Théorie des langages — TP Compilation

Introduction

Analyse

- lexicale : reconnaissance des “mots”
- syntaxique : reconnaissance des “phrases”
- sémantique : reconnaissance du “sens” des mots et des phrases

Synthèse

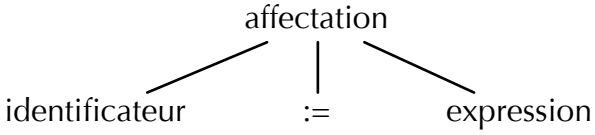
- production et optimisation de code

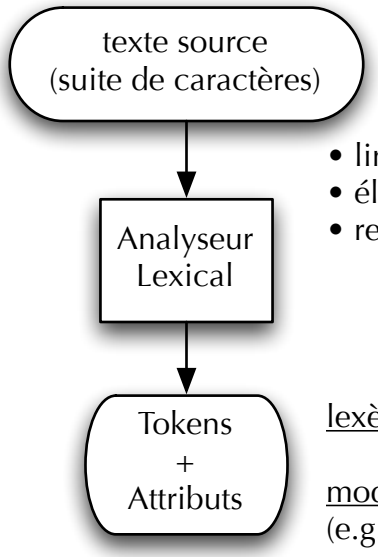
Exemple `x := y + 1;`

Analyse lexicale

```
( identificateur, x )
( affectation )
( identificateur, y )
( + )
( entier, 1 )
( ; )
```

Analyse syntaxique





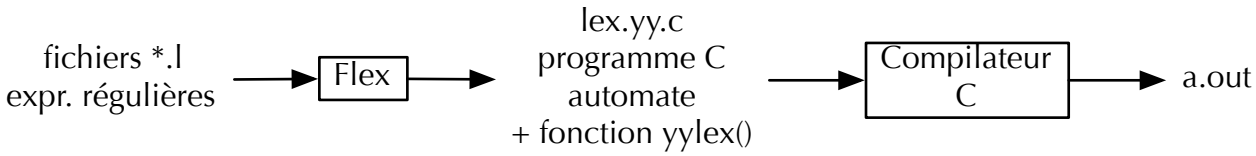
- lire les caractères du code source
- éliminer les espaces, fins de ligne, commentaires
- regrouper les caractères en
  - unités lexicales (UL)
  - catégories lexicales
  - tokens

lexème : valeur de l'unité lexicale

modèle : spécification de l'unité lexicale
(e.g. suite de lettres, chiffres, commençant par une lettre, etc.)

Spécification LEX

modèle d’UL { action à exécuter }



Exemple `-?[0-9]+ {printf(“entier\n”);}`

Expressions régulières

`.`	un caractère quelconque sauf `\n`
`^`	début de ligne (en début début d'expression régulière) — e.g. `^e`
`$`	fin de ligne (en fin d'expression régulière) — e.g. `e$`
`e*`	0, 1 ou plusieurs fois e
`e+`	au moins 1 fois e
`e?`	0 ou 1 fois e
e_1`|`e_2	e_1 ou e_2
`e{n}`	exactement n fois e
`e{n,}`	au moins n fois e
`e{n,p}`	de n à p fois e

Classes de caractères

`[abcd]`	un caractère parmi {a, b, c, d}
`[^abc]`	un caractère quelconque sauf a, b ou c
`[a-z]`	un caractère de l'intervalle a…z
`[a-zA-Z]`	un caractère de l'intervalle a…z ou A…Z
`\`	échappement pour les méta-caractères
`"…"`	les méta-caractères ne sont pas interprétés sauf `\` (anti-slash) — e.g. `"1+2*3"`

Exemple de spécification LEX

```
%%
-?[0-9]+  { printf("entier %s\n", yytext); }
.|\n      ;
%%
 /* fonction main() par défaut */
main() {
  yylex();
}
```

N.B.
Pour connaître la valeur des tokens : char * yytext
Pour connaître la longueur des tokens : int yylen

Définir des noms pour des expressions régulières

Exemple 1

```
chiffre [0-9]
%%
-?{chiffre}+       { /* action à effectuer */ }
```

Exemple 2

```
mot1 a|b
mot2 (a|b)
%%
{mot1}+   /* a|b+ */
{mot2}+   /* (a|b)+ */
```

Compter les mots

```
%{
int nb_mots = 0;
%}
%%
[^ \t\n]+ { nb_mots++; }
.|\n ;
%%
main() {
  yylex();
  printf("Nombre de mots : %d\n", nb_mots);
}
```

Exemple d'utilisation

```
a ba cd
1 2  3
```

Identificateurs et mots clés

```
lettre [a-zA-Z]
%%
begin|end|if|while { printf("mot clé\n"); }     /* (1) */
{lettre}+ { printf("identificateur\n"); }       /* (2) */
.|\n ;                                          /* (3) */
%%
```

Exemple d'utilisation

```
begin          // (1) mot clé
begintoto      // (2) identificateur
toto           // (2) identificateur
toto1          // (2) identificateur + (3)
```

Notation équivalente

```
begin |
end |
if |
while { printf("mot clé|n"); }
```

Application

Ecrire un interpréteur qui reconnaît des indentificateurs, des constantes entières, constantes réelles (`4.5`,`.5`,`.5`), des mots reservés, des symboles spéciaux (`[.,+*/()\-<>]|<=|>=|:=|=`)

Exemple d'utilisation

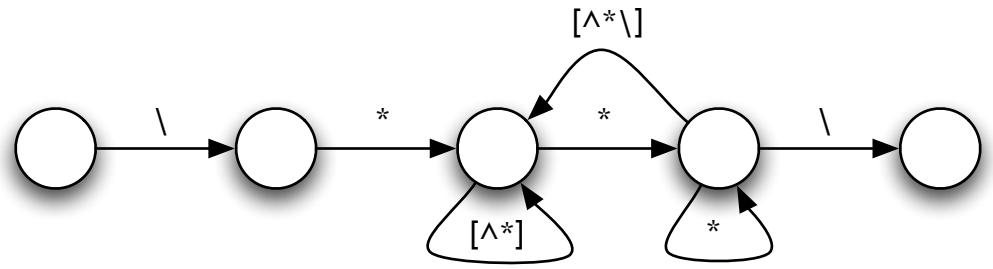
```
prog x + 1 47 + ; if @

( mot reservé, prog )
( identificateur, x )
( symbole spécial, + )
( constante entière, 1 )
( constante entière, 47 )
…
```

Les commentaires /* … */

Expression régulière

```
"/*"([^*]|\*+[^*/])*\*+\/
```

Automate



Conditions de déclenchement des règles

```
<Nom_Cond>[0-9]+ {…}
```

activer une condition

```
action(code C)
BEGIN(Nom_Cond);
```

désactiver une condition

```
BEGIN(INITIAL);

%x Nom_Cond // exclusif
%s Nom_Cond // inclusif
```

Exemple

```
%x MAGIQUE
%s

%%
[0-9]+ {printf("entier\n");}
magic {BEGIN(MAGIQUE);}
<MAGIQUE>[0-9] {printf("chiffre\n");}
<MAGIQUE>normal {BEGIN(INITIAL);}
<*> . |\n ;
```

Exemple d'utilisation

```
12                  entier
magic               begin MAGIQUE
12b                 chiffre
                    chiffre
normal              begin INITIAL
12b                 entier
```

Table des symboles

Structure de données qui contient un enregistrement pour chaque identifiant du programme

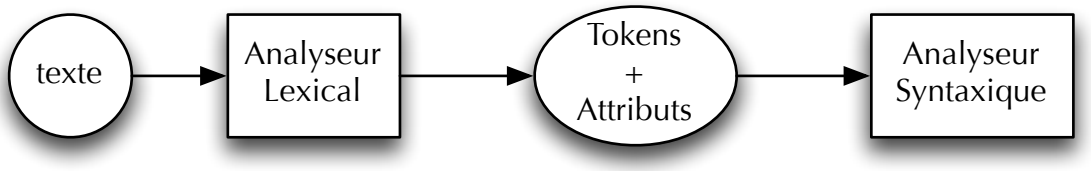
Algorithme
Quand on reconnaît un identifiant, on l'ajoute à la table des symboles s'il n'y est pas déjà.

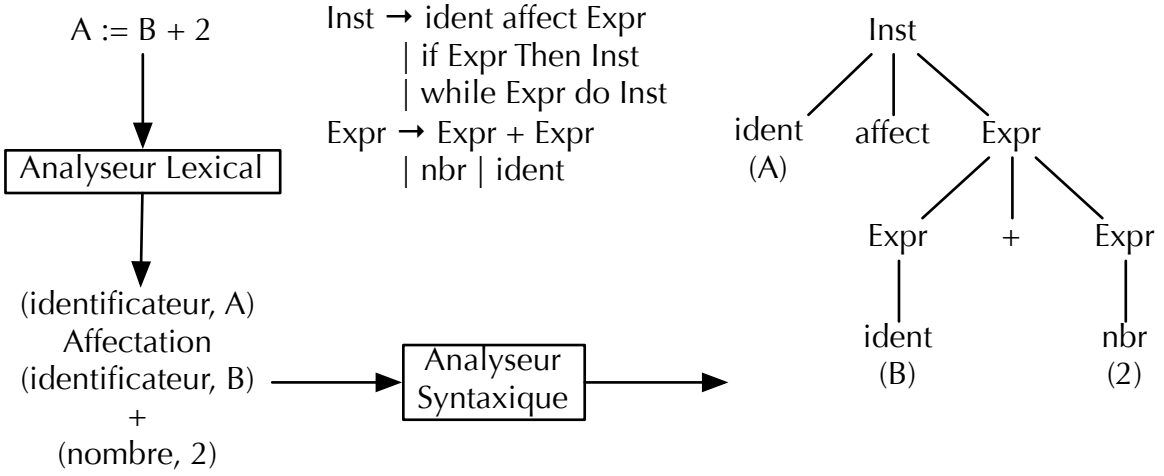
Exemple d'entrée
`int x = 2;` *ou* `x := y + x;`

Table des symboles

	nom	**type**	**val**
0	x	INT	2
1	y	…	…

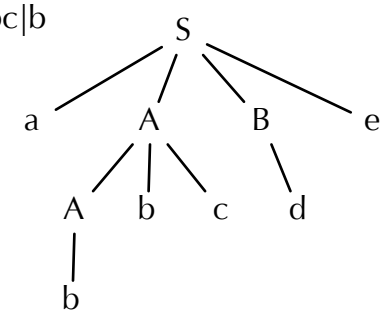
```
%{
 /* declaration de la table des symboles */
 /* declaration des fonctions */
%}
%x COMMENT1 COMMENT2
%%
"/*" {BEGIN(COMMENT1);}
<COMMENT1>"*/" {BEGIN{INITIAL);}
<COMMENT1>. /* ne rien faire */
<*>\n {num_ligne++;}
"{" {BEGIN(COMMENT2);}
<COMMENT2>"}" {BEGIN(INITIAL);}
<COMMENT2>. /* ne rien faire */
%%
 /* code des fonctions */
```





w = abbcde

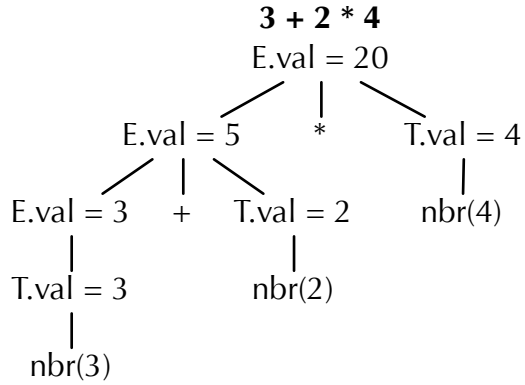
$$\begin{cases} S \rightarrow aABe \\ A \rightarrow Abc|b \\ B \rightarrow d \end{cases}$$



pile	phrase	action
	abbcde	décaler
a	bbcde	décaler
ab	bcde	réduire (A -> b)
aA	bcde	décaler
aAb	cde	pas réduire (A->b)
aAA	cde	décaler
aAbc	de	réduire (A->Abc)
aA	de	décaler
aAd	e	réduire (B->d)
aAB	e	décaler
aABe	Ø	réduire (S->aABe)
S	Ø	SUCCES

Evaluation d'expression arithmétique

```
E → E + T | E * T | T
T → nbr | ident
```



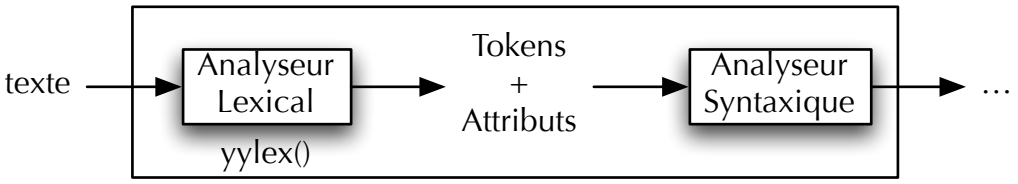
Attributs val pour E et T

E.val est la valeur de l'expression E
T.val est la valeur de l'expression T

Attributs vallex : valeur lexicale

nbr.vallex = valeur de nombre
ident.vallex = entrée dans la table des symboles

E → E_1 + T { E.val ← E_1.val + T.val }
E → E_1 * T { E.val ← E_1.val * T.val }
E → T { E.val ← T.val }
T → nbr { T.val ← nbr.vallex }
T → ident { T.val ← rechercher_valeur(ident.vallex) }



fichier.lex

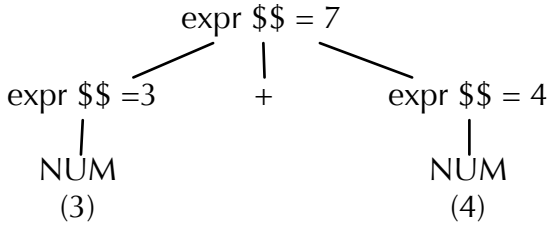
```
[0-9]+    { yylval = atoi(yytext); return(NUM); } // valeur lexicale et token
[+*]      { return (yytext[0]); } // les caractères isolés sont acceptés

 /* Partie 1 */

%{
  #define YYSTYPE double // type de yylval
%}
%token NUM
%start S

%%

 /* Partie 2 */

expr : expr '+' expr { $$ = $1 + $3; } // E → E + E { E.val ← E1.val + E2.val }
 | NUM { $$ = $1; } // E → nbr { E.val ← nbr.vallex }
 ;
%%

 /* Partie 3 */

#include "lex.yy.c"
int yyerror(char *s) {
          printf("%s\n", s);
          return 0;
}
```

Application 3 + 4

LEX

NUM yylval = 3
'+'
NUM yylval = 4

YACC



Définition des priorités (partie 1)

```
%left '+'
%right
%nonassoc
```

Exemple

```
%nonassoc MOINS_UNAIRE      // moins prioritaire
%left '+' '-'
%left '*'                   // plus prioritaire
```

La calculatrice doit accepter :
- des déclarations de variables (nom = expression)
- des expressions contenant des variables