



Université
de Lomé



Ecole
Polytechnique de
Lomé

INF 1529 : COMPILATION

Année : 2024-2025

Titre : Réalisation d'un Analyseur Lexical

Etudiants:

- GANDONOU Koffi Patrick-léon
- BITORI Essoham

Chargé du Cours: Mr ANAKPA Manawa

I. Introduction

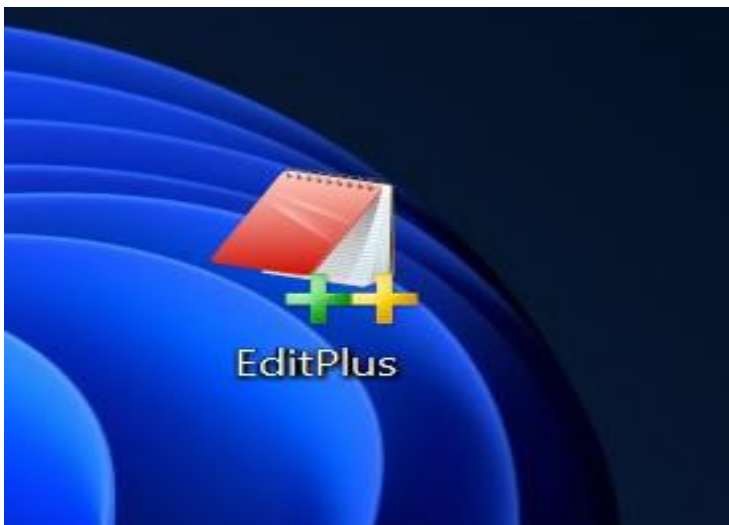
Dans le cadre de ce travail pratique, nous explorons l'utilisation des expressions régulières pour l'analyse lexicale. L'objectif principal est de concevoir et de tester un programme capable de reconnaître et de classifier différents types de chaînes de caractères. Ce programme utilise des règles lexicales définies pour identifier des éléments tels que les entiers, les identificateurs, les nombres réels ou encore des mots respectant un motif spécifique.

Ce projet met en évidence l'importance des expressions régulières en programmation, notamment dans le développement de compilateurs, d'interpréteurs ou d'autres outils d'analyse de texte. À travers les tests effectués, nous démontrerons la pertinence de cette méthode pour résoudre des problèmes complexes liés au traitement des chaînes de caractères

- ♦ Objectif : Le but du TP est de réaliser un analyseur lexical en utilisant l'outil LEX pour manipuler des expressions régulières.
- ♦ Résumé des étapes : Installer les outils nécessaires, configurer l'environnement, écrire le code et analyser différentes expressions.

Installation de l'Environnement

- ♦ **Étape 1 : Installation de l'éditeur EditPlus**
 - Description : Télécharger et installer EditPlus Text Editor.
Voici une **capture d'écran** après installation



- ♦ **Étape 2 : Configuration de l'environnement (ici nous avons choisi un environnement Linux)**

- Commandes utilisées :

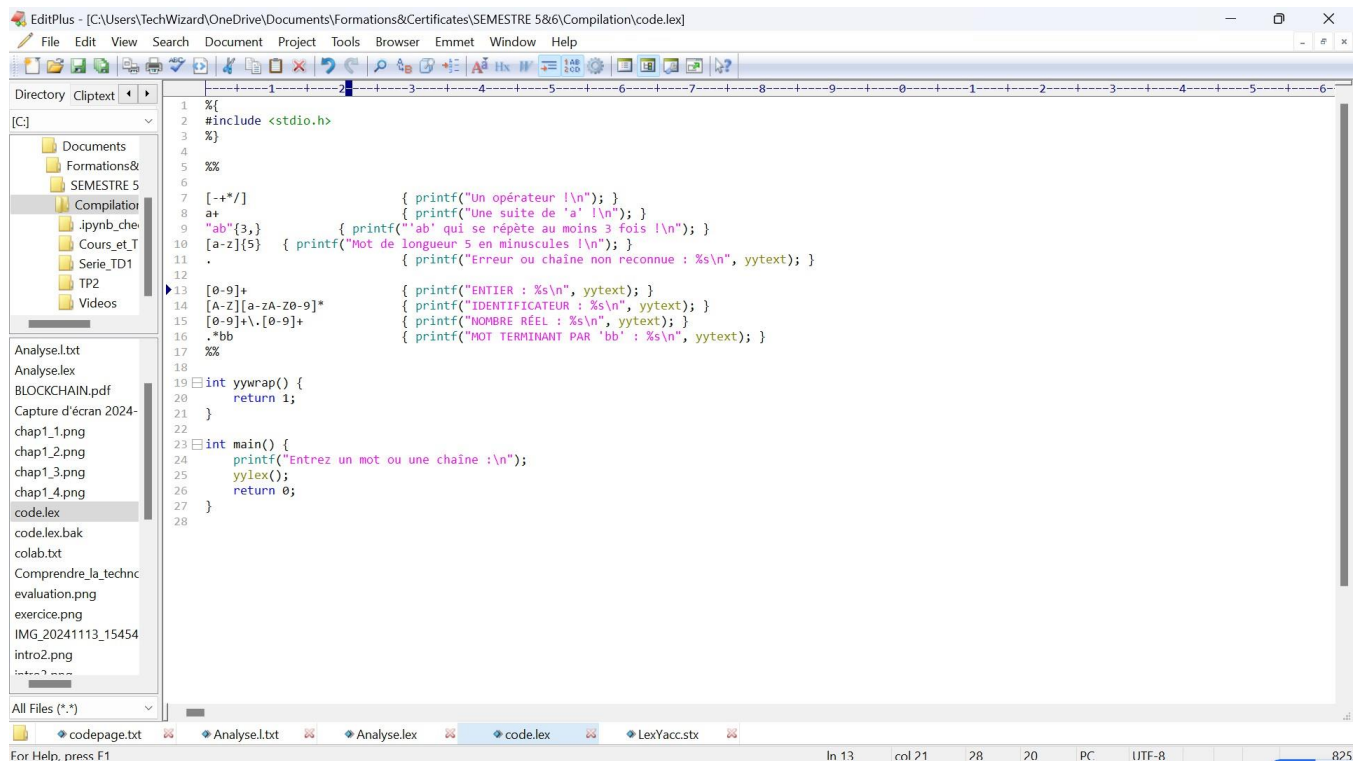
```
sudo apt update
```

```
sudo apt install flex gcc (Pour l'installation de flex et le compilateur gcc)
```

II- DEVELOPPEMENT DU CODE

- ♦ **Étape 3 : Écriture du code**

Description : Le code est écrit dans EditPlus. Et nous y voyons le code analyseur fourni dans le TP et les ajouts des Expressions Régulières demandées:



```
1  %{
2  #include <stdio.h>
3  %}
4
5  %%
6
7  [-+*/]          { printf("Un opérateur \n"); }
8  a+              { printf("Une suite de 'a' \n"); }
9  "ab"{3,}        { printf("'ab' qui se répète au moins 3 fois \n"); }
10 [a-z]{5}         { printf("Mot de longueur 5 en minuscules \n"); }
11 .               { printf("Erreur ou chaîne non reconnue : %s\n", yytext); }
12
13 [0-9]+           { printf("ENTIER : %s\n", yytext); }
14 [A-Z][a-zA-Z0-9]* { printf("IDENTIFICATEUR : %s\n", yytext); }
15 [0-9]+\.[0-9]+   { printf("NOMBRE RÉEL : %s\n", yytext); }
16 .*bb            { printf("MOT TERMINANT PAR 'bb' : %s\n", yytext); }
17 %%
18
19 int yywrap() {
20     return 1;
21 }
22
23 int main() {
24     printf("Entrez un mot ou une chaîne : \n");
25     yywrap();
26     return 0;
27 }
28
```

- ♦ **Étape 4 : Exécution du code (L'Analyseur)**

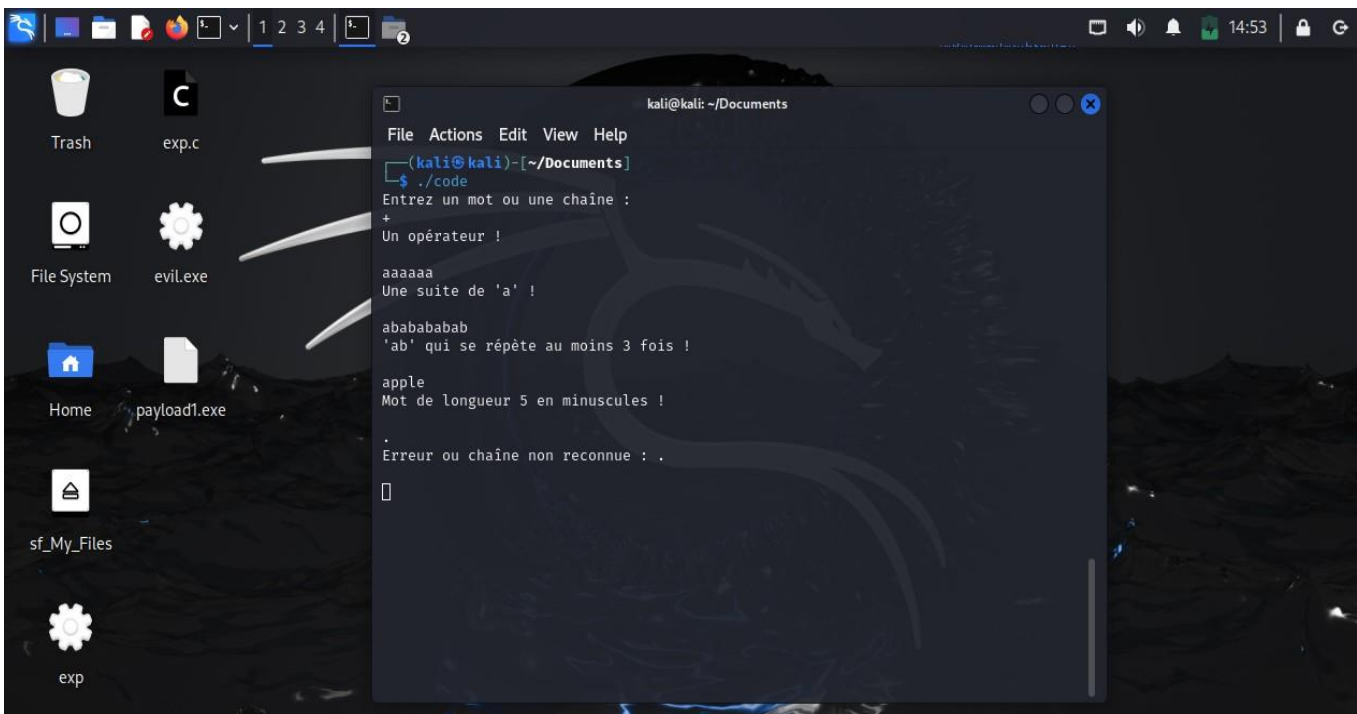
Code de l'analyseur

```
%{
#include <stdio.h>
%}
%%
[~+*/] {printf("un operateur !\n");}
a+ {printf(" une suite de a !\n");}
"ab"{3,} {printf("ab qui se répète au moins 3 fois!\n");}
[a-z]{5} {printf("Mot de longueur 5 en minuscules! \n");}
.+ {printf("Erreur !\n");}
%%
int yywrap()
{return 1;}
main()
{
printf("Entrez un mot:\n");
yylex();
}
}
```

- Description : Nous allons Compiler et exécuter le code avec les commandes suivantes :

```
flex code.lex
gcc lex.yy.c -o code
./analyseur
```

Voici ce que l'exécution du code analyseur dans le terminal, tout marche ! (Figure ci-contre)



1. Interprétation de l'Exécution du code analyseur

- ♦ **Opérateurs :**

- **Entrée :** +

- **Sortie :** Un opérateur !

*(Les opérateurs mathématiques comme +, -, *, et / sont identifiés.)*

- ♦ **Suite de 'a' :**

- **Entrée :** aaaaaa

- **Sortie :** Une suite de 'a' !

(Les séquences composées uniquement de lettres 'a' sont détectées.)

- ♦ **Mot de longueur 5 en minuscules :**

- **Entrée :** apple

- **Sortie :** Mot de longueur 5 en minuscules !

(Le programme identifie les mots de longueur 5 contenant uniquement des lettres minuscules.)

- ♦ **'ab' se répétant au moins trois fois :**

- **Entrée :** abababababab

- **Sortie :** 'ab' qui se répète au moins 3 fois !

(Les séquences avec trois occurrences ou plus de 'ab' sont reconnues.)

- ♦ **Chaînes non reconnues :**

- **Entrée :** .

- **Sortie :** `Erreur ou

chaîne non reconnue : .

(Les chaînes qui ne correspondent à aucune règle sont signalées comme non reconnues.)

1. Analyse des Résultats des Expressions Régulières Ajoutées (Résolution)

Voici les questions que nous allons répondre ici:

4. Ajouter des expressions régulières pour analyser
 - Les entiers
 - Les identificateurs qui commencent par une lettre majuscules
 - Les nombre réels
 - Les mots qui se terminent par deux b.

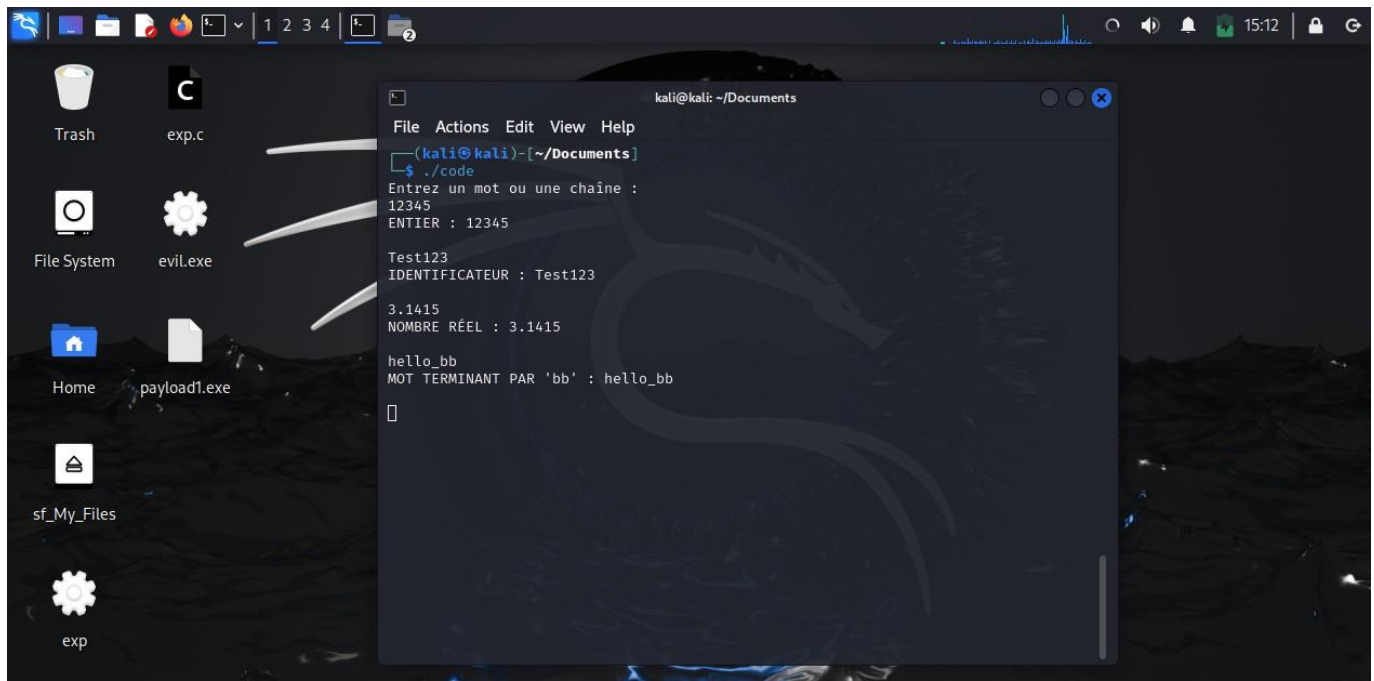
Voici le bout de code obtenu que nous allons ensuite ajouter au **code analyseur** du TP:

```
[0-9]+           printf("ENTIER : %s\n", yytext);
[A-Z][a-zA-Z0-9]* printf("IDENTIFICATEUR : %s\n", yytext);
[0-9]+\.[0-9]+   printf("NOMBRE RÉEL : %s\n", yytext);
.*bb$           printf("MOT TERMINANT PAR 'bb' : %s\n", yytext);
.               printf("CARACTÈRE NON RECONNU : %s\n", yytext);
```

III- Tests et résultats :

Pour tester l'analyseur lexical (pour les nouvelles expressions régulières), nous avons saisi différentes entrées correspondant aux expressions définies dans le fichier LEX. Voici un résumé des tests effectués avec les résultats obtenus .

- ♦ Voici dans le **terminal** le résultat obtenu après exécution de notre nouveau **code Analyseur**



- **Interprétation des sorties:**

1. **Entiers :**

- **Entrée :** 12345
- **Sortie :** ENTIER : 12345

(Le programme identifie correctement les entiers composés uniquement de chiffres.)

2. **Identificateurs :**

- **Entrée :** Test123
- **Sortie :** IDENTIFICATEUR : Test123

(Les identificateurs commencent par une lettre majuscule et peuvent contenir des lettres ou des chiffres.)

3. **Nombres réels :**

- **Entrée :** 3.1415
- **Sortie :** NOMBRE RÉEL : 3.1415

(Le programme reconnaît les nombres réels au format entier.partie_décimale .)

4. **Mots terminant par 'bb' :**

- **Entrée :** hello_bb
- **Sortie :** MOT TERMINANT PAR 'bb' : helloabb

(Tous les mots se terminant par 'bb' sont correctement identifiés.)

5. Conclusion

L'analyseur lexical développé avec LEX a correctement identifié et classifié les entrées selon les règles définies dans le fichier source. Les résultats obtenus montrent que l'outil est efficace pour manipuler des expressions régulières et effectuer une reconnaissance précise des motifs.

Importance des expressions régulières et de l'analyse lexicale :

- ♦ **Expressions régulières** : Elles permettent de définir des motifs complexes pour analyser et filtrer des chaînes de caractères. Leur puissance réside dans leur capacité à manipuler de grands ensembles de données textuelles.
- ♦ **Analyse lexicale** : C'est une étape essentielle dans les compilateurs et les outils de traitement de texte. Elle simplifie l'identification et la classification des unités lexicales (tokens), un processus fondamental dans le développement de langages de programmation et d'interfaces utilisateur.

Grâce à ce TP, nous avons acquis une compréhension pratique de l'utilisation de LEX pour développer un analyseur lexical performant.

Difficultés rencontrées

Au cours de la réalisation de ce travail pratique, nous avons rencontré quelques défis tels que:

1. **Gestion des erreurs lexicales** : Une difficulté majeure résidait dans la prise en charge des chaînes non reconnues ou ne correspondant à aucun des motifs définis. Cela impliquait une adaptation pour garantir que le programme reste robuste face à ces cas.
2. **Environnement de développement** : L'utilisation de l'outil EditPlus ou d'autres éditeurs pour configurer et exécuter le code a parfois posé des problèmes, tels que des erreurs de compilation dues à une mauvaise configuration ou des dépendances manquantes.

Ces difficultés ont été surmontées grâce à une approche méthodique incluant des tests progressifs, une recherche approfondie sur les expressions régulières et une collaboration efficace pour résoudre les problèmes rencontrés.

NB: Vous trouverez dans un fichier zippé tout les fichiers avec lesquels nous avons travaillé plus haut: **code.lex** qui est le **fichier.l** demandé, et le fichier **code** qui est l'exécutable pour que l'on ait plus besoin de compiler le fichier code.lex avant de l'exécuter.