Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

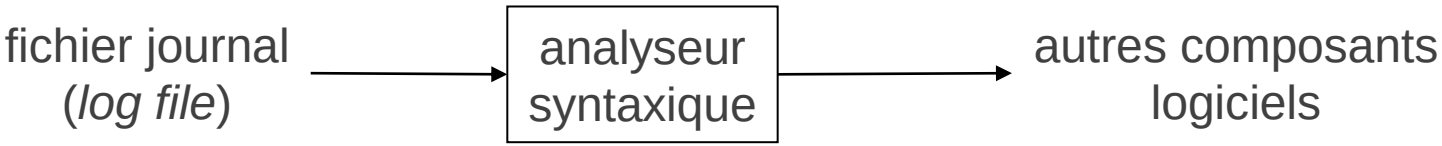
Analyse lexicale

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Qu'est-ce qu'un analyseur syntaxique (*parser*) ?




```
main -- time : 1543098418.019513s
construire_arbre_profiler -- time : 1543098418.019639s
debut_arbre_p -- time : 1543098418.019664s
indice_nom -- time : 1543098418.019696s
hachage -- time : 1543098418.019712s
END -- time : 1543098418.019725s
alloue_lst_nom -- time : 1543098418.019740s
END -- time : 1543098418.019750s
END -- time : 1543098418.019758s
alloue_noeud_profiler -- time : 1543098418.019767s
END -- time : 1543098418.019776s
nouvelle_ligne -- time : 1543098418.019784s
END -- time : 1543098418.019795s
choix_prochain_noeud -- time : 1543098418.019804s
END -- time : 1543098418.019813s
reste_arbre_p -- time : 1543098418.019822s
indice_nom -- time : 1543098418.019830s
hachage -- time : 1543098418.019839s
END -- time : 1543098418.019848s
```

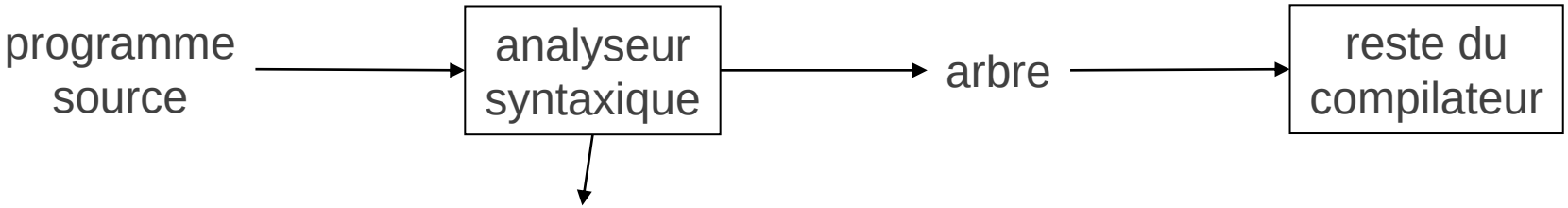
Un composant logiciel qui analyse un fichier texte

Analyser = identifier chaque partie

Si le fichier d'entrée est un fichier journal

Mots-clés, dates, heures...

Qu'est-ce qu'un analyseur syntaxique ?




```
while b != 0
   if a > b
      a := a - b
   else
      b := b - a
return a
```

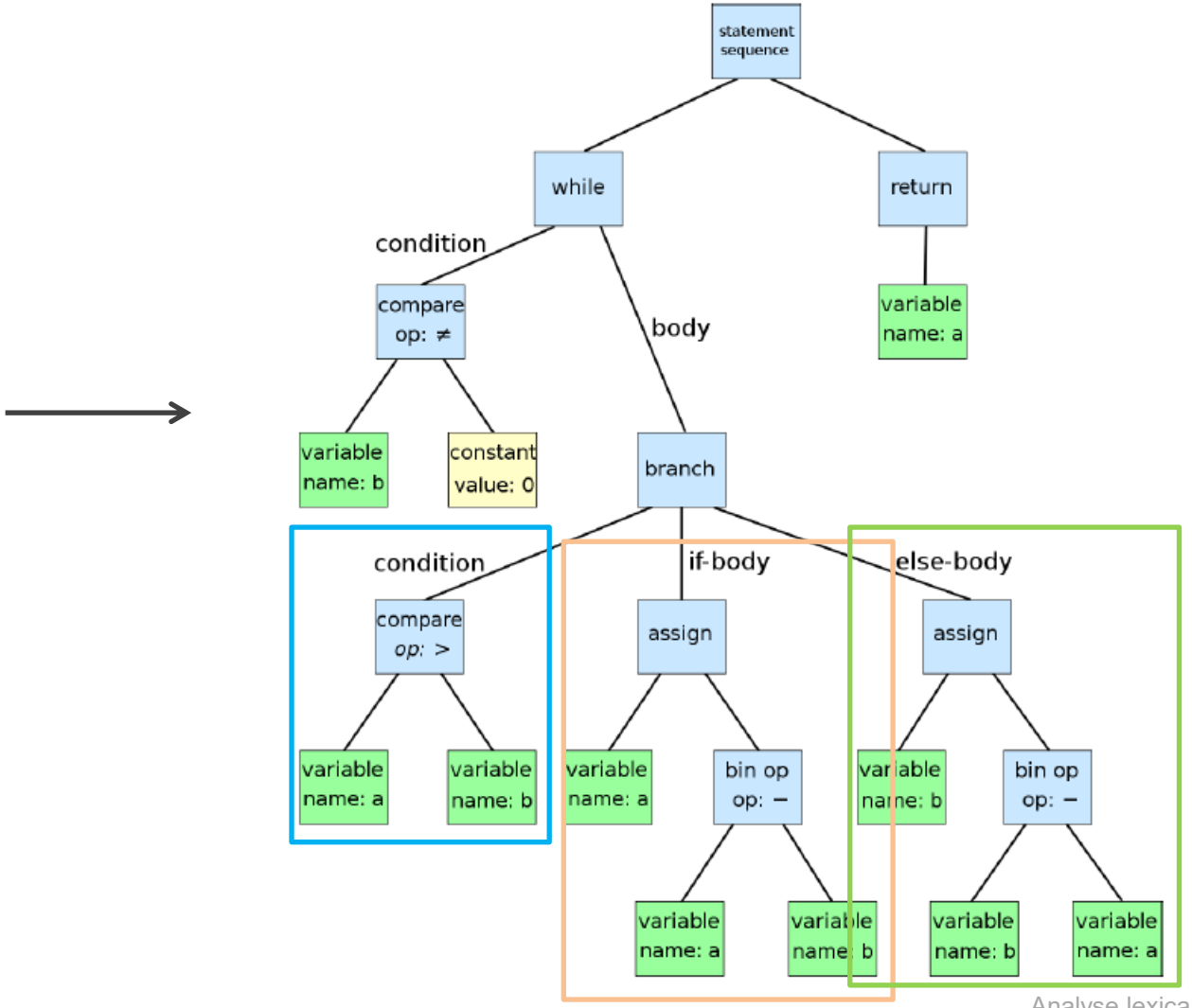
Dans un compilateur

Identifier chaque partie

Instructions, expressions, fonctions...

Qu'est-ce qu'un analyseur syntaxique ?

```
while b != 0
   if a > b
      a := a - b
   else
      b := b - a
return a
```



Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Objectifs du cours

```
> gcc toto.c
toto.c:1: erreur d'analyse syntaxique before ',' token
/tmp/ccImHOOb.o(.text+0x27): In function 'main' : undefined reference to 'get'
toto.c: Dans la fonction "main" :
toto.c:5: attention : suggest explicit braces to avoid ambiguous 'else'
```

- Savoir écrire des analyseurs syntaxiques, de fichiers journaux (*log files*) par exemple
- Savoir utiliser les logiciels Lex/Flex et Yacc/Bison
- Développer un analyseur syntaxique pour un compilateur (projet)
- Comprendre les messages d'erreur de syntaxe des compilateurs
- Savoir lire et écrire des grammaires

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Planning

12 cours, 12 TD sur machine

1 projet de programmation 40 %

1 examen final 60 %

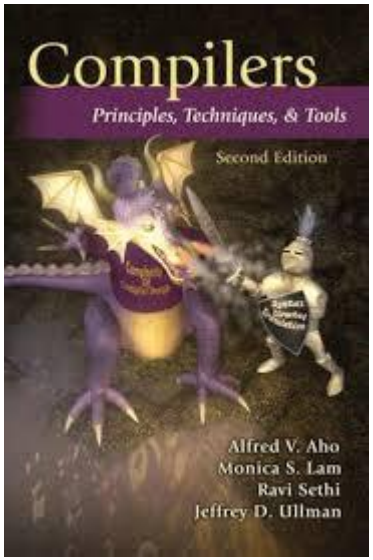
Le projet de programmation est obligatoire

Ne pas rendre un projet entièrement vide

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Bibliographie



Aho, Sethi, Ullman, 1986/2007. *Compilateurs. Principes, techniques et outils*, Pearson Education France

Levine, Mason, Doug, 1990. *Lex & Yacc*, O'Reilly.

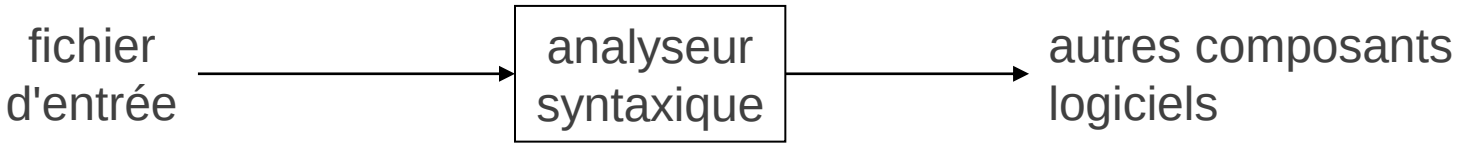
Ce support de cours est inspiré de Aho *et al.* (1986)

Université Gustave Eiffel

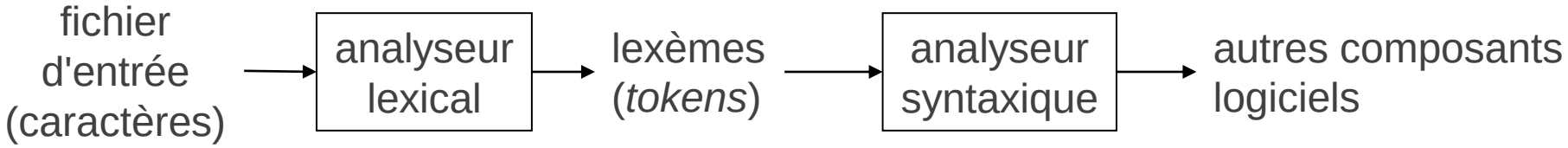
INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Qu'est-ce qu'un analyseur lexical (*lexer*) ?



Un analyseur syntaxique traite le fichier d'entrée caractère par caractère



ou (mieux) lexème par lexème

Analyseur lexical pour un compilateur

```
position = initial + vitesse * 60
```

```
[id, 1] [=] [id, 2] [+] [id, 3] [*] [60]
```

Identificateurs = noms choisis par les développeurs : variables, fonctions...

Analyseur lexical

Trouve les lexèmes :
mots-clés, identificateurs, opérateurs...

Les blancs et les commentaires sont éliminés

À quoi servent les espaces dans le code source ?

Analyseur syntaxique

Trouve les instructions, expressions, fonctions...

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

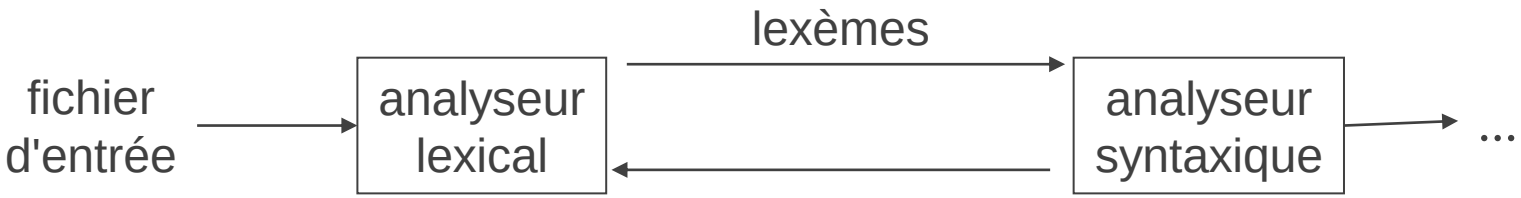
Analyseur lexical pour un compilateur

```
float perimeter= 3.1416 ;
if(min[n] <= max(1)){
   result='/';
   return ;
}
```

En langage C et en Java,

on est entre deux lexèmes si et seulement si on peut y insérer de l'espace blanc

Rôle d'un analyseur lexical



Un analyseur lexical

- parcourt le fichier d'entrée
- reconnait les lexèmes
- pour chaque lexème, lance une action

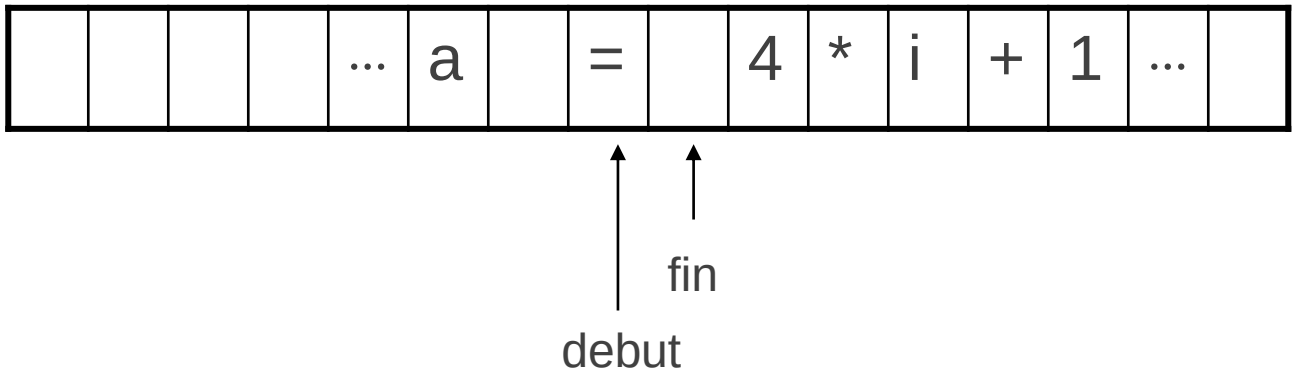
Transformer un flot de caractères en flot de lexèmes

Modulariser

Réduire la complexité de la conception et de l'implémentation

Augmenter la flexibilité, la portabilité, la maintenabilité

Deux méthodes de construction



Utiliser un générateur d'analyseurs lexicaux (Gcc contient Flex, l'implémentation de Lex la plus courante)

Écrire l'analyseur lexical directement

Université Gustave Eiffel

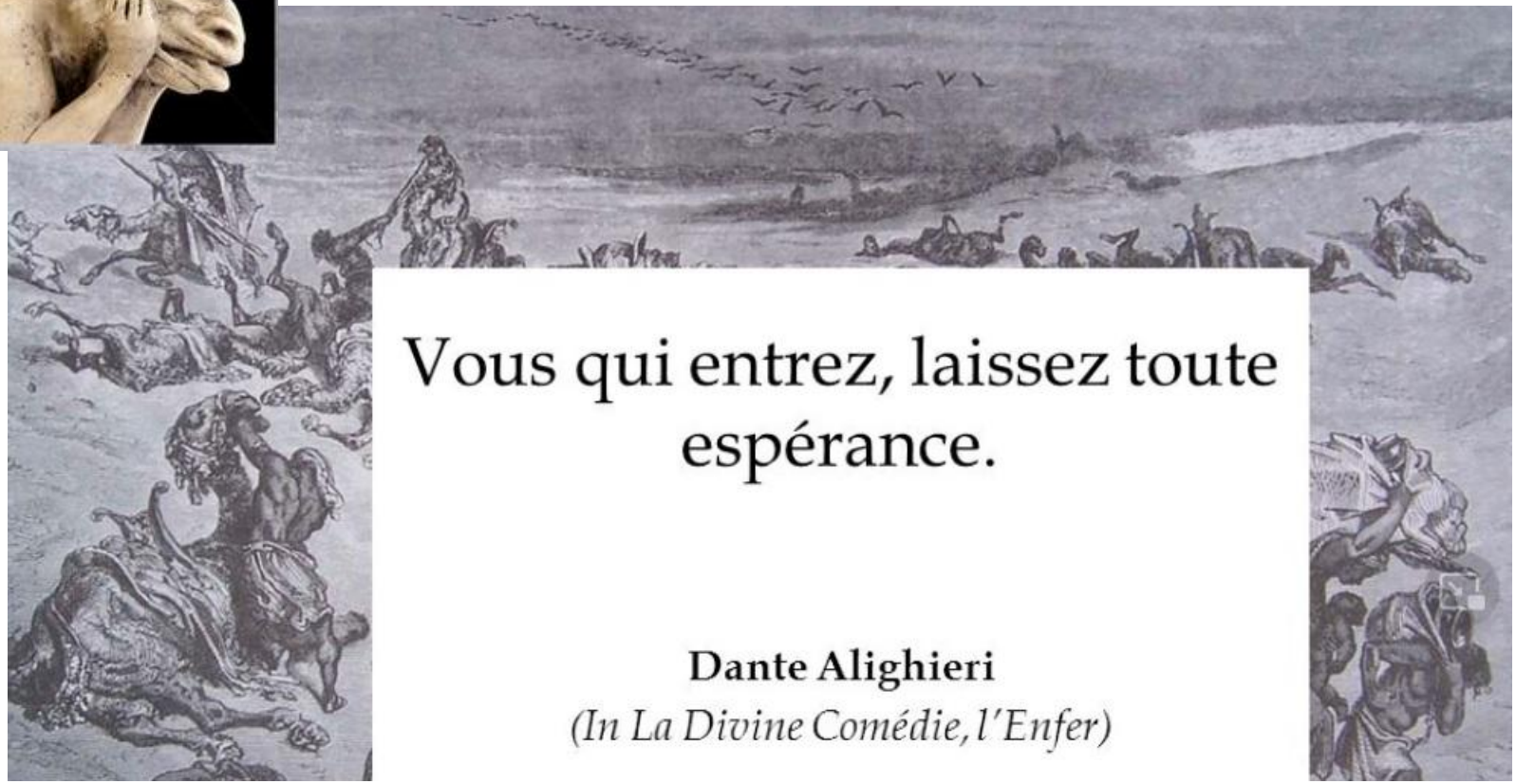
INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

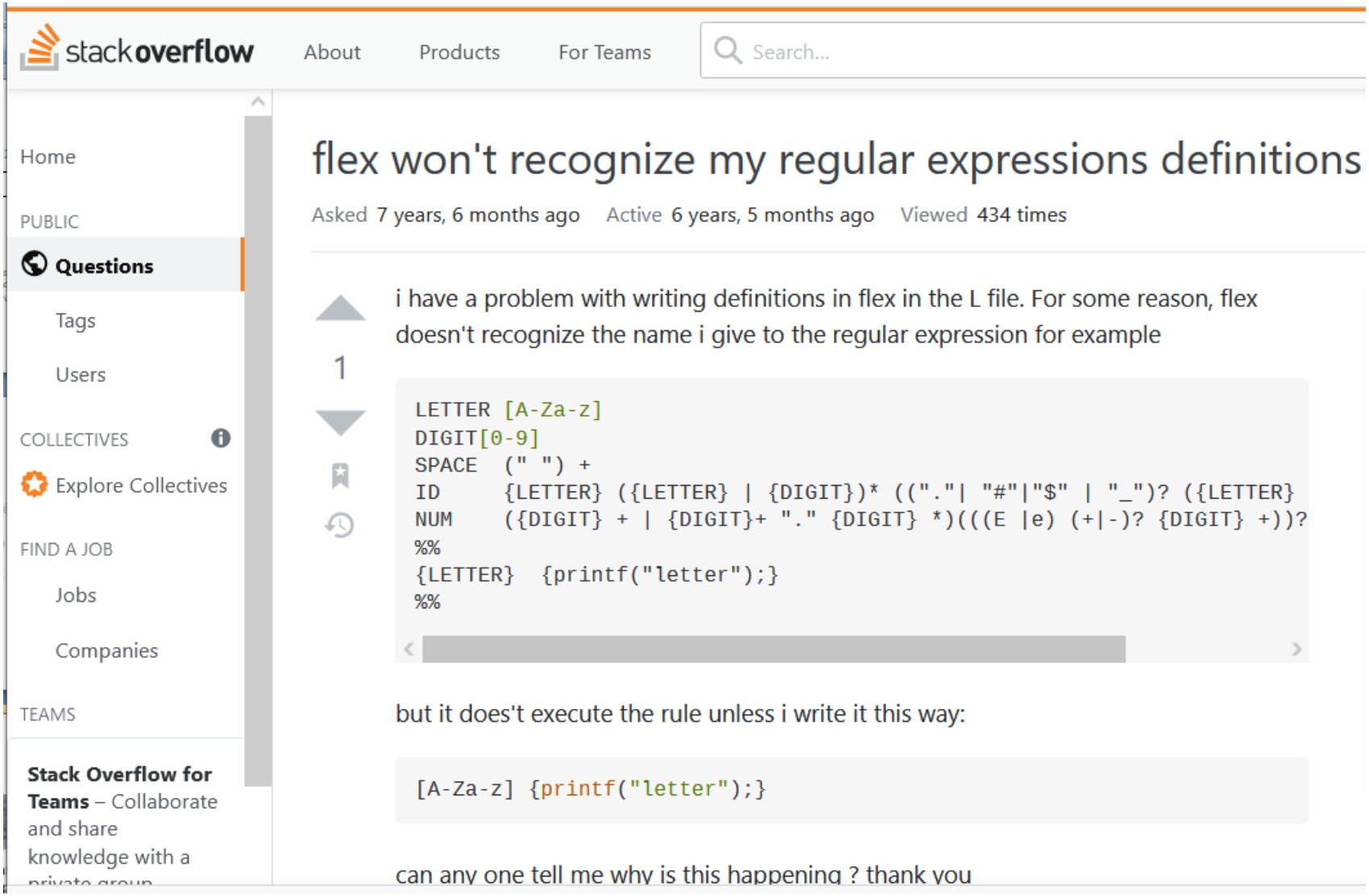
Flex n'est pas facile à utiliser



Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Flex fait ce qui est spécifié et il y a énormément d'aide en ligne



stackoverflow About Products For Teams Search...

Home

PUBLIC

Questions

Tags

Users

COLLECTIVES

Explore Collectives

FIND A JOB

Jobs

Companies

TEAMS

Stack Overflow for Teams – Collaborate and share knowledge with a private group

flex won't recognize my regular expressions definitions

Asked 7 years, 6 months ago Active 6 years, 5 months ago Viewed 434 times

1

i have a problem with writing definitions in flex in the L file. For some reason, flex doesn't recognize the name i give to the regular expression for example

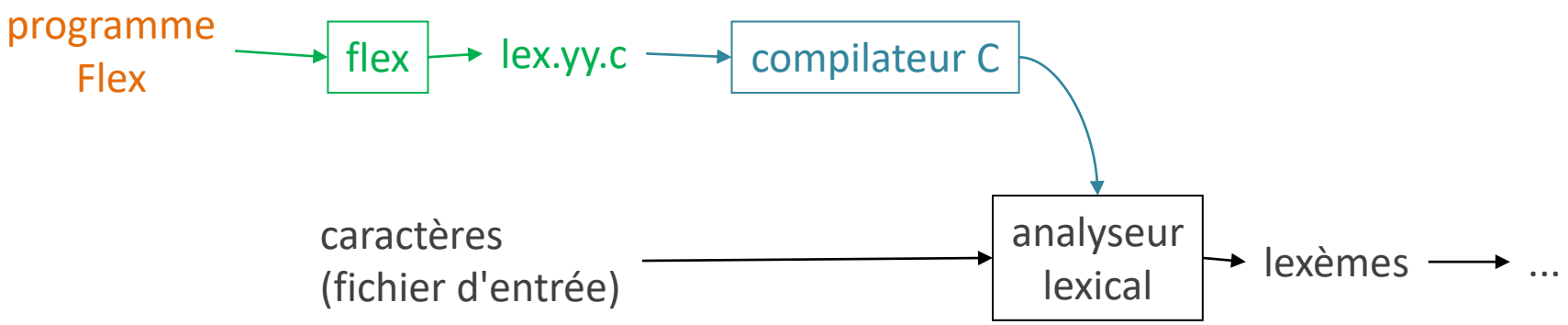
```
LETTER [A-Za-z]
DIGIT[0-9]
SPACE  (" ") +
ID     {LETTER} ({LETTER} | {DIGIT})* (("."| "#"|"$" | "_")? ({LETTER}
NUM    ({DIGIT} + | {DIGIT}+ "." {DIGIT} *)(((E |e) (+|-)? {DIGIT} +))?
%%
{LETTER}  {printf("letter");}
%%
```

but it does't execute the rule unless i write it this way:

```
[A-Za-z] {printf("letter");}
```

can any one tell me why is this happening ? thank you

Utilisation de Flex



4 étapes :

- créer avec un éditeur un programme Flex
  (expressions régulières)
- traiter ce programme par la commande `flex`
- compiler le programme C obtenu
- exécuter le programme exécutable obtenu

Il faut un makefile

Un programme Flex pour utilisation avec un analyseur syntaxique

```
[ \t\n]*          { /* pas d'action */ }
if                { return IF ; }
then              { return THEN ; }
else              { return ELSE ; }
{letter}({letter}|[0-9])*    { yylval = install_id() ;
                  return ID ; }
([0-9]+(\.[0-9]*)?|\.[0-9]+)((E|e)(\+|-)?[0-9]+)? {
                  yylval = install_num() ; return NUMBER ; }
"<"               { yylval = LT ; return RELOP ; }
"<="              { yylval = LE ; return RELOP ; }
```

L'analyseur lexical produit par Flex

trouve les lexèmes en utilisant les **expressions régulières**

pour chaque lexème reconnu, exécute l'**action (code en C)** correspondante

Utilisation de Flex avec un analyseur syntaxique

```
else            { return ELSE ; }
{letter}({letter}|[0-9])*    { yylval = install_id() ;
                return ID ; }
```

L'analyseur lexical produit par Flex

contient une fonction `yylex()` qui trouve les lexèmes et exécute les actions correspondantes

Si une action contient un `return`, il termine `yylex()`

L'analyseur syntaxique appelle `yylex()`

Pour chaque lexème reconnu, on écrit l'action correspondante pour qu'elle renvoie des informations à l'analyseur syntaxique par un `return`

Autres utilisations de Flex (sans analyseur syntaxique)

```
%{
int line_count=1;
%}
%%
.*   { line_count++; }
%%
```

On commencera par là en TP avant de savoir faire un analyseur syntaxique

L'analyseur lexical produit par Flex

pour chaque lexème reconnu, exécute un bout de code en C

La fonction **`yylex()`** se rappelle elle-même

(S'il y a un **`return`** dans une action, elle ne peut pas se rappeler parce que le **`return`** la termine avant)

Exercice

```
#include <stdio.h>
int yylex();
void yyerror(char *);
#line 74 "instr-comm.c" /* yacc.c:339  */
# ifndef YY_NULLPTR
#  if defined __cplusplus && 201103L <= __cplusplus
#   define YY_NULLPTR nullptr
#  else
#   define YY_NULLPTR 0
#  endif
# endif
/* Enabling verbose error messages.  */
#ifdef YYERROR_VERBOSE
# undef YYERROR_VERBOSE
# define YYERROR_VERBOSE 1
#else
# define YYERROR_VERBOSE 0
#endif
```

Compter dans un fichier en C les **`#define`**, **`#ifdef`**, **`#ifndef`**

Découpage du fichier d'entrée par l'analyseur lexical

```
hachage -- time : 1543098418.019712s
```

```
hachage[ ]*[-]*  {act1();}
[-]*[ ]*time     {act2();}
```

Pour aller plus loin avec Flex, il faut savoir les règles qu'il applique pour découper le fichier d'entrée en lexèmes

A priori, plusieurs découpages sont possibles

Découpage du fichier d'entrée par l'analyseur lexical (1/3)

```
position = initial + vitesse * 60
```

Si *pirate* est reconnu, *rat* n'est pas reconnu

Un programme Flex minimal

L'analyseur lexical produit par Flex

- commence à chercher les lexèmes au début du code source ;
- après chaque lexème reconnu, recommence à chercher les lexèmes **juste après**

Si aucun lexème n'est reconnu à partir d'un point du code source

l'analyseur affiche le premier caractère sur la sortie standard et recommence à chercher à partir du caractère suivant

Découpage du fichier d'entrée par l'analyseur lexical (2/3)

Si plusieurs lexèmes sont reconnus à partir d'un point du code source (conflit)

1. Deux lexèmes de longueurs différentes

C'est **le plus long** qui gagne

Exemple 1 : les identificateurs

[a-zA-Z_][a-zA-Z_0-9]*

Cela permet d'aller jusqu'à la fin d'un identificateur

Exemple 2 : **`a+++b`**

Découpage du fichier d'entrée par l'analyseur lexical (3/3)

Exemple : conflit entre
[a-zA-Z_] [a-zA-Z_0-9]*
et *while*

```
while
[a-zA-Z_][a-zA-Z_0-9]*
```

2. Deux lexèmes de même longueur

Ils commencent au même point et terminent au même point : s'il y a conflit, c'est qu'ils sont issus de deux règles différentes. C'est la règle qui **apparaît la première** dans la spécification Flex qui gagne

Mettre l'exception avant la règle

En dehors de ce cas, l'ordre des règles ne joue pas

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Expressions régulières en Flex

.	N'importe quel caractère (octet) sauf retour à la ligne
[xyz]	Un caractère parmi x, y et z
[abj-oZ]	Un caractère parmi a, b, Z et n'importe lequel entre j et o
[^A-Z]	N'importe quel caractère autre que ceux entre A et Z
r*	Zéro, une ou plusieurs fois l'expression r
r+	Un ou plusieurs r
r?	Zéro ou un r
r{2,5}	Entre deux et cinq r
r{2,}	Au moins deux r
r{4}	Quatre r

Source : la page man de la commande flex

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Expressions régulières en Flex

{nom} Comme l'expression définie comme « nom » dans les déclarations

"[xyz]\"glb" La chaine de caractères [xyz]"glb

(r) Un r. Les parenthèses imposent un ordre d'application des opérateurs

rs Un r suivi d'un s

r|s Un r ou un s

r/s Un r, mais uniquement s'il est suivi d'un s. La longueur prise en compte pour résoudre les confilts est celle de rs, mais la partie reconnue (yytext) est uniquement r

^r Un r, mais uniquement s'il est en début de ligne

r$ Un r, mais uniquement s'il est en fin de ligne

<<EOF>> Fin de fichier

Les expressions régulières ci-dessus sont groupées en fonction de leur priorité, la plus haute au sommet et la plus basse en bas.

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Exercices

```
sport[0]=foot;
sport[1]=repos;
sport[2]=spectacle1;
sport[3]=rollers;
sport[4]=fete1;
```

1. Copier le fichier d'entrée sur la sortie standard à l'identique
2. Copier du code en C en remplaçant les **`sport`** par des **`activite`**

Indication : l'exemple donné n'est qu'un exemple

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Programmes Flex

Un programme Flex est fait de trois parties :

déclarations

%%

règles de traduction

%%

fonctions auxiliaires en C

Les règles de traduction sont de la forme

p_1 **{** *action*$_1$ **}**

p_2 **{** *action*$_2$ **}**

...

p_n **{** *action*$_n$ **}**

où chaque p_i est une expression rationnelle et chaque action une suite d'instructions en C.

Un programme Flex pour utilisation avec un analyseur syntaxique (1/2)

```
%{
/* Partie en langage C : définitions de constantes,
déclarations de variables globales, commentaires... */
%}
letter [a-zA-Z]
%%
[ \t\n]*        { /* pas d'action */ }
if              { return IF ; }
then            { return THEN ; }
else            { return ELSE ; }
{letter}({letter}|[0-9])*    { yylval = install_id() ;
                return ID ; }
([0-9]+(\.[0-9]*)?|\.[0-9]+)((E|e)(\+|-)?[0-9]+)? {
                yylval = install_num() ; return NUMBER ; }
```

Un programme Flex pour utilisation avec un analyseur syntaxique (2/2)

```
"<"             { yylval = LT ; return RELOP ; }
"<="            { yylval = LE ; return RELOP ; }
%%
int install_id() {
/* fonction installant dans la table des symboles le
   lexème vers lequel pointe yytext et dont la longueur
   est yyleng. Renvoie l'indice de l'entrée dans la table
   */
}

int install_num() {
/* calcule la valeur du lexème et renvoie son indice dans
   une table */
}
```

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Variables définies par Flex

`yytext`

Chaine de caractères contenant la partie reconnue

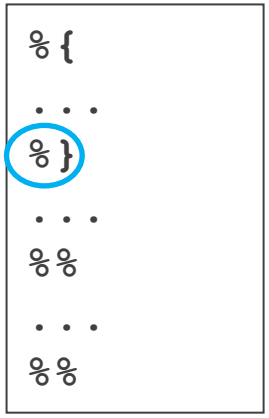
`yyleng`

Longueur de **`yytext`** (sans le 0 final)

`ECHO;`

Écrit **`yytext`**

Syntaxe de Flex et Lex

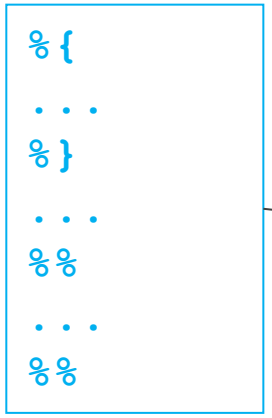


Flex est une implémentation de Lex (1975)

On utilise toujours l'extension .lex

Les **conventions d'écriture** des spécifications Flex sont contre-intuitives

- Plusieurs parties avec des conventions différentes : C, expressions régulières, déclarations
- Balises bizarres
- Espaces blancs significatifs



Espaces blancs

Les espaces blancs sont significatifs
Les balises doivent être en début de ligne
Dans les règles de traduction, l'espace blanc sert de délimiteur entre expression régulière et action

```
%%
[ \t\n]*        { /* pas d'action */ }
if              { return IF ; }
then            { return THEN ; }
```

```
%%
 [ \t\n]*       { /* pas d'action */ }
if              { return IF ; }
then            { return THEN ; }
```

Espaces blancs

```
%%
[ \t\n]*        { /* pas d'action */ }
if              { return IF ; }
then            { return THEN ; }
{letter}({letter}|[0-9])*    { yylval = install_id() ;
                return ID ; }
```

!

```
%%
 [ \t\n]*       { /* pas d'action */ }
if              { return IF ; }
then            { return THEN ; }
{letter} ({letter}|[0-9])*   { yylval = install_id() ;
                return ID ; }
```

!

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Programmes Flex

Les commentaires /* ... */ ne peuvent être insérés que dans une portion en C :

- dans la partie déclaration, seulement entre **%{** et **%}** ou dans des lignes commençant par un espace blanc ;
- dans la partie règles, seulement dans les actions ;
- dans la partie fonctions auxiliaires, n'importe où.

Dans les règles

p_i **{** $action_i$ **}**

les expressions rationnelles p_i ne peuvent pas contenir d'espaces blancs (ou alors dé-spécialisés).

Au début de la partie règles, si une ou plusieurs lignes sont entre **%{** et **%}** ou commencent par un espace blanc, elles sont interprétées comme du langage C et insérées dans lex.yy.c au début de la fonction qui reconnait les motifs (utilisable pour déclarer des variables locales et ajouter des commentaires).

`main()` dans Flex

```
%%
int main(){
  return yylex();
}

gcc lex.yy.o -Wall -lfl
```

On peut écrire un `main()` dans un programme Flex

Sinon, la **bibliothèque de Flex** (option -lfl, ou -ll sous OS-X) fournit un `main()` qui appelle seulement `yylex()`

L'option `-lfl` doit apparaitre à la fin et non au début de la ligne de commande

Options Flex

```
%{
/* Partie en langage C */
%}
letter  [a-zA-Z]

%option nounput
%option noinput
%option noyywrap

%%
[ \t\n]*        ;
if              { return IF ; }
then            { return THEN ; }
else            { return ELSE ; }
```

%option nounput

%option noinput

Suppriment des avertissements du compilateur sur des fonctions définies et non utilisées

%option noyywrap

Utile si on n'utilise pas la bibliothèque de Flex (-lfl)

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Expressions régulières en Flex (suite)

Séquences d'échappement

\X Si X est a, b, f, n, r, t ou v, le caractère \X en C ANSI. Si X est 0, le caractère de code ASCII 0. Sinon, un X déspécialisé

\123 Le caractère de valeur octale 123

\x2a Le caractère de valeur hexadécimale 2a

Pour Flex, un « retour à la ligne » est le '\n' du compilateur C utilisé pour compiler Flex. En Windows, on doit parfois filtrer soi-même les \r de l'entrée ou utiliser explicitement r/\r\n pour « r$ ».

Université Gustave Eiffel | INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Expressions régulières en Flex

À l'intérieur des crochets ([xyz])

À l'intérieur des crochets, tous les opérateurs d'expressions régulières perdent leur signification spéciale sauf « \ », « - », «] » (sauf juste après le crochet ouvrant), et, juste après le crochet ouvrant, « ^ », donc les caractères suivants ne sont **pas** des caractères spéciaux :

. * + ? { } " ( ) | / $

et l'espace non plus.

Les crochets peuvent également contenir des expressions de classes de caractères. Ce sont des expressions entourées par des délimiteurs [: et :] (qui doivent elles-mêmes apparaître entre le '[' et le ']' extérieurs ; d'autres éléments peuvent aussi être présents dans les mêmes crochets). Les expressions valides sont :

[:alnum:] [:alpha:] [:blank:] [:cntrl:] [:digit:] [:graph:] [:lower:] [:print:] [:punct:] [:space:] [:upper:] [:xdigit:]

Ces expressions désignent chacune un groupe de caractères équivalent à la fonction C standard correspondante isXXX.

```
#include <stdio.h>
int yylex();
void yyerror(char *);
#line 74 "instr-comm.c" /* yacc.c:339  */
# ifndef YY_NULLPTR
#  if defined __cplusplus && 201103L <= __cplusplus
#   define YY_NULLPTR nullptr
#  else
#   define YY_NULLPTR 0
#  endif
# endif
/* Enabling verbose error messages.  */
#ifdef YYERROR_VERBOSE
# undef YYERROR_VERBOSE
# define YYERROR_VERBOSE 1
#else
# define YYERROR_VERBOSE 0
#endif
```

Exercices

Extraire d'un fichier en C les **`#define`**, **`#ifdef`**, **`#ifndef`** et les copier dans un autre fichier

- avec la ligne entière
- avec les extensions sur une ou plusieurs autres lignes

Un peu de vocabulaire

Différence entre lettre et caractère

Caractères

- Lettres aàbcçdé...AÀBCÇDÉ...
- Chiffres 0123...
- Espaces blancs (*whitespace*)
- Autres % = " ' . * + ? { } " ( ) | / $...

Un peu de vocabulaire bilingue

Entre apostrophes (*in single quotes*) `'r'`

Entre guillemets (*in double quotes*) `"r"`

Entrecôte (*rib steak*)

Photo by Dcollard - Own work, CC BY-SA 3.0
https://commons.wikimedia.org/w/index.php?curid=21614585

Exercice plus difficile : chaines de caractères entre guillemets

\"[^"]*\"

Reconnait "bonjour" ""

Ne reconnait pas "bonjo "bon"jour"

"%s\n\"%s\"\n"

\"([^"\\]|\\.|\\\n)*\"

Reconnait "%s\n\"%s\"\n"

Ne reconnait pas "bonjo "bon"jour"