Université de Carthage Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul Département Mathématique	IPEIN المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2021/2022 Filière : MPCT Niveau d'étude : 2èmme année Semestre : 01 Nombre de pages : 03 Date : 11/11/2021 Durée : 1h :30

DS : Informatique

Exercice 1 : ---------- (6points)

Dans cet exercice, on se propose avoir créer le module **Pile.py** qui implémente la structure Pile où on a défini les fonctions suivantes :

- **CreerPile()** permettant de créer une pile vide
- **Empiler(P,e)** permettant d'ajouter un élément **e** à la pile
- **Depiler(P)** permettant de dépiler et retourner le sommet de la pile
- **EstVide(P)** à résultat booléen, permettant de vérifier si la pile est vide

On appelle mot de **Dyck** tout mot construit à partir de deux lettres **lettre1** et **lettre2** vérifiant les deux propriétés suivantes :

- Le mot doit commencer par **lettre1** et contenir autant de la **lettre1** que de la **lettre2**
- Dans tout préfixe du mot, le nombre d'occurrences de **lettre1** est supérieur ou égal au nombre d'occurrences de **lettre2**, c'est-à-dire lorsqu'on fait le parcours du mot du gauche à droite on a toujours, le **nombre_lettre1>=nombre_lettre2**

Exemple : Pour lettre1='E' et lettre 2='D':

- Le mot **'EDED'** est un mot **Dyck** car dans tous ses préfixes ; qui sont **'E', 'ED'**, **'EDE'** et **'EDED'** ;on a **nombre_lettre1>=nombre_lettre2**
- Le mot **'EEDE'** n'est pas un mot **Dyck** car **nombre_lettre1>nombre_lettre2**
- **Le mot 'EDDE'** n'est pas un mot **Dyck** car dans le préfixe **'EDD'** on a **nombre_lettre2>nombre_lettre1**

Travail demandé

On veut tester si un mot donné est un mot de **Dyck** par différentes méthodes. Pour cela on définira plusieurs fonctions dont chacune prend trois paramètres : une chaine de caractère **mot**, supposé de taille paire et composé uniquement des lettres **lettre1** et **lettre2**, ainsi que deux caractères **lettre1** et **lettre2** et qui retourne **True** si **mot** est un mot de **Dyck** et False sinon.

1. Ecrire une fonction **est_Dyck1(mot, lettre1 ,lettre2)** qui vérifie si **mot** est un mot de **Dyck**
2. Ecrire une fonction récursive **DyckR(mot, lettre1 ,lettre2)** qui vérifie si **mot** est un mot de **Dyck**

3. Ecrire une fonction **est_Dyck2(mot, lettre1 ,lettre2)** qui vérifie si **mot** est un mot de **Dyck en utilisant une pile**
   NB : Penser au principe de résolution de la **fonction TestParenthèse** permettant de vérifier si une chaine est bien parenthésée

<u>Problème : Ordonnancement des processus</u> ---------- (12points)

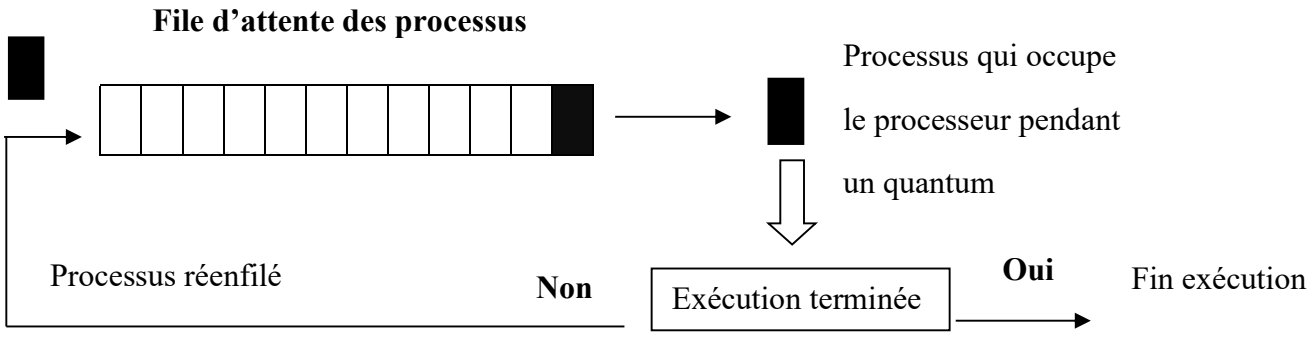
Dans ce problème, on se propose avoir créé le module **File.py** qui implémente la structure **File** où on a défini les fonctions suivantes :

- **CreerFile()** permettant de créer une file vide
- **Enfiler(F,e)** permettant d'ajouter un élément **e** à la file
- **Defiler(F)** permettant de défiler et retourner la tête de la file
- **EstVide(F)** à résultat booléen, permettant de vérifier si la file est vide

Le système d'exploitation d'un ordinateur doit gérer l'allocation du processeur aux différents processus (tache /application à exécuter par le processeur) à exécuter.

Parmi les algorithmes permettant l'ordonnancement de l'allocation du processus, on applique le Round-Robin dont le principe est le suivant :

- Les processus en attente de l'exécution seront stockés dans une file d'attente : une structure **FIFO** « premier entrant premier sortant »
- A son tour, le processus occupe le processeur mais pendant un temps fini appelé **quantum** (unités de temps) même si ce quantum n'est pas suffisant pour qu'il termine son exécution
- Si le processus n'a pas terminé son exécution, il sera réenfilé dans la file
- Tous les processus disposent du même quantum de temps que les autres
- Chaque processus peut passer par trois états :
  - ✓ **Prêt** : en attente d'exécution pour la première fois
  - ✓ **Actif** : en cours d'exécution (c'est-à-dire défilé de la file)
  - ✓ **Interrompu :** réenfilé dans la file



Travail Demandé :

Module processus.py

Dans la suite, chaque processus sera décrit par une liste de la forme :
[Identifiant :Temps_traitement, Temps_Attente, T] avec :

- **Identifiant :** une chaine de caractère qui représente l'identifiant du processus d'une façon **unique**
- **Temps_traitement** : Un entier qui représente le temps d'exécution du processus **en nombre de quantums**
- **Temps_Attente :** Entier qui représente le temps que le processus doit attendre avant de passer au processeur **(nb*quantum)**
- **T** est un tuple, de la forme **(prêt, actif, interrompu)** , décrivant l'état du processus comme suit :
    - **T= (1,0,0)** si le processus est en attente (prêt)
    - **T= (0,1,0)** si le processus est actif (défilé de la file)
    - **T= (0, 0,1)** si le processus est interrompu
- **Exemple : P=['processus1', 5,20,(1,0,0)]**

1. Ecrire une fonction **miseAjour(P, k,Q)** permettant de mettre à jour le temps d'attente du processus **P** en lui affectant **k*Q** avec **k** un entier représentant le nombre des quantums **Q** ajoutés.
2. Ecrire une fonction **AfficheEtat(P)** permettant d'afficher l'état d'un processus **P** passé comme paramètre selon ce modèle : « le **Processus1** est **en attente** »
3. Ecrire une fonction **ModifierEtat(P,etat)** qui prend comme paramètre le processus **P** et un caractère **c ,** représentant le nouveau état du processus :
   'P' pour prêt, 'A' pour actif et 'I' pour interrompu.
   Exemple : ModifierEtat(P,'A') modifie l'état du processus P en le rendant actif

Module FileAttente.py

En utilisant **seulement les primitives définies** dans le module **File.py** décrit en dessus, définir les fonctions suivantes :

1. **CopyF(F)** permettant de retourner une copie **FC** d'une file **F**
2. **Taille(F)** permettant de retourner le nombre des éléments de la file **F**
   NB : il est interdit d'utiliser la fonction prédéfinie « len »
3. **AppartientFile(F,ident)** à résultat booléen , permettant de vérifier si un identifiant **ident** est un des identifiants des processus de la file **F**
4. **Ajouter(F,Q)** prenant comme paramètre la file **F** ainsi que la valeur du quantum **Q** et permettant de saisir et enfiler un processus dans la file **F**

5. **Reduire(F,Q)** prenant comme paramètre la file **F** ainsi que la valeur du quantum **Q** et permettant de réduire le temps d'attente de tous les processus d'un quantum
6. **Executer(F,Q)** permettant d'exécuter le processus en tête de la file suivant le principe décrit en dessus
7. **Affiche(F)** permettant d'afficher tous les processus de la file **F** ainsi que leurs états.