```
/* scanner for a toy Pascal-like language */

%{
/* need this for the call to atof() below */
#include <math.h>
%}

DIGIT   [0-9]
ID      [a-z][a-z0-9]*

%%

{DIGIT}+        {
                  printf("An integer: %s (%d)\n", yytext, atoi(yytext));
                }

{DIGIT}+"."{DIGIT}*     {
                  printf("A float: %s (%g)\n", yytext, atof(yytext));
                }

if|then|begin|end|procedure|function    {
                  printf("A keyword: %s\n", yytext);
                }

{ID}            printf("An identifier: %s\n", yytext);

"+"|"-"|"*"|"/" printf("An operator: %s\n", yytext);

"{"[^}\n]*"}"   /* eat-up one line comments */

[ \t\n]+        /* eat-up whitespace */

.               printf("Unrecognized character: %s\n", yytext);

%%

main(int argc, char **argv) {
   ++argv, --argc; /* skip over program name */
   if (argc>0)
      yyin = fopen(argv[0], "r");
   else
      yyin = stdin;
   yylex();
}
```

'x'	correspond au caractère 'x'
'.'	n'importe quel caractère sauf retour à la ligne (\n)
'[xyz]'	une classe de caractère ; ici le motif reconnu est soit x, soit y, soit z
'j-p'	une classe de caractères ordonnée ; ici les caractères alphabétiques minuscules de j à p
'[^A-Z\n]	le complément d'une classe ; ici n'importe quel caractère qui n'est ni une majuscule ni un \n
'r*'	zéro, un ou plusieurs motif(s) décrit(s) par l'expression rationnelle r
'r+'	un ou plusieurs motif(s) décrit(s) par r
'r?'	zéro ou un motif décrit par r
'r{2,5}'	entre 2 et 5 motifs décrits par r
'r{2,}'	au moins 2 motifs décrits par r
'r{4}'	4 motifs exactement décrits par r
'{def}'	correspond aux motifs décrits par la définition rationnelle def dans la 1ère partie du fichier FLEX
'\x'	l'interprétation C de \x si x est un 'a', un 'b', un 'f', un 'n', un 'r', un 't' ou un 'v' ; le caractère littéral x sinon (sert à annuler la signification d'un opérateur FLEX)
'\0'	le caractère NUL (code ASCII 0)
'\123'	le caractère de code ASCII 123 en octal
'\x2a'	le caractère de code ASCII 2a en hexadécimal
'(r)'	identique à r (sert à gérer les priorités)
'rs'	correspond à un motif décrit par r suivi d'un motif décrit par s
'r|s'	correspond à un motif décrit par r ou par s
'r/s'	correspond à un motif décrit par r à condition qu'il soit suivi par un motif décrit par s (c'est la somme des longueurs des deux motifs qui est utilisée pour vérifier qu'on a affaire au plus long motif reconnaissable par une expression du fichier FLEX, mais yytext ne contient que le motif associé à r)
'^r'	correspond à un motif décrit par r à condition qu'il soit en début de ligne
'r$'	correspond à un motif décrit par r à condition qu'il soit en fin de ligne
'<s>r'	correspond à un motif reconnu par r si l'analyseur est dans le contexte s
'<*>r'	correspond à un motif reconnu par r quel que soit le contexte dans lequel se trouve l'analyseur (y compris un contexte exclusif)
'<<EOF>>'	correspond au motif fin de fichier

Remarques :
- Dans une classe de caractères les opérateurs FLEX perdent leur signification spéciale sauf '\', '-', ']' et '^' s'il est en tête de la classe.
- La liste est donnée par ordre de priorité décroissant des opérateurs FLEX.

Exercice 1 *Devinez quels traitements réalisent les analyseurs produits à partir des quatre fichiers FLEX qui suivent en sachant que les motifs non reconnus sont simplement renvoyés sur la sortie standard. Vérifiez ensuite vos hypothèses en testant ces fichiers sur machine.*

1.
```
%%
```

2.
```
%%
[ \t]+$
```

3.
```
%%
^[^\n]
```

4.
```
%%
[ \t]+$
[ \t]+          printf(" ");
```

Exercice 2 *Voici la spécification FLEX d'un analyseur lexical :*

```
%%
[A-Z][a-z]*     printf("IDENT : %s\n", yytext);
\'[a-z]*\'      printf("CHAINE : %s\n, yytext);
:=              printf("AFFECTATION\n");
\;              printf("SEPARATION\n");
.               /* ignorer le reste */
```

On suppose cette spécification compilée et on appelle lexico *l'analyseur obtenu. Imaginez puis vérifiez sur machine les réactions de lexico sur les entrées suivantes :*

- `Alpha := 'alpha'; Beta := 'beta';`
- `Alpha := 'Alpha'; Beta := 'Beta';`
- `ALPHA := 'alpha'; BETA := 'beta';`

Comment réagit lexico *sur les mêmes entrées quand on remplace la ligne :*

```
\'[a-z]*\'      printf("CHAINE : %s\n, yytext);
```

par :

```
\'.*\'          printf("CHAINE : %s\n, yytext);
```

Exercice 3 *Ecrire un analyseur lexical évolutif auquel l'utilisateur transmet des éléments en précisant leur type, l'analyseur réagissant ensuite lorsqu'il rencontre un des éléments définis précédemment.*

Concrètement les éléments seront des mots et l'analyseur se contentera d'afficher leur type quand il les rencontrera et ce selon la forme décrite au travers de l'exemple suivant (l'utilisateur tape en gras et l'analyseur répond en clair) :

verbe est suis sont étais étaient être a as ont
est
est : verbe
nom chien chat maison assiette
verbe courir sauter manger
manger chien
manger : verbe
chien : nom
adj gros grand étroit frugal
pronom un une des
nom verbe
un gros adverbe n'est pas un verbe
un : pronom
gros : adj
adverbe : non reconnu
n : non reconnu
est : verbe
pas : non reconnu
un : pronom
verbe : nom

Pour réaliser ce travail l'analyseur doit stocker sous la forme d'une liste chaînée les mots et leur type au fur et à mesure qu'ils sont donnés puis consulter cette liste pour afficher le type des mots apparaîssant dans une ligne qui ne commence pas par un des mots clés : verbe, nom, adj ou pronom.