

Théorie des langages et de la compilation
Partie Pratique

Année académique 2008 – 2009

Séance 1: Langages réguliers et automates finis

Pour la théorie, se reporter au chapitre 2 du syllabus.

Certains exercices de cette séance sont extraits et/ou adaptés des exercices du livre *Introduction to Automata Theory, Languages and Computation*, seconde édition, J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley, 2000.

Exercice 1

Démontrez, à l'aide de la définition inductive des langages réguliers, que les deux langages suivants sont réguliers (l'alphabet considéré est $\Sigma = \{0, 1\}$) :

1. L'ensemble des mots composés d'un nombre arbitraire de 1, suivis de 01, suivis d'un nombre arbitraire de 0.
2. L'ensemble des nombres binaires impairs.

Exercice 2

1. Démontrez que tout langage fini est régulier.
2. Le langage $L = \{0^n 1^n \mid n = 0, 1, 2, \dots\}$ est-il régulier ? Expliquez.

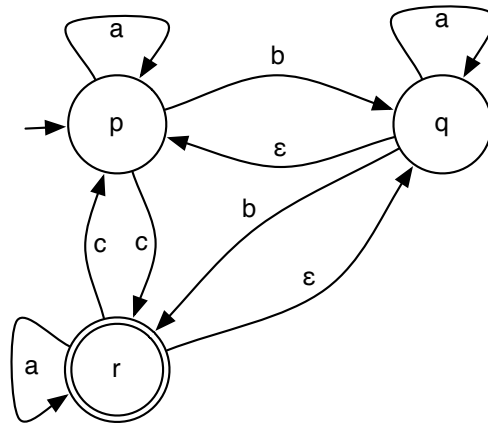
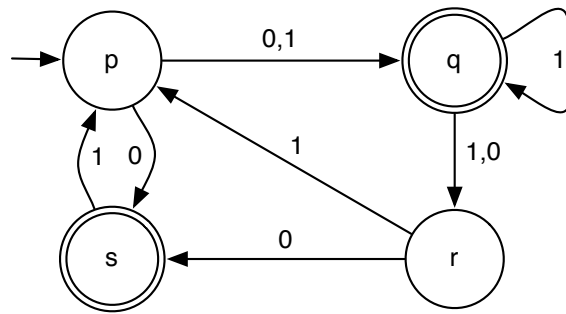
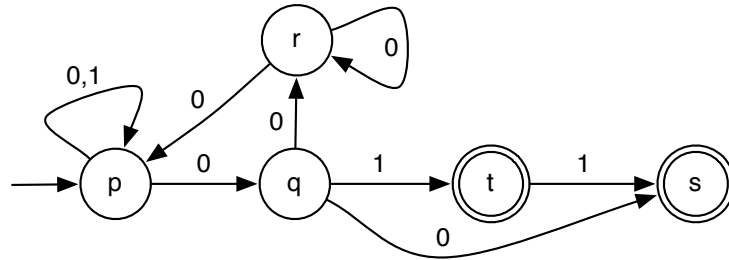
Exercice 3

Donnez un automate non déterministe qui accepte chacun des langages suivants (définis sur l'alphabet $\Sigma = \{0, 1\}$) :

1. Toutes les chaînes qui se terminent par 00.
2. Toutes les chaînes dont le 10ème symbole, compté à partir de la fin de la chaîne, est un 1.
3. Ensemble de toutes les chaînes dans lesquelles chaque paire de 0 apparaît devant une paire de 1.
4. Ensemble de toutes les chaînes ne contenant pas 101.
5. Tous les nombres binaires divisibles par 4.

Exercice 4

Déterminez les automates suivants :



Exercice 5

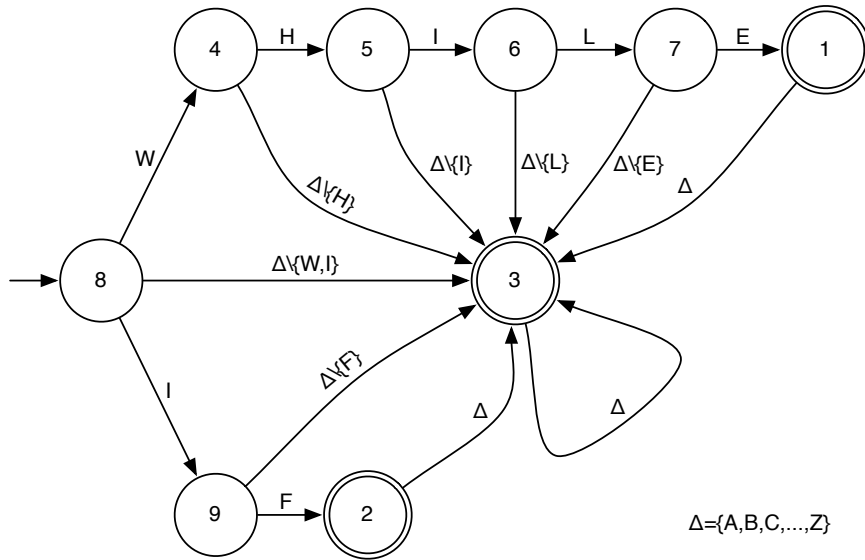
Voici un automate qui peut servir à analyser des programmes écrits dans un langage ayant WHILE et IF comme mots clef. Toutes les autres chaînes de caractères ($\{A, B, \dots, Z\}$) sont considérées comme des noms de variables. Les états accepteurs permettent de différencier ces trois cas.

Sur base de cet automate, écrivez une fonction (C, C++,...) qui lit une

chaîne de caractères sur input (caractère par caractère) et qui renvoie :

- 1 si la chaîne lue est **WHILE** ;
- 2 si la chaîne lue est **IF** ;
- 3 si la chaîne lue est un nom de variable.

Attention ! IFAB, par exemple, est un nom de variable valide.



Séance 2: Langages réguliers et expressions régulières

Pour la théorie, se reporter au chapitre 3 du syllabus.

Certains exercices de cette séance sont extraits et/ou adaptés des exercices du livre *Introduction to Automata Theory, Languages and Computation*, seconde édition, J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley, 2000.

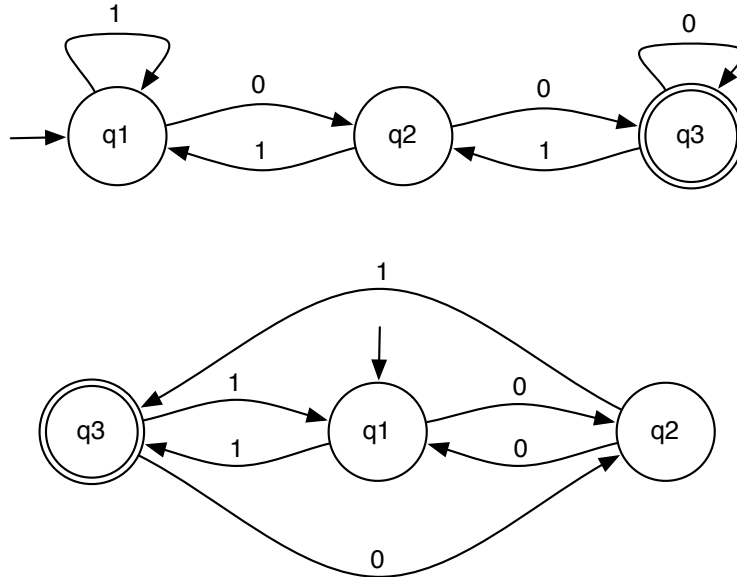
Exercice 1

Donnez une expression régulière qui accepte chacun des langages suivants (définis sur l'alphabet $\Sigma = \{0, 1\}$) :

1. Toutes les chaînes qui se terminent par 00.
2. Toutes les chaînes dont le 10ème symbole, compté à partir de la fin de la chaîne, est un 1.
3. Ensemble de toutes les chaînes dans lesquelles chaque paire de 0 apparaît devant une paire de 1.
4. Ensemble de toutes les chaînes ne contenant pas 101.
5. Tous les nombres binaires divisibles par 4.

Exercice 2

Construisez les expressions régulières qui correspondent aux automates suivants :



Exercice 3

Convertissez les expressions régulières suivantes en NFA_ε :

1. 01^*
2. $(0 + 1)01$
3. $00(0 + 1)^*$

Exercice 4

1. Donnez l'expression régulière étendue (ERE) qui désigne n'importe quelle suite de 5 caractères, y compris $\backslash n$.
2. Donnez l'ERE qui désigne une chaîne formée de n'importe quel nombre de \backslash , suivi de n'importe quel nombre de $*$.
3. Les shells UNIX (du type `bash`) permettent d'écrire des fichiers *batch* dans lesquels on peut insérer des commentaires. Une *ligne* est considérée comme commentaire si elle commence par $\#$. Quelle est l'ERE qui accepte de tels commentaires ?
4. Donnez l'ERE qui désigne un nombre en notation scientifique. Ce nombre sera composé d'au moins un chiffre. Il comportera deux parties optionnelles : une partie « décimale » (un $.$ suivi d'une série de chiffres) et une partie « exposant » (un E suivi d'un nombre entier, éventuellement préfixé d'un $+$ ou un $-$).

5. Donnez l'ERE acceptant l'ensemble des phrases « correctes » selon les critères suivants :
- Le premier mot de la phrase a une majuscule ;
 - la phrase se termine par un point ;
 - la phrase est composée d'un ou plusieurs mots (caractères `a...z` et `A...Z`), séparés par un espace ;
 - on trouve une phrase par ligne.
- Remarquons que les caractères de ponctuation autres que le point ne sont pas admis.
6. Écrivez l'ERE qui accepte tous les noms de fichiers DOS (composés de 8 caractères : `A...Z`, `a...z` et `_`), dont l'extension est `ext` et commençant par la chaîne `abcde`. Attention, l'ERE ne doit accepter que *le nom du fichier sans l'extension!*

Séance 3: Introduction aux grammaires

Pour la théorie, se reporter aux chapitres 4 à 6 du syllabus.

Exercice 1

Décrivez, en *français*, les langages générés par les grammaires suivantes, ainsi que leur type (hiérarchie de Chomsky) :

$$\begin{array}{lll} S \rightarrow abcA & & S \rightarrow 0 \\ Aabc & & S \rightarrow 1 \\ 1. \quad A \rightarrow \varepsilon & 2. & 1S \\ Aa \rightarrow Sa & & \\ cA \rightarrow cS & & \\ & & 3. \quad S \rightarrow a \\ & & *SS \\ & & +SS \end{array}$$

Exercice 2

Soit la grammaire \mathcal{G} suivante :

$$\begin{array}{l} S \rightarrow AB \\ A \rightarrow Aa \\ \quad bB \\ B \rightarrow a \\ \quad Sb \end{array}$$

Cette grammaire est-elle régulière ?

Donnez l'arbre de dérivation pour les formes phrasées suivantes :

- $baabaab$
- $bBABb$
- $baSb$

Par ailleurs, donnez les dérivations gauche et droite de $baabaab$.

Exercice 3

1. Écrivez une grammaire *context-free* qui génère toutes les chaînes de a et de b (dans n'importe quel ordre), tel qu'il y a plus de a que de b . Testez votre grammaire sur $baaba$ en donnant sa dérivation.

2. Écrivez une grammaire *context-sensitive* qui génère toutes les chaînes de a , de b et de c (quel que soit l'ordre), ayant le même nombre de a , de b et de c . Donnez la dérivation de $cacbab$ selon votre grammaire.

Exercice 4

1. Écrivez une grammaire context-free pour les langages $0^i 1^n 2^n$ et $0^n 1^n 2^i$, avec $n, i > 0$

2. Montrer que soient L_1 et L_2 deux langages context-free, $L_1 \cap L_2$ n'est pas nécessairement context-free.

Séance 4: Analyseurs gauches et droits

Pour la théorie, se reporter aux chapitres 8 et 9 du syllabus.

Exercice 1

Construisez l'automate à pile qui accepte le langage composé de tous les mots de la forme ww^R , où w est un mot quelconque sur l'alphabet $\{a, b\}$ et w^R est son image miroir.

Exercice 2

En utilisant la grammaire donnée en figure 1,

1. construisez l'arbre de dérivation de

```
begin
  ID := ID - INTLIT + ID ;
end
```

2. simulez l'analyseur gauche sur le mot :

```
begin
  A := BB - 314 +A ;
end
```

3. simulez l'analyseur droit sur ce mot.

1	<program>	→	begin <statement list> end
2	<statement list>	→	<statement> <statement tail>
3	<statement tail>	→	<statement> <statement tail>
4	<statement tail>	→	ϵ
5	<statement>	→	ID :=<expression>
6	<statement>	→	read (<id list>)
7	<statement>	→	write (<expr list>)
8	<id list>	→	ID <id tail>
9	<id tail>	→	, ID <id tail>
10	<id tail>	→	ϵ
11	<expr list>	→	<expression> <expr tail>
12	<expr tail>	→	. <expression> <expr tail>
13	<expr tail>	→	ϵ
14	<expression>	→	<primary> <primary tail>
15	<primary tail>	→	<add op> <primary> <primary tail>
16	<primary tail>	→	ϵ
17	<primary>	→	(<expression>)
18	<primary>	→	ID
19	<primary>	→	INTLIT
20	<add op>	→	+
21	<add op>	→	-
22	<System goal>	→	<program>

FIGURE 1 – Grammaire des séances 4 et 5

Séance 5: Analyseurs (suite)

Pour la théorie, se reporter aux chapitres 8 et 9 du syllabus.

Exercice 1

En tenant compte de la grammaire de la figure 1 :

1. Donnez le $\text{First}^1(A)$ et le $\text{Follow}^1(A)$ pour tout $A \in V$.
2. Donnez le $\text{First}^2(\langle \text{expression} \rangle)$ et le $\text{Follow}^2(\langle \text{expression} \rangle)$.

Exercice 2

Lesquelles de ces grammaires sont LL(1) ?

$$1. \begin{cases} S \rightarrow ABBA \\ A \rightarrow a|\varepsilon \\ B \rightarrow b|\varepsilon \end{cases}$$

$$2. \begin{cases} S \rightarrow aSe|B \\ B \rightarrow bBe|C \\ C \rightarrow cCe|d \end{cases}$$

$$3. \begin{cases} S \rightarrow ABc \\ A \rightarrow a|\varepsilon \\ B \rightarrow b|\varepsilon \end{cases}$$

$$4. \begin{cases} S \rightarrow Ab \\ A \rightarrow a|B|\varepsilon \\ B \rightarrow b|\varepsilon \end{cases}$$

Exercice 3

Construisez la table des actions pour la grammaire suivante :

- 0 $S \rightarrow \text{exp } \$$
- 1 $\text{exp} \rightarrow - \text{exp}$
- 2 $\text{exp} \rightarrow (\text{exp})$
- 3 $\text{exp} \rightarrow \text{var expr-tail}$
- 4 $\text{exp-tail} \rightarrow - \text{exp}$
- 5 $\text{exp-tail} \rightarrow \varepsilon$
- 6 $\text{var} \rightarrow \text{ID var-tail}$
- 7 $\text{var-tail} \rightarrow (\text{exp})$
- 8 $\text{var-tail} \rightarrow \varepsilon$

Exercice 4

Écrivez (en C, C++, ...) un analyseur gauche en descente récursive pour les règles 14 à 21 de la grammaire de la figure 1.

Séance 6: Retour aux grammaires

Pour la théorie, se reporter au chapitre 6 du syllabus.

Exercice 1

Supprimez les symboles inutiles dans les grammaires suivantes :

$$\left\{ \begin{array}{l} S \rightarrow a|A \\ A \rightarrow AB \\ B \rightarrow b \end{array} \right. \quad \left\{ \begin{array}{l} S \rightarrow A|B \\ A \rightarrow aB|bS|b \\ B \rightarrow AB|Ba \\ C \rightarrow AS|b \end{array} \right.$$

Exercice 2

Soit la grammaire :

$$\left\{ \begin{array}{l} E \rightarrow E \text{ op } E \\ \quad \quad \quad ID[E] \\ \quad \quad \quad ID \\ op \rightarrow * \\ \quad \quad \quad / \\ \quad \quad \quad + \\ \quad \quad \quad - \\ \quad \quad \quad - > \end{array} \right.$$

- Montrez que $2 + 3 \rightarrow 4 + 5$ possède plusieurs dérivations.
- L'ordre de priorité des opérateurs est le suivant : $\{[], - >\} > \{*, /\} > \{+, -\}$.
Modifiez cette grammaire de manière à prendre en compte la priorité des opérateurs, ainsi que l'associativité à gauche.
- Dérivez $2 + 3 \rightarrow 4 + 5$ avec votre nouvelle grammaire.

Exercice 3

Appliquez l'algorithme de dérécursification à la grammaire suivante :

$$\left\{ \begin{array}{l} E \rightarrow E + T | T \\ T \rightarrow T * P | P \\ P \rightarrow ID \end{array} \right.$$

Exercice 4

Factorisez la règle :

$$\begin{array}{l} \langle \text{stmt} \rangle \rightarrow \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{stmt list} \rangle \text{ end if} \\ \langle \text{stmt} \rangle \rightarrow \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{stmt list} \rangle \text{ else } \langle \text{stmt list} \rangle \text{ end if} \end{array}$$

Exercice 5

Exemple de question d'examen

Transformez la grammaire suivante pour la rendre LL(1) :

$$\left\{ \begin{array}{l} S \rightarrow aE|bF \\ E \rightarrow bE|\epsilon \\ F \rightarrow aF|aG|aHD \\ G \rightarrow Gc|d \\ H \rightarrow Ca \\ C \rightarrow Hb \\ D \rightarrow ab \end{array} \right.$$

Séance 7: La génération de code

Pour la théorie, se reporter aux chapitres 11 et 12 du syllabus.

P-Langage

Exercice 1

Écrivez le code qui calcule et affiche la valeur de :

$$(3 + x) * (9 - y)$$

où x est une valeur lue sur *input* et y est la valeur qui se trouve en mémoire à l'adresse 0 (on verra plus tard comment placer correctement une valeur en mémoire).

Exercice 2

Écrire le code qui affiche toutes les valeurs impaires dans l'intervalle $[7, 31]$. Pour ce faire vous aurez besoin de l'instruction `dupl i` qui duplique le sommet (entier) de la pile.

Exercice 3

Donner la structure d'un if-else en P-langage.

Exercice 4

Écrire le code qui :

- Se réserve de la place pour deux variables que nous appellerons a et b .
- Initialise a et puis b avec des valeurs lues sur *input*.
- Ajoute 5 à a .
- Divise b par 2.
- Si $a > b$, affiche a , sinon affiche b .

Tout au long de l'exercice, les valeurs stockées dans les cases correspondant à a et b devront rester cohérentes.

Analyse sémantique

Exercice 5

Voici une règle pour définir le `if` d'un langage impératif :

$$\begin{aligned}\langle \text{Si} \rangle &\rightarrow \text{if} \langle \text{Cond} \rangle \text{then} \langle \text{Code} \rangle \langle \text{SuiteSi} \rangle \\ \langle \text{SuiteSi} \rangle &\rightarrow \text{else} \langle \text{Code} \rangle \text{endif} \\ \langle \text{SuiteSi} \rangle &\rightarrow \text{endif}\end{aligned}$$

Indiquez quelles instructions en P-langage vous devez générer pour traduire ce code dans un compilateur en descente récursive. Faites l'hypothèse que les règles correspondant à $\langle \text{Cond} \rangle$ et $\langle \text{Code} \rangle$ sont déjà décorées pour générer le code correct.

Exercice 6

- Réécrivez la grammaire suivante en tenant compte de la priorité et de l'associativité des opérateurs :

$$E \rightarrow E \text{ op } E \mid (E) \mid \text{nb}$$

$$\text{op} \rightarrow + \mid - \mid * \mid /$$

- Associez à cette grammaire les règles et attributs nécessaires pour calculer la valeur d'une expression E .
- Dérécursifiez cette grammaire et adaptez les règles.

Séance 8: Pumping Lemma

Pour la théorie, se reporter aux chapitres 2 et 7 du syllabus.

Exercice 1

Donnez, de manière formelle, la contraposée du *Pumping Lemma* pour les langages réguliers.

Exercice 2

1. Le langage $L = \{1^n \mid \exists m \in \mathbb{N} : n = m^2\}$ est-il régulier ? Prouvez votre réponse
2. Étant donné un mot $w \in \Sigma^*$, on dénote par w^M l'image miroir de ce mot. On définit alors le langage $L_p = \{ww^M \mid w \in \Sigma^*\}$, qui contient donc tous les palindromes. Par exemple (L. Merce) : « A man, a plan, a canal : Panama ! » Étant donné l'alphabet $\Sigma = \{0, 1\}$, L_p est-il régulier ? Prouvez votre réponse.

Exercice 3

Soit w un mot sur l'alphabet Σ . Pour tout $c \in \Sigma$, nous notons $|w|_c$ pour représenter le nombre de c qui apparaissent dans w . Le langage de Dyck est le langage de tous les mots $w_1 \dots w_n$ sur $\Sigma = \{(\cdot), (\cdot)\}$ tels que : $|w|_> = |w|_< \wedge \forall 1 \leq i \leq n : w_i = > \rightarrow |w_1 \dots w_{i-1}|_< > |w_i \dots w_{i-1}|_<$. Ce langage est-il régulier ?

Exercice 4

Donnez, de manière formelle, la contraposée du *Pumping Lemma* pour les langages hors contexte.

Exercice 5

Montrez que $L = \{a^{n^2} \mid n \geq 1\}$ n'est pas un CFL.

Séance 9: Analyseurs $LR(0)$ et $LR(k)$

Pour la théorie, se reporter au chapitre 10 du syllabus.

$LR(0)$

Exercice 1

Soit la grammaire :

1. $S' \rightarrow S\$$
2. $S \rightarrow aCd$
3. $S \rightarrow bD$
4. $S \rightarrow Cf$
5. $C \rightarrow eD$
6. $C \rightarrow Fg$
7. $C \rightarrow CF$
8. $F \rightarrow z$
9. $D \rightarrow y$

Construisez-en l'automate canonique, et la table des actions.

Exercice 2

Simulez le fonctionnement de l'analyseur que vous avez construit à l'exercice 1 sur la chaîne `aeyzdz`.

$LR(k)$

Exercice 3

Construisez l'analyseur LR(1) pour la grammaire :

1. $S' \rightarrow S\$$
2. $S \rightarrow A$
3. $A \rightarrow bB$
4. $A \rightarrow a$
5. $B \rightarrow cC$
6. $B \rightarrow cCe$
7. $C \rightarrow dAf$

Cette grammaire est-elle LR(0)? Justifiez.

Exercice 4

Construisez l'analyseur LR(1) pour la grammaire :

1. $S' \rightarrow S\$$
2. $S \rightarrow SaSb$
3. $S \rightarrow c$
4. $S \rightarrow \varepsilon$

Simulez-en le fonctionnement sur `abacb`.

Séance 10: Analyseurs $SLR(1)$ et $LALR(1)$

Pour la théorie, se reporter au chapitre 10 du syllabus.

Exercice 1

Construisez l'analyseur $SLR(1)$ pour la grammaire :

- (1) $S' \rightarrow S\$$
- (2) $S \rightarrow A$
- (3) $A \rightarrow bB$
- (4) $A \rightarrow a$
- (5) $B \rightarrow cC$
- (6) $B \rightarrow cCe$
- (7) $C \rightarrow dAf$

Exercice 2

Construisez l'analyseur $LALR(1)$ pour la même grammaire.

Séance 11: Lex

Pour la théorie, se reporter au chapitre 3 du syllabus.

Exercice 1

1. Écrire un filtre qui compte le nombre de caractères, de mots et de lignes d'un fichier.
2. Écrire un filtre qui numérote les lignes d'un fichier (hormis les lignes blanches).
3. Écrire un filtre qui n'imprime que les commentaires d'un programmes. Ceux-ci sont compris entre { }.
4. Écrire un filtre qui transforme un texte en remplaçant le mot `compilateur` par `beurk` si la ligne débute par `a`, par `schtroumpf` si la ligne débute par `b` et par `youpi !!` si la ligne débute par `c`.
5. Écrire une *fonction d'analyse lexicale* à l'aide de LEX. Les *tokens* reconnus seront :
 - Les nombres décimaux (en notation scientifique) ;
 - Les identifiants de variables ;
 - Les opérateurs relationnels (`<`, `>`, `≤`, *etc*) ;
 - Les mots-clés `si`, `sinon` et `alors`.Cette fonction a pour but d'être utilisée dans un analyseur syntaxique écrit en YACC.

Exercice 2

Écrire un petit programme qui *embellit* du code C à l'aide de (f)lex. Ce programme lira un fichier C et l'affichera ainsi :

- Indenter correctement le code ;
- Les mots clef en gras (`while`, `for`, ...) ;
- Les *strings* (délimités par des ") en vert ;
- Les entiers en bleu ;
- Les commentaires (délimités pas `/*` et `*/`, ou bien les lignes commençant par `//`) en noir sur fond blanc (vidéo inverse).

Pour ce faire, vous pouvez vous aider de la fonction `void textcolor(int attr, int fg, int bg)`

- `attr` permet de mettre en gras (valeur `BRIGHT`), en vidéo inverse (`REVERSE`) ou en normal (`RESET`).
- `fg` et `bg` permettent de préciser la couleur du texte et du fond (valeurs `GREEN`, `BLUE`, `WHITE`, `BLACK`...)

Séance 12: Yacc

Pour la théorie, se reporter au chapitre 10 du syllabus.

Exercice 1

Vous avez reçu une spécification LEX et une spécification YACC (voir feuilles annexes).

1. Décrivez en français le langage reconnu par le compilateur qu'on obtiendrait à partir de ces spécifications ;
2. Modifiez la spécification pour n'accepter que les polynômes d'une seule variable. On entre un polynôme par ligne, et chaque polynôme porte sur une variable différente.
3. Ajoutez le code nécessaire pour afficher la dérivée première du polynôme. Par exemple :

```
2x^3+2x^2+5
Derivee premiere: 6x^2+4x
```

4. Ajoutez la possibilité de reconnaître des produits de polynômes. Le symbole de multiplication `*` devra être précisé explicitement dans l'*input*. Adaptez les actions qui affichent la dérivée. Par exemple :

```
(3x^2+6x)*(9x+4)
La derivee premiere de (3x^2+6x)*(9x+4) est
((3x^2+6x)*(9))+((6x+6)*(9x+4))
```

5. Ajoutez les actions nécessaires pour évaluer le polynôme et sa dérivée en un point donné. L'utilisateur entrera la valeur de la variable, suivie de ; puis du polynôme (le tout sur une même ligne). Par exemple :

```
2 ; (3x^2+6x)*(9x+4)
La derivee premiere de (3x^2+6x)*(9x+4) est
((3x^2+6x)*(9))+((6x+6)*(9x+4))
p(2)=528, p'(2)=612
```

La spécification Lex

```
chiffre [0-9]
lettre  [a-zA-Z]
entier  {chiffre}+
var     {lettre}+
%{
    #include "derive.tab.h"
    int yywrap (void) {}
%}
```

```

%%

{entier} {return ENTIER;}
{var}    {return VAR;}
" "      {}
.        {return yytext[0];}
"\n"    {return yytext[0];}

```

```
%%
```

La spécification Yacc

```
%token ENTIER
```

```
%token VAR
```

```
%left '+' '-'
```

```
%left '*' '/'
```

```
%%
```

```

input : line input    {}
      | line          {}
      ;

```

```

line : polynome '\n'  {printf("OK\n");}
      ;

```

```

polynome : terme '+' polynome {}
          | terme '-' polynome {}
          | terme              {}
          ;

```

```

terme    : '-' terme      {}
          | VAR '^' ENTIER {}
          | ENTIER VAR '^' ENTIER {}
          | VAR           {}
          | ENTIER VAR    {}
          | ENTIER        {}
          ;

```

```
%%
```

```

int main (void)
{yyparse();}

```

```

int yyerror (char * s)
{printf("yyerror: I encountered an error: %s.\n\n", s);}

```