TD

(CNC-2013)

- On attachera une importance à la concision, à la clarté, et à la précision de la rédaction
- Si, au cours du DS, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.
- Le devoir se compose de deux problèmes indépendants.
- Les questions non traitées peuvent être admises pour traiter des questions ultérieures.

Problème 1.

ANALYSEUR LEXICAL D'UN LANGAGE

Présentation. Avant son exécution, tout code source d'un programme informatique doit être traduit en un autre code appelé code machine. Cette traduction est réalisée par un compilateur dont le rôle est de transformer le code source en une suite d'instructions élémentaires directement exécutables par le processeur.

Ainsi le rôle de l'analyseur lexical est d'identifier puis supprimer les caractères superflus du code source (commentaires, espaces, ..) et de reconnaître les mots clés, les identificateurs, les opérateurs qui sont définis par un ensemble de règles.

En plus, l'analyseur lexical signale les éventuelles erreurs de syntaxe et associe à chaque erreur le numéro de ligne dans laquelle elle intervient.

Dans ce problème, on se propose de mettre en œuvre un analyseur lexical. Il s'agit d'implémenter des fonctions en langage PYTHON pour un analyseur lexical d'un pseudo langage informatique noté PL.

***

A- Gestion des commentaires et des espaces

Question A-1 : test d'un commentaire

Un commentaire est une instruction qui n'est pas traduite par le compilateur. Pour ce pseudo langage PL, un commentaire est une chaine de caractères qui doit commencer obligatoirement par les 2 caractères * (antislash étoile) et se terminer par les 2 caractères *\ (étoile antislash).

⊙ Ecrire une fonction ***comment(instr)*** qui retourne True si la chaîne de caractères instr est un commentaire du langage LIM et False sinon.

Exemple :

- soit la chaine de caractères instr= "* explication *\" alors l'appel de la fonction **comment(instr)** retourne True.
- si la chaine de caractères instr= "explication" alors l'appel **comment(instr)** retourne False.

Question A-2- Suppression d'espaces multiples dans une instruction

Dans une instruction, on a tendance à supprimer les espaces multiples qui se succèdent et de ne conserver qu'un seul espace.

Exemple. La chaîne de caractères inst= ' x = 21 ' ne doit pas être traitée par le processeur avant de supprimer les espaces superflus. On doit avoir le résultat suivant : inst= 'x = 21'.

⊙ Ecrire une fonction ***suppEspaces(inst)*** qui supprime les espaces multiples consécutifs (qui se suivent) entre les caractères de la chaîne ***inst*** en paramètre. S'il y'a plusieurs espaces au début ou à la fin, on ne conserve aucun.

B : Reconnaissance des mots-clés et des identificateurs.

Tout langage de programmation définit un ensemble de mots clés qui sont des chaînes de caractères ayant des significations et des utilisations spécifiques pour ce langage.

Pour notre pseudo langage PL, on suppose avoir défini :

- **MotsCles** : une liste qui contient les chaînes de caractères qui représentent les mots clés du notre pseudo langage PL. Pour le moment le tuple est : ***MotsCles = ['si','sinon','tantque','pour','definir']***.

Question B-1 : Vérification d'un mot clé

⊙ Ecrire une fonction ***motCle(mot)*** qui retourne True si ***mot*** (le paramètre de la fonction) est un mot clé de PL et False sinon.

Exemple. : l'appel **motCle("si")** retourne **True** et l'appel **motcle("entier")** retourne **False**.

Question B-2 : Validité d'un identificateur

Un identificateur est une chaîne de caractères qui permet de définir et d'identifier les éléments d'un programme (les variables, les fonctions, ..). Il doit respecter un ensemble de règles particulières pour chaque langage.

Un identificateur du pseudo langage PL est valide s'il respecte les quatre conditions (C1, C2, C3 et C4) suivantes :
C1- Sa longueur est inférieure strictement à 80
C2- Ne contient aucun espace
C3- Ne commence pas par un chiffre ('0','1','2','3','4','5','6','7','8','9').
C4- Ne doit pas être un mot clé du langage PL (n'est pas dans ***MotsCles***)

⊙ Ecrire une fonction ***identificateur(id)*** qui retourne True si son paramètre (la chaîne de caractères ***id***) est un identificateur valide de PL et False sinon.

C : Implémentation de l'analyseur lexical.

On suppose maintenant que les mots clés du pseudo langage PL sont mis dans un fichier texte de nom physique **"c:\fmotscles.txt"** supposé existant. Chaque ligne de ce fichier texte correspond à un mot clé de PL.

Question C-1 : Analyse d'une instruction

Une instruction du langage PL est correcte si elle vérifie l'une des conditions suivante :

- L'instruction est un commentaire du langage PL (voir question A-1).
- L'instruction commence par un identificateur valide suivi d'un espace suivi d'une chaîne de caractères. On doit supprimer les espaces superflus.
- L'instruction commence par un mot clé du langage PL suivi par un espace, suivi d'une chaîne de caractères et se termine par le caractère ' ;' (point virgule).

⊙ Ecrire une fonction ***analyserInstruction(inst)*** qui retourne True si la chaîne de caractères ***inst*** en paramètre correspond à une instruction correcte de PL et False sinon.

Exemple.

Soit le fichier **"c:\fmotscles.txt"** contenant les lignes suivantes :

si
sinon
tantque
pour
definir

L'appel de la fonction ***analyserInstruction***('si x<y ;') retourne Tue

L'appel de la fonction ***analyserInstruction***('5=7') retourne False.

L'appel de la fonction ***analyserInstruction***('alpha = 5.55') retourne Tue

Question C-2 Analyse d'un fichier source.

On suppose avoir écrit un lanage avec ce pseudo langage PL. le code source de ce programme est enregistré dans un fichier texte.

Chaque ligne de ce fichier correspond à une instruction de PL.

⊙Ecrire une fonction ***analyserSource(source)*** qui analyse le ficgier source dont le nom est en paramètre de la fonction. Cette fonction affiche sur l'écran les numéros de toutes les lignes du fichier source qui correspondent à des instructions incorrectes ou ffiche sur l'écran "succès" dans le cas où toutes les lignes du fichier source sont des instructions correctes.

Exemple

Soit le fichier **"c:\fmotscles.txt"** de l'exemple précédent.

et soit le fichier de nom "sourcePL" contenant les lignes suivantes du code source d'un programme écrit en pseudo langage PL :

```
x =5 ; y =7;
repeter (x<10) x=x+1 ;
\*instructions*\
5 = 7
Si (y==0) alors afficher(NUL) ;
Sinon afficher (Non NUL)
```

Exemple. Après l'appel de la fonction : ***analyserSource(sourcePL)*** on aura sur l'écran :
Les numéros des lignes correspondants à des instructions incorrectes sont : 2, 4, 6

Problème 2.

DETERMINATION DES ITINERAIRES

Présentation. Dans le monde actuel, il est très utile de pouvoir localiser géographiquement des personnes ou des objets aussi bien pour des besoins privés que professionnels.

Au niveau professionnel, la géo-localisation est très utilisée notamment pour la navigation routière mais aussi dans plusieurs autres domaines (Transport, logistique, énergie, ..) favorisant ainsi un gain de productivité et une sécurité accrue.

Des systèmes de géo-localisation – tels que GPS – permettent le repérage de la position géographique exacte de leurs usagers, ainsi que le calcul d'itinéraires.

C'est dans cette optique que s'inscrit le problème suivant concernant des algorithmes de détermination de chemins et d'itinéraires entre les points d'un plan.

***

Notations.

- Soit $\mathscr{P}$ un plan, on notera P(x, y), un point de coordonnées *x* et *y* dance ce plan représenté par un tuple *(x, y)*.
- Soit ***N*** une variable de type ***int*** strictement positive à laquelle on donnera la valeur 10 ***(N = 10)*** dans tout le problème. On notera $\mathscr{P}$ (N), l'ensemble des points du plan $\mathscr{P}$ ayant des coordonnées *x* et *y* entières positives telles que (0<=*x*< N) et (0<=*y*< N).
- Soient A et B deux points distincts de $\mathscr{P}$ (N). on appelle **chemin de A vers B**, une liste C, non vide et ordonnée de points distincts (tous différents) de $\mathscr{P}$ (N) tel que :

  $C = [(x_A, y_A), (x_1, y_1), .., (x_i, y_i), .., (x_B, y_B)]$

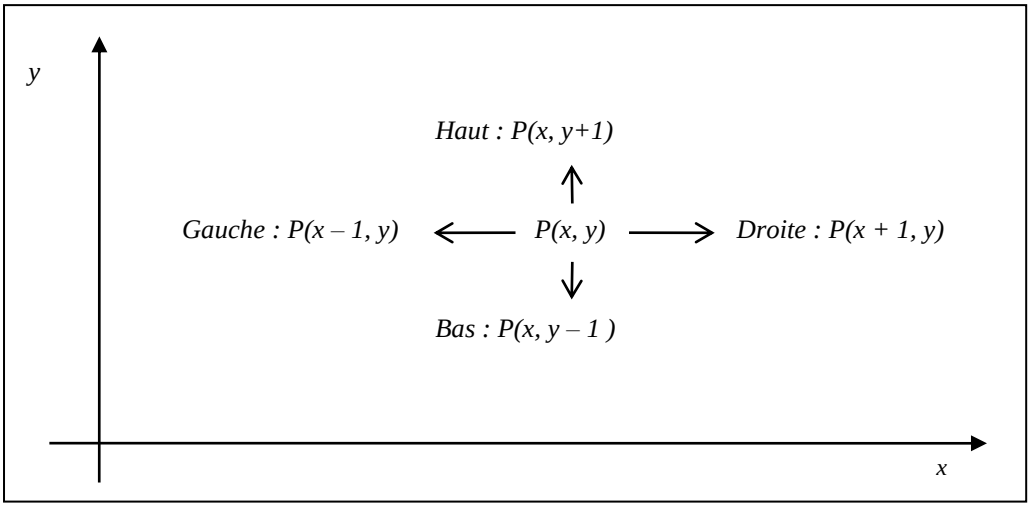
Chaque point *P(x, y)* de ce chemin est relié au point suivant *P(s,t)* par l'une des 4 relations suivantes :

a. Gauche : *P(s, t) est à gauche de P(x, y). c'est-à-dire :* ***s == x – 1*** *et* ***t == y***

b. Droite : *P(s, t) est à droite de P(x, y). c'est-à-dire :* ***s == x + 1*** *et* ***t == y***

c. Haut : *P(s, t) est au dessus de P(x, y). c'est-à-dire :* ***s == x*** *et* ***t == y + 1***

d. Bas : *P(s, t) est en dessous de P(x, y). c'est-à-dire :* ***s == x*** *et* ***t == y – 1***



Exemples de chemins. Soient ***A (2, 3)*** et ***B (5, 5)*** deux points de $\mathscr{P}$(10) :

C1 = [(2, 3), (3, 3), (4, 3), (4, 4), (4, 5), (5, 5)]

C2 = [(2, 3), (2, 4), (3, 4), (3, 5), (4, 5), (5, 5)]

Les listes C1 et C2 représentent 2 chemins différents de A vers B.

***

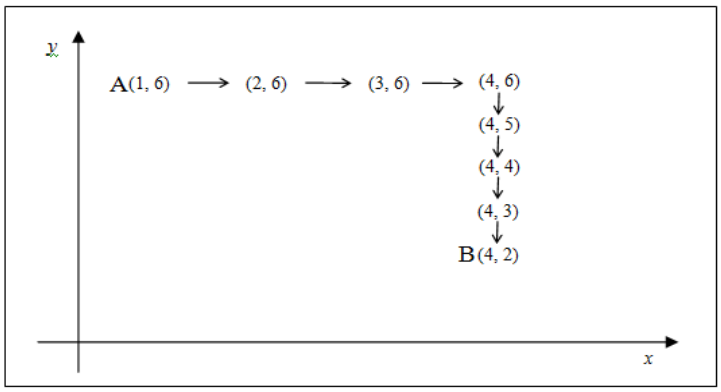
Construction de chemins.

- Soit ***Max*** une variable de type ***int*** strictement positive qui contient le nombre maximal de points que peut contenir un chemin entre 2 points quelconques.

Question 1. Ecrire une fonction ***initChemin( )*** qui initialise une liste de tuples par les valeurs *(-1, -1)* et la retourne comme résultat. Les points *P (-1, -1)* n'appartiennent pas à $\mathscr{P}$(N) mais permettent juste d'initialiser la liste.

Chemin horizontal puis vertical. Soient A et B deux points de $\mathscr{P}$(N). On appelle ***CheminHV*** de A vers B le chemin construit de telle sorte à se déplacer à partir de A horizontalement (soit à gauche, soit à droite) jusqu'à arriver au point ayant la même abscisse que B, puis se déplacer verticalement (soit en haut, soit en bas) jusqu'à arriver au point B.

Exemple. N = 10. Soient A= (1, 6) et B= (4, 2) de $\mathscr{P}$(10), alors ***CheminHV*** de A vers B est :
C = [(1, 6), (2, 6), (3, 6), (4, 6), (4, 5), (4, 4), (4, 3), (4, 2)]



Question 2. Ecrire une fonction ***cheminHV(A, B)*** qui crée et retourne la liste ***HVAB*** de tuples représentant le cheminHV de A vers B. Dans ce cas ***HVAB*[0]** contiendra le tuple (x_A, y_A). ***HVAB*[-1]** contiendra le tuple (x_B, y_B).

Distance d'un chemin et chemin minimal. Soit C un chemin reliant deux points A et B. on appelle distance du chemin C, le nombre de points du chemin C différents du point A.

Exemple. Soient A= (2, 6) et B= (4, 3) deux points et C = [(2, 6), (2, 5), (2, 4), (2, 3), (3, 3), (4, 3)] un chemin de A vers B. La distance de C est de **5**.

On considère maintenant plusieurs chemins différents et tous menant de A vers B. On représente ces chemins par un tableau double dimension (liste de listes) **tabC**.

Exemple. Pour les deux points A= (2,6) et B= (4, 3), **tabC** aura la représentation suivante :

tabC = [[(2, 6), (3, 6), (4, 6), (4, 5), (4, 4), (4, 3)] ,
[(2, 6), (2, 5), (2, 4), (2, 3), (2, 2), (3, 2), (3, 3), (4,3)] ,
[(2, 6), (2, 5), (2, 4), (3, 4), (3, 3), (4, 3)] ,
[(2, 6), (1, 6), (0, 6), (0, 5), (0, 4), (0, 3), (1, 3), (2, 3), (3, 3), (4, 3)]]

La distance minimale reliant les 2 points A et B (du tableau **tabC**) est 5.
Le chemin minimal est donc [(2, 6), (3, 6), (4, 6), (4, 5), (4, 4), (4, 3)].

Si on a plusieurs chemins minimaux, on prend le premier chemin minimal trouvé pendant le parcours du tableau **tabC**.

Question 3. Ecrire une fonction ***distance (num)*** qui retourne la distance d'un chemin num entre A et B du tableau **tabC**.

Question 4. Ecrire une fonction ***distanceMinimale(*tabC*)*** qui retourne la distance minimale de plusieurs chemins du tableau **tabC** passé en paramètre.

Question 5. Ecrire une fonction ***cheminMinimal(*tabC*)*** qui retourne le chemin minimal des chemins du tableau **tabC** passé en paramètre.

Question 6. Ecrire une fonction ***cheminRetour(C)*** qui retourne le chemin de retour qui est le chemin inverse de C. Si C est un chemin de A vers B, la fonction retourne le chemin de B vers A.