

---

# Projet de Théorie des Langages (Compte-Rendu)

---

Auteur : Florian Legendre



# Légendes et Abréviations utilisées

**Question :** Ceci est une question de l'enseignant

**Réponse :** Ceci est une réponse de l'enseignant ou validée par l'enseignant

**Réponse :** *Ceci est une réponse du ou d'un des auteurs non validée par l'enseignant*

```
1 Ceci est du code source.  
2 Selon les langages, différents mots seront colorés selon  
3 si ce sont des mots clefs ou non (comme int, char, etc.).
```

**Listing 1** – Exemple de code source

```
Ceci est un formatage automatique Latex d'un texte copié-collé  
directement depuis un terminal Bash ayant valeur de capture  
d'écran. La coloration correspond à une coloration quelconque  
d'un terminal Bash (les chemins étant habituellement coloré et  
le nom de l'utilisateur aussi comme crex@crex:~$ ...)
```

**Listing 2** – Exemple d'une pseudo capture d'écran Bash

# Table des matières

<b>I</b>	<b>Analyse de billet de concert</b>	<b>3</b>
0.1	Exercice 1 : Traitement d'une commande de billets . . . . .	4
<b>II</b>	<b>Des automates en récursif</b>	<b>6</b>
0.2	Exercice 2 : Programmation en dur de manière récursive . . . . .	7

Première partie

Analyse de billet de concert

## Exercice 1 : Traitement d'une commande de billets

### Spécification de Concert.1.lex : reconnaissance des champs clefs

```
1  alpha  [a-zA-Z]
2  codeConcert  ^T[0-9]{2,6}
3  codeDossier  [0-9]{8}
4  date  [0-9][0-9]\\/[0-9]{1,2}(\\/[0-9][0-9])?
5  FL  (\\n)
6  heure  [0-9][0-9]:[0-9][0-9]
7  ignore  [\\t ]+
8  nbPlaces  [1-9]?[0-9](\\ places)
9  nomConcert  [A-Z0-9]([A-Z0-9]|(-[A-Z0-9]))*
10 nomPropre  {alpha}({alpha}|(-{alpha}))*
11 prenomNom  ^{nomPropre}\\/{nomPropre}
12
13
14 %%
15 {codeConcert}      {printf("codeConcert");}
16 {codeDossier}      {printf("codeDossier");}
17 {date}             {printf("date");}
18 DOSSIER             {printf("dossier");}
19 <<EOF>>             {printf("FinFichier\\n"); return 0;}
20 {FL}               {printf(" FL\\n");}
21 {heure}            {printf("heure");}
22 {ignore}           {printf(" ");}
23 {nbPlaces}         {printf("nb places");}
24 {nomConcert}        {printf("nomConcert");}
25 {prenomNom}        {printf("prenomNom");}
```

**Listing 3** – Première spécification en vue d'un test de reconnaissance des différents champs d'une commande de billets

## Spécification de Concert.2.lex : application de la reconnaissance à un besoin 'réel'

```

1  %{
2      char* codeDossier;
3      char* prenomNom;
4      int nbPlaces = 0;
5      int nbConcerts = 0;
6  %}
7
8  alpha [a-zA-Z]
9  codeConcert ^T[0-9]{2,6}
10 codeDossier [0-9]{8}
11 date [0-9][0-9]\/[0-9]{1,2}(\/[0-9][0-9])?
12 FL (\n)
13 heure [0-9][0-9]:[0-9][0-9]
14 ignore [\t ]+
15 nbPlaces [1-9]?[0-9]
16 nomConcert [A-Z0-9]([A-Z0-9]|(-[A-Z0-9]))*
17 nomPropre {alpha}({alpha}|(-{alpha})) *
18 prenomNom ^{nomPropre}\/{nomPropre}
19
20
21 %%
22 {codeConcert}      {nbConcerts++;}
23 {codeDossier}      {codeDossier=strdup(yytext);}
24 {date}             {}
25 DOSSIER            {}
26 <<EOF>>            {return 0;}
27 {FL}               {}
28 {heure}            {}
29 {ignore}           {}
30 {nbPlaces}         {nbPlaces+=strtol(yytext, NULL, 10);}
31 {nomConcert}        {}
32 places             {}
33 {prenomNom}        {prenomNom=strdup(yytext);}
34
35 %%
36 int main()
37 {
38     yylex();
39     printf("Pour le dossier %s, %s a acheté %i places de %i concerts\n",
40           codeDossier, prenomNom, nbPlaces, nbConcerts);
41 }

```

**Listing 4** – Seconde spécification appliquant la reconnaissance des différents champs d'une commande de billets

Deuxième partie

Des automates en récursif

## Exercice 2 : Programmation en dur de manière récursive

### Questions de compréhension

**Question :** Si votre automate a N états, combien de fonctions reconnaitRec\_i devez vous écrire ?

**Réponse :** Si l'automate a N états alors il faudra écrire N fonctions reconnaîtRec\_i. En effet, dans les faits nous sommes en train d'implanter un système d'équations aux langages.

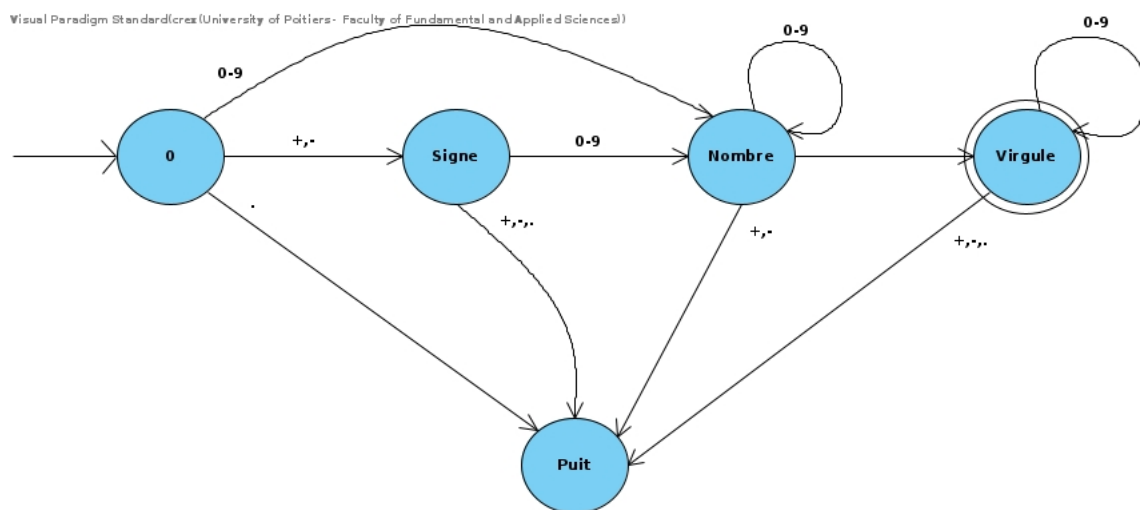
**Question :** Si l'état  $i$  est final, que doit retourner `reconnaitRec_i("")` ? Et si  $i$  n'est pas final ?

**Réponse :** Un état  $i$  final signifie que `reconnaitRec_i("")` doit retourner 'true', "" étant le mot vide aussi appelé  $\epsilon$ . Tout état  $i$  non final doit alors retourner 'false' pour le mot vide.

**Question :** Si le paramètre 'mot' n'est pas vide et commence par un caractère c, quelle fonction reconnaitRec\_i(mot) doit-elle appeler? Et avec quel paramètre?

**Réponse :** Si le paramètre 'mot' n'est pas vide et commence par un caractère c alors on doit appeler la fonction reconnaîtRec\_i(mot) qui correspond à l'état de destination dans la transition  $q_{courant} \xrightarrow{c} q_i$ . On appelle alors cette fonction avec pour paramètre le mot 'mot' tronqué de sa première lettre.

## Automate des réels





Implantation des reconnaîtRec

Implantation de l'automate complet

Programme complet