MiniProjet d'ITL. à déposer avant le 30 avril 2021 minuit

- Ce projet est à rendre en monôme. Vous devrez rendre une archive votreNom.zip contenant deux répertoires `Billet` et `Automates`, chacun contenant une des deux parties de ce mini-projet.
- Certaines questions ne demandent pas de programmation, vous répondrez à ces questions dans un document pdf.
- **Attention :** nous attacherons une grande importance à la *propreté* et à la lisibilité de vos spécifications et programme et si vous respectez toutes les contraintes du sujet.

Analyse de billet de concert

Nous devons traiter automatiquement des commandes de billets de concert, dont voici un exemple.

```
DOSSIER 12345678
ANNE-CLAIRE/BERGEY
T1114 LES-CHATS-SAUVAGES  14/04/21   19:30   2 places
T1239 LES-12-HEURES           11/7   12:30   4 places
T32   CAROTTES             22/4/21   18:30  11 places
```

Une commande est constituée exactement ainsi :

1. Sur la première ligne, le mot clé `DOSSIER` commençant en première colonne, écrit en majuscule puis le *codeDossier*, un entier composé d'exactement 8 chiffres.
2. Sur la deuxième ligne, nous trouvons le `prenomNom`, toujours en première colonne, composé du prénom et nom (écrits en majuscules, sans caractères accentués) séparés par un `/`. Si il y a un nom composé, on utilisera un tiret : un nom ou prénom ne peut commencer ni terminer par un tiret, et on ne doit pouvoir utiliser deux tirets à la suite. On ne met pas de blancs à l'intérieur de `prenomNom`.
3. Puis nous avons une suite de lignes (chacune séparée de la suivante par une seule fin de ligne). Nous trouvons sur chaque ligne les informations d'un concert dans cet ordre.
   (a) le `codeConcert`, toujours en première colonne, code du concert composé de la lettre `T` suivi de 2 à 6 chiffres.
   (b) le `nomConcert`, nom du concert écrit en majuscules ou chiffres, sans blanc ni ponctuation ni lettre accentuée. Comme pour les noms, on utilisera les tirets pour séparer les mots, sans pouvoir en mettre deux à la suite, et on ne peut mettre de blancs dans le nom du concert.
   (c) une `date` écrite sous le format `DD/MM` ou `DD/M` ou `DD/MM/YY` ou `DD/M/YY`
   (d) une `heure` écrite sous le format `HH:MM`
   (e) l'entier `nb` indiquant le nombre de places, cet entier est compris entre 1 et 99,
   (f) le mot clé `places` écrit en minuscules.
4. Enfin la fin de fichier.

L'utilisation de la fin de ligne est stricte, on ne peut insérer deux fins de ligne à la suite, ni grouper deux concerts sur la même ligne. Par ailleurs, ces différents mots sont séparés par des blancs ou des tabulations qui ne jouent que le rôle de séparateur (on peut en mettre plusieurs à la suite sans importance). Enfin, le mot clé DOSSIER, la personne ainsi que les codesDesConcerts doivent commencer en première colonne.

Exercice 1 Traitement d'une commande de billets

1. Écrire une spécification `Concert.1.lex` reconnaissant chacun des éléments (y compris la fin de ligne) intervenant dans une commande de billet. Cette spécification devra permettre d'afficher en sortie chaque token reconnu, en éliminant blancs et tabulations en trop. sur l'exemple, la sortie doit être exactement la suivante.

```
dossier codeDossier FL
prenomNom FL
codeConcert nomConcert date heure nb places FL
codeConcert nomConcert date heure nb places FL
codeConcert nomConcert date heure nb places FL
FinFichier
```

   Attention : faire en sorte que si le nom d'un concert est `DOSSIER`, ce nom soit correctement traité (i.e. ce nom doit apparaître comme `nomConcert`, et pas comme `dossier`).

2. Compléter la spécification précédente en écrivant la spécification `Concert.2.lex`. Elle permettra de résumer les informations de cette commande (l'affichage précédent est supprimé) : Lors du traitement du fichier, on affiche le numéro de dossier, le *prenomNom* de la personne ainsi que les nombres de places commandés ainsi que le nombre de concerts pour lesquels elle a commandé des places.

   Sur l'exemple, vous afficherez : `Pour le dossier 12345678, ANNE-CLAIRE/BERGEY a acheté 17 places de 3 concerts.`

Des automates en récursif

Pour les exercices 2 et 4, nous souhaitons écrire un programme reconnaissant les réels écrits comme suit : un signe (`+` ou `-`) optionnel, suivi d'une suite non vide de chiffres (`0`,`1`,`2`,`3`,`4`,`5`,`6`,`7`,`8`,`9`) suivi d'un point, suivi d'une suite vide ou non de chiffres. Le problème est simplifié car nous ne gérons pas les exposants.

Vous pourrez utiliser Ada, C/C++, Java, ou OCamL, (nous demander pour tout autre langage) pour écrire les programmes demandés. Chaque programme final devra avoir un petit main permettant de tester les fonctions demandées.

Exercice 2 *Programmation en dur de manière récursive :* l'automate est modélisé en *"dur"* dans le code même par un ensemble de fonctions récursives.

Soit un automate $\mathcal{A}$. On peut supposer que les états sont numérotés de 0 à $N-1$, où N est le nombre d'états; et l'état 0 est l'état initial.

Vous allez programmer cet automate en écrivant : *en dur*, un ensemble de fonctions *récursives croisées* (qui s'appellent les unes les autres). Comme vous décrirez cet automate *en dur* dans le code, il faudrait réécrire complètement l'ensemble de toutes ces fonctions pour un autre automate que $\mathcal{A}$.

Principe :
Pour chaque état q, vous allez écrire une fonction `reconnaitRec_q(m:string) return boolean` dont la signification sera : *"Nous sommes sur l'état q, et nous sommes en train de tester si le mot m est reconnu à partir de q, cette fonction retourne vrai si m est reconnu depuis q, faux sinon"*.

La fonction `reconnaitRec_q1` appellera la fonction `reconnaitRec_q2` s'il y a une transition $q_1 \xrightarrow{c} q_2$, et que le mot m commence par la lettre c. C'est la façon dont s'appelleront les unes les autres toutes ces fonctions qui représentera l'automate et le chemin lors de la reconnaissance d'un mot.

1. Ne prenez pas encore le clavier, prenez un papier et un crayon ! Pour chaque état i de l'automate, il faudra écrire une fonction *reconnaitRec_i(mot : string) retourne boolean*, elle retourne `vrai` si `mot` est reconnu en partant de l'état `i`, `faux` sinon. Répondez aux 3 questions suivantes avant de commencer la programmation de ces fonctions `reconnaitRec_i`.
   - Si votre automate a N états, combien de fonctions `reconnaitRec_i` devez vous écrire ?
   - Si l'état i est final, que doit retourner `reconnaitRec_i("")` ? Et si i n'est pas final ?
   - Si `mot <> ""` et commence par un caractère c, quelle fonction `reconnaitRec_i(mot)` doit-elle appeler ? Et avec quel paramètre ?

2. Donner (dans un document pdf) l'automate $\mathcal{A}_{\mathcal{R}}$ reconnaissant les rééls sans exposant.

3. Vous pouvez maintenant reprendre le clavier. Écrivez toutes les fonctions `reconnaitRec_i(m:string)` nécessaires pour implémenter un automate reconnaissant un réel.

4. Une fois que vous avez écrit toutes ces fonctions `reconnaitRec_i`, l'écriture de la fonction *reconnaitReelRec(Mot : String) return boolean* tient en une ligne.

5. Utiliser tout ceci pour écrire un programme testant les chaines `123.`, `123.45`, `-123.`, `+123.34`, `-123.34`, `12A3.34`, `123..33`, `123.34.44`, `.34`, pour savoir si elles représentent des rééls. Puis ce programme demande répétitivement une chaine en entrée, puis indique si cette chaine représente un réel ou non.

Exercice 3 *Des automates non déterministes représentés dans le code de manière récursive.*
Dans l'exercice précédent, l'automate était déterministe. On veut maintenant représenter un automate non déterministe à 5 états (numérotés de 0 à 4) dont voici les transitions principales : $0 \xrightarrow{a} 1$, $0 \xrightarrow{a} 2$, $1 \xrightarrow{b} 1$, $1 \xrightarrow{\varepsilon} 2$, $2 \xrightarrow{c} 2$, $1 \xrightarrow{a} 3$, $2 \xrightarrow{a} 3$.
L'état 4 est l'état puits, toutes les autres transitions y mènent.
L'état 3 est le seul état terminal.
Cet automate reconnait L_4, le langage défini (sauf erreur de ma part) par l'expression rationnelle $ab^*c^*a + ac^*a$.

Comme dans l'exercice 2 vous écrirez les fonctions récursives *reconnait_i* associées aux états de l'automate, ainsi que la fonction *reconnaitRec* reconnaissant les mots de ce langage.

1. Répondre dans un document texte :
   - Comment dans le code de *reconnait_0* allez vous représenter le fait qu'en lisant un a, on puisse aller soit de l'état 0 à l'état 1, soit de l'état 0 à l'état 2 ?
   - Comment dans le code de *reconnait_1* allez vous représenter le fait que l'on peut passer directement, sans rien lire, à l'état 2 ?

2. Écrire les 5 fonctions `reconnaitRec_i` et la fonction `reconnaitRecL4` de cet exercice. Les utiliser pour tester si `abbcca`, `accca` et `abbccccba` sont reconnus par cet automate.

Exercice 4 *Évaluation du réél correspondant à la chaîne de caractères*

La fonction *reconnaitRecReel* de l'exercice 2 indiquait si une chaîne de caractères représentait un réel ou non, mais n'indiquait pas sa valeur. Nous souhaitons nous inspirer de l'exercice 2 pour écrire une fonction *evalueRecReel(mot : String)*, fonction calculant le réél correspondant au *mot* passé en entrée, elle devra retourner un enregistrement indiquant à la fois, si *mot* représentait un réél, et si oui, la valeur du réél représenté.

La structure de *evalueRecReel* (utilisant un ensemble de routines récursives associées à chaque état n'est pas fondamentalement changée), mais il faudra maintenant gérer le calcul de la valeur finale (et peut-être le calcul de valeurs intermédiaires).

Attention : quelques points sur lesquels réfléchir avant de bouger les doigts sur le clavier.

1. Il peut être utile de séparer le calcul de la partie entière du calcul de la partie décimale avant de rassembler ces deux valeurs à un moment donné.
2. Comment allez vous gérer votre position dans la partie décimale (x^{eme} position après la virgule = indique la puissance de dix négative ?), et vous en servir pour prendre en compte la nouvelle décimale lue ?
3. Comment allez vous gérer le calcul de la partie entière, lorsqu'un nouveau chiffre est lu ?
4. Comment allez vous gérer la transmission du calcul d'une routine récursive à l'autre ? variables globales, paramètres d'entrée sortie, valeurs de retour de fonction ?

Si vous utilisez pour cela des variables globales, faites attention à l'endroit où vous les déclarez !

Questions :

• Écrire la fonction *evalueRecReel* (et toute autre fonction utile).

• Ecrire un programme testant si les chaines `3.14`, `002.24`, `-3.14`, `1000.14`, `123.`, `123.45`, `-123.`, `+123.34`, `-123.34`, `12A3.34`, `123..33`, `123.34.44`, `.34`, sont valides, et affichant leur valeur réélle.