TP Compilation
Master I IV

Dr FERRAHI

L’analyse lexicale

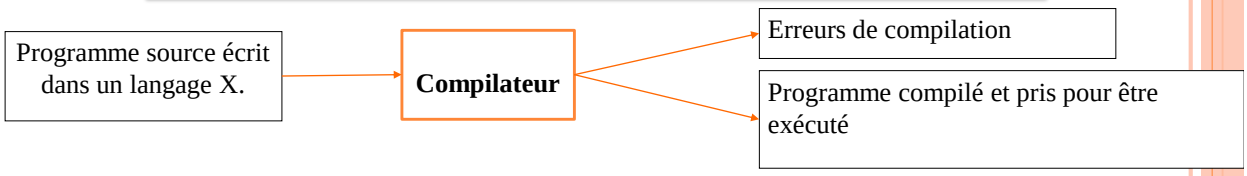
COMPILATION

- On a l'habitude de faire la compilation des langage naturel qu'on utilise dans notre vie courante.

  Je suis un étudiant

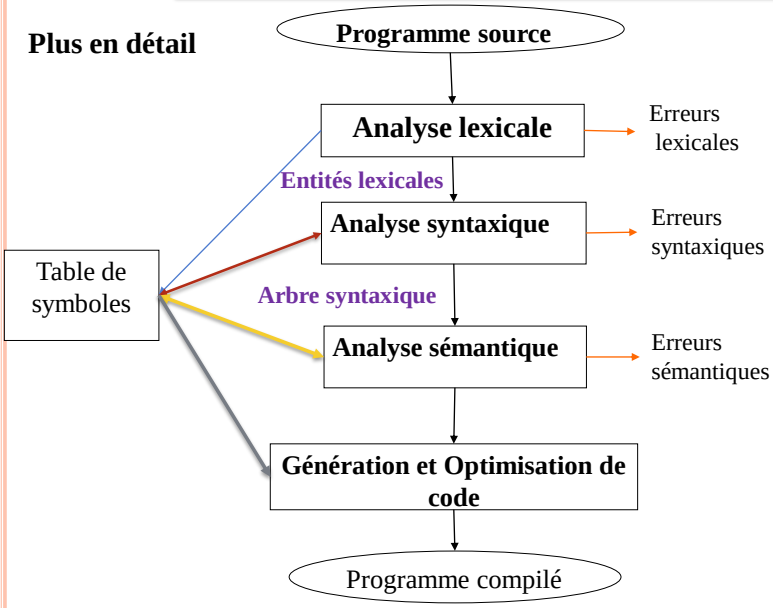
- **Analyse lexicale :** la phrase est lexicalement correcte (tous les mots appartient à la langue française).
- **Analyse syntaxique :** la phrase est syntaxiquement correcte (il existe une règle de grammaire qui accepte la succession Sujet + Verbe + Complément).
- **un suis Je étudiant (** lexicalement correcte mais syntaxiquement fausse)
- **Analyse sémantique :** la phrase est sémantiquement correcte (puisque elle a un sens)
- Je mange une table lexicalement et syntaxiquement correcte, mais sémantiquement fausse.
- **Ses tâches principales sont :**
  - La lecture des caractères d'entrée et de la production des entités lexicales en sortie.
  - L'élimination de caractères superflus : les commentaires, les tabulations, fin de lignes, …
  - Gérer les numéros de ligne dans le programme source afin de récupérer la ligne dans laquelle une erreur lexicale est apparue.

Introduction à la compilation



Introduction à la compilation

Plus en détail



Exemple

Soit le programme C suivant :

```
Main ()
{
    Int x, y  ;
    If x==2) y=1;
    X=3,14;
}
```

→ **Le pgm est correcte lexicalement**,
→ **Le pgm est syntaxiquement incorrecte** : manque de la (dans l'instruction if
→ **Le pgm sémantiquement incorrecte :** x = 3,14

Structure d'un fichier FLEX

- L'analyseur lexical constitue la première étape d'un compilateur.
- **Ses tâches principales sont :**
  - La lecture des caractères d'entrée et de la production des entités lexicales en sortie.
  - L'élimination de caractères superflus : les commentaires, les tabulations, fin de lignes, …
  - Gérer les numéros de ligne dans le programme source afin de récupérer la ligne dans laquelle une erreur lexicale est apparue.
- **Exemple:**
- **Code source avant la compilation** **IF (x==2) y=z; /* affectation*/**
- **Après l'analyse lexicale:** **MC_IF (IDF EGAL CONST) IDF AFF IDF PVG**

L'ANALYSE LEXICALE

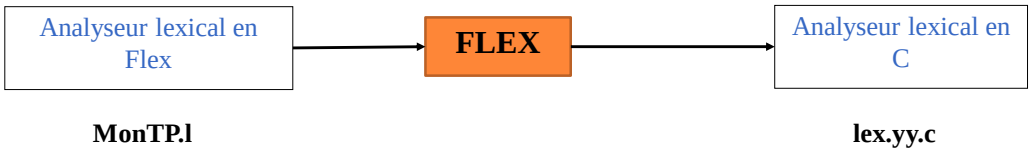
- L'analyseur lexical est basé sur l'algorithme simple suivant :

```
Lire (ChaineEntrée) ;
Switch (ChaineEntrée)
Case (ExpReg1) : coder(« Entité1 ») ;
Case (ExpReg2) : coder (« Entité2 ») ;
….
Case (ExpRegN) : coder (« EntitéN ») ;
Default : écrire (« Erreur lexicale ») ;
```

- L'implémentation de cet algorithme nécessitera l'écriture de centaines/milliers de lignes de code.
- **D'où l'utilisation de l'outil FLEX**

L'analyse lexicale « FLEX »

- **FLEX** est un générateur d'analyseur lexicale.
- Il traduit notre analyseur lexical écrit dans un langage simple (FLEX) vers la langage C



- **But :**
- Reconnaître les entités lexicales du langage
- Exécuter des actions a chaque entité rencontré

Structure d'un fichier FLEX

- Un document FLEX comporte trois partie importante séparées par le symbole %%

%{
Définitions en langage C
%}
Les définitions des expressions régulière

Partie 1

%%

Les règles de traduction { Action C}

Partie 2

%%

Code additionnel :
Redéfinitions des fonctions prédéfinies

Partie 3

Structure d'un fichier FLEX

La partie 1 : Les expressions régulières

- **But :** Donner les ER des différentes entités lexicales sous la forme suivante :

  <identificateur_de_l'entité> <expression_régulière>

- **identificateur_de_l'entité :**
  - doit commencer par une lettre
  - Il ne comporte que des caractère alphanumériques et des _ ou bien des -
  - L'expression régulière doit être valide
- Les caractères suivants sont réservés : ils doivent être précédés par \ ou notés entre " " si l'on veut leur valeur littérale

Structure d'un fichier FLEX

La partie 1 : Les expressions régulières

caractères	significations	exemples
+	Répéter 1 ou plusieurs fois	x+ x...x
*	Répéter 0 ou plusieurs fois	t* vide ou t...t
?	0 ou 1 fois	a? vide ou a
|	union	ab|bc ab ou bc
( )	factorisation	(a|b)c ac ou bc
"	valeur littérale des caractères	"+?"+ +?...+?
\	valeur littérale de caractère qui le précède même entre ""	\++ +...+ "\"+ erreur car la première " ne se ferme pas.
.	N'importe quel caractère sauf la fin de ligne (\n).	.|\n n'importe quel caractère

Structure d'un fichier FLEX

La partie 1 : Les expressions régulières

caractères	significations	exemples	
[...]	ensemble de caractères.	[aeiou]	la voyale :a ou e ou i ou o ou u.
-	Utilisé dans [] signifie un intervalle d'ensemble	[0-9]	0|1|2|3|4|5|6|7|8|9
^	Utilisé dans [] signifie le complément d'ensemble	[^0-9]	tout caractère sauf chiffre

Attention :

[...] = ensemble de caractères, pas d'expressions régulières, ce qui veut dire:

[(0|1)+] un des caractères (,), 0, 1, |, +, pas une suite de 0 ou 1.

Mais les caractères ^, \, - restent des caractères spéciaux.

[^\]] tout caractère sauf]

Structure d'un fichier FLEX

- **Donner une expression régulière qui définit les entités suivantes :**
- Une suite de caractères alphanumériques qui commence par un caractère alphabétique.

  [a-zA-Z]([a-zA-Z]|[0-9])*
- Les constantes numérique signées par le (-/+).

  [+-]?([1-9][0-9]*|0) ou bien ([+-]?([1-9][0-9]*)|0
- N'importe quel caractère (la fin de ligne (\n) est inclue).

  .|\n
- Tous les caractère à part les espace, les tabulations et les fins de lignes.

  [^ \n\t]

Structure d'un fichier FLEX

Partie 2 : Les règles de traduction

- **But :** présenter l'ensemble des actions associées à chaque entité lexicale sous la forme suivante :

  <pattern> <action>

- **Un pattern :** une expression régulière décrivant une entité lexicale
- **Une action :** c'est un code C qui sera exécuté à chaque fois qu'une entité lexicale apparaît

Exemple :

{entier } {printf (‘’ L'entité reconnu est un entier’’) ; }

Structure d'un fichier FLEX

Partie 2 : Les règles de traduction

- **Exemple 1:** Un analyseur lexical pour le langage :
- X=5 ;

```
%{
      int nb_ligne=1;
%}
lettre [a-zA-Z]
chiffre [0-9]
IDF {lettre}({lettre}|{chiffre})*
cst {chiffre}+
%%
{IDF}  printf ("IDF \n");
{cst}   printf("cst");
"="     printf ("aff");
";" printf("pvg");
[ \t]
\n   {nb_ligne++; }
. printf("erreur lexicale à la ligne %d
\n",nb_ligne) ;
%%
Int main ()
{
   yylex () ;
 return 0
}
```

Structure d'un fichier FLEX

Partie 2 : Les règles de traduction

- **Exemple 1:** Un analyseur lexical pour le langage :

  X=5 ;

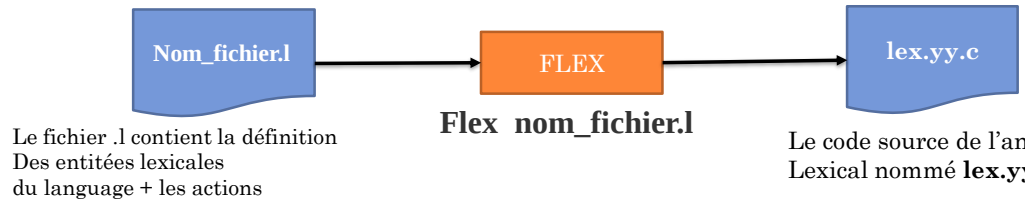
  Y=15,5;

```
%{
      int nb_ligne=1;
%}
lettre [a-zA-Z]
chiffre [0-9]
IDF {lettre}({lettre}|{chiffre})*
cst {chiffre}+
Réel {chiffre}+ ‘’.” {chiffre}+
%%
{IDF}  printf (‘’%s”, yytext) ;
{réel} { printf (‘’%s”, yytext) ;}
{cst}   printf (‘’%s”, yytext) ;
= printf (‘’aff”) ;
; printf ( ‘’pvg‘’);
[ \t]
\n   {nb_ligne++; }
.  printf("erreur lexicale à la ligne %d \n",
nb_ligne) ;
%%
Int main ()
{
   yylex () ;
  return 0 ;
}
```

Commandes de compilation

- Pour compiler le programme :
- **flex MonTP.l**
- **cc lex.yy.c -o MonTP -lfl (Linux)**
- **gcc lex.yy.c -o MonTP -lfl (Windows)**
- Pour exécuter le programme :
- **.\MonTP**
- Pour arrêter le programme :
- **Ctrl + c**

Commandes de compilation



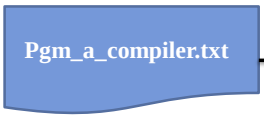
Le fichier .l contient la définition
Des entitées lexicales
du language + les actions

Flex nom_fichier.l

Le code source de l'analyseur
Lexical nommé **lex.yy.c**

gcc lex.yy.c -o Executable.exe -lfl

L'executable de l'analyseur
lexical nommé
Executable.exe



Programme compilé
Erreurs lexicales

Code source à compiler

.\Executable.exe <Pgm_a_compiler,txt

Macro-action de FLEX

Fonctions	yylex	Permet de lancer l'analyseur lexical
	yywrap	Elle est appelée par le Lexer quand il rencontre la fin du fichier. Elle doit soit obtenir un nouveau flux d'entrée et retourner simplement la valeur 0 soit renvoyer 1 signifiant que la totalité des flux a été consommée et que le lexer a fini sa tâche.
	yyterminate	Permet de provoquer la fin d'exécution du lexer
	ECHO	Affiche l'unité lexicale reconnue
Variables	yytext	Récupère le texte formant le lexème reconnu
	yyleng	Détermine la langueur du texte contennue dans yytext
	yylval	Est une variable globale utilisé par FLEX pour stocker la valeur correspondante au Token reconnu
	yylineno	Est le numéro de la ligne courante
	yyin	Fichier d'entrée
	yyout	Fichier de sortie

Exemple

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int nb_ligne=1;
%}
lettre [a-zA-Z]
chiffre [0-9]
IDF {lettre}({lettre}|{chiffre})*
cst {chiffre}+
%%
{IDF}                   printf(" IDF reconnu %s
\n",yytext);
{cst}                   printf(" CST reconnu\n");
"="                     printf(" = reconnu\n");
";"                     printf(" ; reconnu\n");
[ \t]
\n nb_ligne++;
. printf("erreur lexicale à la ligne %d \n",nb_ligne) ;
%%
```

```
int  main( )
{
 yyin = fopen( "programme.txt", "r" );
 if (yyin==NULL) printf("ERROR  \n");
   else yylex();
}
```