs15dSfaE12Koaa

```
%{
#include <math.h>
int i, somma, pot;
%}
MAIUSC [A-Z]
vocali [aeiou]
VOCALI [AEIOU]
CONS [A-Z]-[AEIOU]
VOC_CRESC {[A|a]?[E|e]?[I|i]?[O|o]?[U|u]?}
%array
%option main
%x UNO DUE TRE QUATTRO
%%
```

```
…
[A-Z]/[A-Za-z0-9]*     {BEGIN(QUATTRO);}
…
<QUATTRO>[0-9]* {somma=0;
                 for (i=0; i<yyleng; i++)
                     somma=somma+atoi(yytext[i]);
                printf("%d%", somma);}
<QUATTRO>[A-Z] {printf("%d%", atoi(yytext)+32);>
<QUATTRO>[a-z] {printf("%d%", atoi(yytext)-32);>
```

```
%{
#include <math.h>
int i, somma, pot;
%}
MAIUSC [A-Z]
vocali [aeiou]
VOCALI [AEIOU]
CONS [A-Z]-[AEIOU]
VOC_CRESC {[A|a]?[E|e]?[I|i]?[O|o]?[U|u]?}
%array
%option main
%x UNO DUE TRE QUATTRO
%%
MAIUSC/MAIUSC*    {BEGIN(UNO); printf(, yytext)}
[A-Z]/[a-z]*      {BEGIN(DUE)}
{CONS}/[0-9]*     {BEGIN(TRE); printf(yytext)}
[A-Z]/[A-Za-z0-9]* {BEGIN(QUATTRO); printf(yytext)}
[A-Za-z0-9]*      {BEGIN 0;}
<UNO>A+                    {printf("a%d", yyleng);}
<UNO>E+                    {printf("e%d", yyleng);}
<UNO>I+                    {printf("i%d", yyleng);}
<UNO>O+                    {printf("o%d", yyleng);}
<UNO>U+                    {printf("u%d", yyleng);}
<UNO>.                     {ECHO;}
<DUE> a(e?i?o?u?){printf("A%d", yyleng - 1);}
<DUE> e(i?o?u?)  {printf("E%d", yyleng - 1);}
<DUE> i(o?u?)    {printf("I%d", yyleng - 1);}
<DUE> o(u?)          {printf("O%d", yyleng - 1);}
<DUE>u              { printf("U%d", 0);}
```

```
<TRE>[0-9]*       {if (atoi(yytext)%2==0)
                                  {i=0; pot=1;
                                  while(atoi(yytext)<=pot)
                                          {i=i+1;
pot=2*pot;}
                                  printf("%db%d",
atoi(yytext)-pot/2, i-1);
                                  };
                            }
<QUATTRO>[0-9]* {somma=0;
                  for (i=0; i<yyleng; i++)
                          somma=somma+atoi(&yytext[i]);
                  printf("%d", somma);}
<QUATTRO>[A-Z]    {printf("%d", atoi(yytext)+32);>
<QUATTRO>[a-z]    {printf("%d", atoi(yytext)- 32);>

. {ECHO;}         ;
```

# ESERCIZIO CON FLEX

Con l'ausilio di flex scrivere un analizzatore lessicale che effettui la trasformazione di un testo scritto in un Linguaggio, chiamato linguaggio AA. Tale linguaggio è case-sensitive e le parole del linguaggio sono costituite solo da caratteri alfanumerici. Due parole possono essere separate l'una dall'altra da spazi, tabulazioni o newline. Ogni parola che contiene almeno un carattere diverso da quelli alfanumerici è considerata una parola non appartenente al lessico.

Le parole del lessico del linguaggio vengono trasformate come di seguito descritto:

1. Ogni parola w costituita da lettere maiuscole che inizia per vocale viene trasformata in una parola ottenuta da w rimuovendone le occorrenze di sequenze di consonanti consecutive di lunghezza dispari. Le sequenze di consonanti consecutive di lunghezza pari vengono trasformate in minuscolo.
   - Per esempio la parola ABSFBERTPLDAWRRTDDORR viene trasformata in absfbeawrrtddorr.
2. Ogni parola costituita da lettere minuscole e cifre e che iniziano per lettera minuscola viene trasformata in una parola in cui, se le occorrenze di cifre consecutive costituiscono un numero naturale pari, questo viene trasformato nella sequenza di cifre che rappresentano la metà del suddetto numero, altrimenti viene trasformato nel numero successivo.
   - Per esempio, la parola cd34rftsy567errer20r viene trasformata in cd17rftsy568errer10r.
3. Tutte le altre parole del lessico vengono trasformate nella parola ottenuta ripetendo il primo simbolo della parola tante volte quanti sono i simboli della parola stessa. Per esempio, la parola oeriiioodivjk34hhfjj4k5 viene trasformata in 0000000000000000000000.

Le parole che non fanno parte del lessico vengono lasciate inalterate. Alla fine della trasformazione il programma deve riportare il numero delle parole del lessico e di quelle che non appartengono al lessico.

Per esempio se il file in input è:

```
ABSFBERTPLDAWRRTDDORR hfjkdjfdh553552%&&5cchhsGhskdlaks89404345asd
0eriiioodivjk34hhfjj4k5 CD34RFTSY567ERRER20R cd34rftsy567errer20r
```

Il file in output è:

Absfbeawrrtddorr hfjkdjfdh553552%&&5cchhs Ghskdlaks89404346asda 0000000000000000000000 CCCCCCCCCCCCCCCCCCCC cd17rftsy568errer10r

Lessico: 5

No lessico: 1