TP de Lex.
Outil d'analyse lexicale.

Toute remarque, critique, suggestion, problème sur ce document est bienvenue !
Si il y a un truc qui vous paraît louche, peut-être qu'un de vos camarades aura compris de travers ce point ; en me signaler l'erreur ou l'ambiguité vous lui rendez service.

Présentation de Lex

Lex est un outil permettant d'écrire des programmes en C qui sont des analyseurs lexicaux.

Ce sont des programmes qui vont :

1. prendre en entrée une suite de caractères (lus dans un fichier, ou directement tapés au clavier) ;
2. reconnaître dans cette suite de caractères les mots appartenant à un (ou des) langage(s) reconnaissables (chaque langage est défini par une expression rationnelle) ;
3. exécuter une action lorsqu'ils ont reconnu un mot : .cela peut être une « traduction, un affichage, n'importe quel traitement…

Lex fait partie d'un couple d'outils très utilisés pour écrire un compilateur.

Lex/Yacc Lex pour le lexical et Yacc pour le syntaxique (la grammaire).

Flex/Bison (les versions GNU de Lex/Yacc).

Remarques :

- Avec Lex/Yacc (et les connaissances qui viendront plus tard), vous pouvez écrire un compilateur.
- Lex/Yacc et Flex/Bison sont en C, mais existe des versions dans d'autres langages.
- dans toute distribution Linux.
- ce n'est qu'une intro, on ne verra pas tout en 6h. C'est plus complet, voir par exemple

  http://www.delafond.org/traducmanfr/man/man1/flex.1.html

Sur les machines du campus, Il n'y a que flex et bison d'installés (l'appel de lex ou yacc redirigent vers respectivement flex et yacc, et c'est souvent ainsi sur les machines linux). Je ne sais pas ce que vous avez installé sur votre machine. Mais attention :

- il peut y avoir quelques différences entre lex et flex, et entre ma version, et ce qui est installé sur votre machine.
- comme tous les compilateurs, ou générateurs de code, c'est assez sensible au format d'entrée. Un blanc ou un saut de ligne mal ou pas placé génère une erreur pas toujours évidente à trouver.
- Je risque d'utiliser indifféremment lex et flex dans ce document par inattention.

Bon, normalement vous avez installé sur votre machine :

- **Un compilateur C.**
- **flex** ou **lex**

Je n'ai pas windows sur ma machine, ni n'ai de mac, je ne vais pas pouvoir vous aider pour l'installation ni la configuration… Je suis désolé… Je vous conseille de vous entraider sur ce point.

Utilisation.

Utilisation spécification lex seule

1) On écrit une spécification Lex dans un fichier **`exemple.l`**

Cette spécification contiendra (On verra le détail plus loin):

- des déclarations C utilisées dans le programme final.

- l'ensemble des expressions régulières dont on souhaite reconnaître les mots.

- pour chaque expression régulière, une ou plusieurs instructions C : les instructions que l'on veut effectuer chaque fois que l'on a lu un mot reconnu par cette expression

2) On compile cette spécification avec **`lex exemple.l`**

Cette compilation crée un fichier C **`lex.yy.c`** qui est un programme C qui contient le code :

- de l'automate reconnaissant les expressions régulières de la spécification,
- plus les morceaux de code C qui se trouvent dans la spécification. Ces morceaux de code C sont recopiés tels quels pour le moment.

La fonction essentielle de ce programme est **yylex()**, c'est la fonction qui contient l'automate, et lit par défaut sur l'entrée standard. (On verra le détail plus tard).

3) On compile ce programme C pour obtenir un executable **`a.out`**

```
cc lex.yy.c -ll
```

- **`-ll`** pour utiliser la **l**ibrairie qui correspond à **l**ex. (attention, sur certaines installations, il faut utiliser **-lfl** pour la librairie de **fl**ex)
- si a.out ne convient pas comme nom, bien sur on utilisera l'option `-o` (**cc lex.yy.c -ll -o exemple.exe** crée un exécutable **exemple.exe)**

Attention : ce n'est qu'à ce moment que les morceaux de code C que vous avez mis dans exemple.l sont compilés. Si vous avez oublié une accolade ou une parenthèse, cela peut être compliqué à comprendre et à corriger, parce que l'erreur n'est pas détectée ors de la première compilation

4) On lance l'exécutable a.out (ou **exemple.exe**).

Comportement par défaut de l'executable:

1. Lit sur l'entrée standard,
2. Pour chaque mot reconnu d'une expression régulière, exécute les instructions C indiquées dans

la au niveau de cette expression régulière.
3. Si une lettre n'est pas reconnue, l'affiche telle quelle sur la sortie standard.

Bien sûr on peut faire en sorte de lire dans un fichier, que ce soit en redirigeant l'entrée standard de l'exécutable ou bien autrement.

Rappel : rediriger l'entrée et la sortie standard en unix d'une commande **programme** se fait avec **programme < fic.entree > fic.Sortie**
Au lieu de lire sur le clavier, le **programme** lit dans le fichier f**ic.entree** et écrit dans le fichier **fic.sortie**.

Structure d'une Spécification Lex

Une spécification lex (un fichier lex) est découpé en trois Parties séparées par des %%, **tous seuls**, **et en début de ligne**, et tout seuls sur leur ligne.

Partie I : déclarations C, et déclarations de définitions lex.

%%

Partie II : Expressions régulières et actions associées.

%%

Partie III : Programme principal et fonction en C

La partie I peut être découpée en deux parties en insérant %{ et%} comme suit :

(les < > ne sont pas à écrire, mais les **%%** et **%{** et **%{** et **{}** entourant les instructions C sont à écrire dans le code regarder sur les exemples qui vous sont donnés dans le sujet de TP.)

```
%{
        inclusion de bibliothèques en C et déclarations de variables
%}
   définitions de notions lex écrites comme :
<identificateur>                <expression régulière>
...
%%
   règles écrites comme :
<expression régulière>       {<instructions  en C>}
%%
<programme principal et fonctions en C>
```

Seule la Partie II est obligatoire. Il y a un programme principal par défaut.

La mise en page est TRÈS importante

- `%}`, `%{`, `%%`, doivent être placés au début de la ligne.
- **Pour les définitions :**
  - l'identificateur doit être placé en début de ligne, puis au moins un blanc, puis l'expression régulière
- **Pour les règles :**
  - l'expression régulière doit être placée en début de ligne (sur une seule ligne), puis au moins un blanc puis la commande en C (Optionnelle).
  - un bloc d'instruction C peut s'étaler et être indenté sur plusieurs lignes. Mais surtout en ne commençant pas au premier caractère de la ligne !

comportement parfois bizarre...

exemple : le programme **doit** contenir au moins un saut de ligne à la fin.

La partie I contiendra :

- **dans la partie déclaration C :**
  - les inclusions de bibliothèques, par exemple #include <stdio.h> pour afficher.
  - et les déclarations C (types, variables et fonctions auxiliaires) utiles partout dans le programme final (par exemple si on a besoin d'un compteur, c'est là qu'on le mettra)
- **dans la partie définition lex :**
  - les définitions, cela permet d'utiliser plusieurs fois la même sous expression rationnelle dans le programme. Par exemple, il est très possible que l'on ait besoin dans plusieurs expressions rationnelles des caracteres compris entre 0 et 9, dans ce cas, on mettra quelque chose pour définir un chiffre
    `chif [0-9]`
  - d'autres commandes spécifiques à Lex, par exemple mode de départ, voir en détail plus tard.

La partie II Contient :

Les expressions rationnelles, avec le code correspondant au comportement souhaité pour chaque mot reconnu. Exemple :

```
{chif}+\.{chif}+   {printf("c'est un réél non signé\n");}
```

Dans ce cas, chaque fois que le programme lira un réel dans l'entrée, il affichera en sortie « c'est un réél non signé »

La partie III contient :

a) Soit le programme principal (précédé de fonctions auxiliaires s'il y a lieu et).

Exemple de programme principal.

```
int main() {
        nb = 0 ;
        yylex() ;
        printf(" on a compté %d trucs \n ", nb);
}
```

la fonction ***yylex()*** est la fonction qui :

- demande des trucs en entrée au clavier,
- les reconnaît,
- et exécute les actions associées à chaque expression rationnelle.

Quand l'entrée est terminée (^D au clavier, ou fin de fichier d'entrée), la fonction ***yylex*** termine.

b) soit Rien.

Dans ce cas là, il y a une fonction main par défaut qui est :

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

Ecriture des expressions régulières.

méta charactères `$ ^ () {} <> +- * / | ?` , ont une signification spéciale, les autres représentent ce qu'ils sont, si on a besoin d'un métacaractère, par exemple le `(`, soit on fait `"("`, soit `\(`

```
    Symbole  |  Signification
-------------+------------------
           x  |  Le caractere 'x'
           .  |  N'importe quel caractere sauf \n
      [xyz]   |  Soit x, soit y, soit z
      [^bz]   |  Tous les caracteres, SAUF b et z
      [a-z]   |  N'importe quel caractere entre a et z
     [^a-z]   |  Tous les caracteres, SAUF ceux compris entre a et z
         R*   |  Zero R ou plus, ou R est n'importe quelle expression reguliere
         R+   |  Un R ou plus
         R?   |  Zero ou un R (c'est-a-dire un R optionnel)
        (R)   |  R
     R{2,5}   |  Entre deux et cinq R
      R{2,}   |  Deux R ou plus
       R{2}   |  Exactement deux R
"[xyz\"foo"   |  La chaine '[xyz"foo'
   {NOTION}   |  L'expansion de la notion NOTION definie plus haut
         \X   |  Si X est un 'a', 'b', 'f', 'n', 'r', 't', ou
              |  'v', represente l'interpretation ANSI-C de \X.
         \0   |  Caractere ASCII 0
       \123   |  Caractere ASCII dont le numero est 123 EN OCTAL
       \x2A   |  Caractere ASCII en hexadecimal
         RS   |  R suivi de S
        R|S   |  R ou S
        R/S   |  R, seulement s'il est suivi par S
         ^R   |  R, mais seulement en debut de ligne
         R$   |  R, mais seulement en fin de ligne
     <<EOF>>  | Fin de fichier

\b backspace \n saut de ligne \r retour chariot \f saut de page \t tabulation
```

Comportement par défaut :

Par défaut la fonction **yylex()** lit sur l'entrée standard (i.e. le clavier si l'entrée standard n'a pas été redirigée.

Pour chaque mot reconnu, elle exécute l'action correspondante, et "absorbe" les caractères du mot.

Si un même mot correspond à plusieurs règles, on choisit la règle placée en premier.

Si pas d'action associée, le/les caractères sont absorbés.

Si aucune règle ne s'applique, le caractère est recopié tel quel sur la sortie standard.

Il peut y avoir plusieurs règles reconnues en même temps.

Dans ce cas, c’est celle qui permet de reconnaître le mot le plus long qui est choisie. En cas d’égalité de longueur, c’est la première des deux.

Exemple : nous avons dans la partie II

(ab)+ { printf (‘’ un mot composé de ab ‘’) ;}

[ab]+ { printf(‘’ des a ou des b ‘’) ;}

d {compteur ++ ; }

et le mot en entrée est *ababbacddababd*

Au début le curseur est au début

| ababbacddababd

1) deux règles peuvent s’appliquer :

la première permet de reconnaître **abab,**

la deuxième permet de reconnaître **ababba** qui est plus long

=> On choisit la deuxième règle, le curseur avance de 6 caractères et afffiche « des a et des b »

ababba | cddababd

2) aucune règle ne s’applique, aucune expression ne commence par un c :

=> le curseur avance d’une lettre et laffiche telle quelle

ababbac | ddababd

3) la troisième règle s’applique

=> le curseur avance du ‘d’ et fait compteur++

ababbacd | dababd

4) la troisième règle s’applique

=> le curseur avance du ‘d’ et fait compteur++

ababbacdd | ababd

5) deux règles peuvent s’appliquer :

la première permet de reconnaître **abab,**

la deuxième permet de reconnaître **abab** de même longueur

=> On choisit la première règle, le curseur avance de 4caractères et afffiche « des ab »

ababbaddabab | d

Si plusieurs mots possibles, prend le mot le plus long, par exemple `123.456ab` en entrée, et que l'on a définit une règle pour les entiers, et une règle pour les réels, va choisir la règle des réels. car plus long.

Primitives et fonctions de lex.

Que vous pouvez utiliser dans les parties C.

int yylex()	analyseur lexical, retourne 0 à la fin du fichier d'entrée.
char * yytext	contient la chaîne de caractère reconnue
int yyleng	longueur de la chaîne de caractère reconnue
void yyless(int nb)	remet nb caractères dans le flot d'entrée.
void reject(),	remet complètement le mot dans le flot d'entree (equivalent yyless(yyleng)).
yyinput	flot d'entrée
yyoutput	flot de sortie

char get_char() e int unget_char(char c) , recupère et remet un caractère dans le fot ld'entrée.

(yylval, mais utile que pour yacc, vous verrez l'an prochain).

Activation/desactivations de règles.

Il est possible d'activer ou non un certain jeu de regles.

a) On definit des identificateurs de jeurs dans la section de définition exemple

%start nom1 nom2 nom3

b) Les regles sont préfixées (ou pas) par

<nomI> ou

<nomI,nomJ,nomK>

les regles non préfixées sont toujours actives.

c) Pour changer de jeu de règle ?

BEGIN nomI ;

(à mettre dans le code C), passe dans le jeu de regle nomI (+ les regles non prefixées).

BEGIN 0 ;

(on n'utilise plus que les règles non préfixées).

Prise en compte ou nom de la casse (minuscule ou majuscule)

%option case-insensitive

À mettre dans la section de définitions lex, permet de remplacer chaque lettre x par [xX].

Regarder un manuel plus complet, il y a beaucoup d'autres possibilités... Mais on n'a que 4 h

pour ce TP.

- par exemple choisir le niveau d'optimisation, la table de caractère utilisée, etc... relancer plusieurs fois yylex, sur des fichiers différents, etc...

Vous trouverez beaucoup de sites sur internet présentant plus en détail les fonctionnalités, entre autres https://www.epaperpress.com/lexandyacc/prl.html (même s'il n'est pas très complet, il fait un petit tour clair de lex et yacc).

Programmation séparée, (ou bien lien avec une spécification yacc.)

Dans le cas de plus gros programme (ou bien avec une spécification syntaxique) on incluera le fichier lex.yy.c dans le programme principal, que l'on aura écrit par ailleurs, ou bien on fera de la compilation séparée. (Voir cours de C).

On ecrit :

une specification ***specif.lex*** (sans main)

un programme C ***principal.c*** contenant un main, avec certainement **au moins** l'appel à yylex(). (sinon ce serait complètement inutile.

on execute :

1) ***lex specif.lex*** pour obtenir le fichier lex.yy.c

2) ***cc -c lex.yy.c*** pour obtenir le fichier objet lex.yy.o

3) ***cc -c principal.c*** pour obtenir le fichier objet principal.o

4) ***cc principal.o lex.yy.o -ll -o monexecutatble*** pour obtenir l'executable souhaité