Corrigé du sujet 24NSIJ2ME3

Exercice 1 (6 points)

Cet exercice porte sur l'exécution d'un programme Python et sur la décidabilité.

Partie A : boucle while

1. Lors de l'exécution de `f1(7)`, la variable `i` prend successivement les valeurs 7, 8, 9, 10, la fonction termine et renvoie 10.
2. Lors de l'exécution de `f1(-2)`, la variable `i` prend successivement les valeurs -2, -1, ..., 7, 8, 9, 10, la fonction termine et renvoie 10.
3. Lors de l'exécution de `f1(12)`, la variable `i` prend successivement les valeurs 12, 13, 14, 15, 16 ... la fonction ne termine pas.
4. La fonction `f1` termine lorsque son paramètre est un entier inférieur ou égal à 10.

Partie B : fonction récursive

5. L'appel `f2(4)` termine et renvoie `6`.
6. L'appel `f2(5)` ne termine pas car dans les appels successifs la variable `n` va rester impaire et ne sera donc jamais nulle.
7. L'appel `f2(n)` termine si son paramètre est un entier naturel pair.
8. La fonction récursive ci-dessous ne termina pour aucun entier `n`.

```
def infini(n):
    return infini(n-1)
```

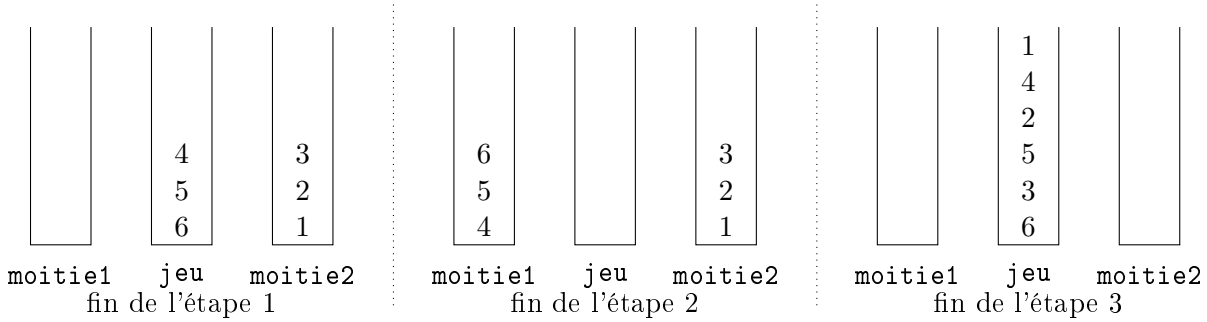
Partie C : le problème de l'arrêt

9. Dans le cas où `arret(code_paradoxe,code_paradoxe)` renvoie `True`, alors l'instruction `infini(42)` est exécutée, et elle ne termine pas, donc `paradoxe(code_paradoxe)` ne termine pas.
10. Dans le cas où `arret(code_paradoxe,code_paradoxe)` renvoie `False`, alors l'instruction `return 0` est exécutée, donc `paradoxe(code_paradoxe)` termine.
11. Par conséquent, l'appel `paradoxe(code_paradoxe)` ne termine pas si `arret(code_paradoxe,code_paradoxe)` renvoie `True` et termine si `arret(code_paradoxe,code_paradoxe)` renvoie `False`, ce qui est contradictoire avec la définition de la fonction `arrêt`.
    On en déduit, par l'absurde, qu'une fonction `arret` possédant la propriété souhaitée ne peut exister.

Exercice 2 (6 points)

Cet exercice porte sur la programmation Python, la programmation orientée objet, les tests et la structure de données pile.

1. On a représenté côte-à-côte le contenu des trois piles à la fin de chacune des trois étapes.



2. On complète le code de la fonction `produire_jeu`.

```
def produire_jeu(n):
    resultat=Pile()
    for i in range(n,0,-1):
        resultat.empile(i)
    return resultat
```

ou bien

```
def produire_jeu(n):
    resultat=Pile()
    for i in range(n):
        resultat.empile(n-i)
    return resultat
```

3. La première erreur est syntaxique, L3 il manque des parenthèses pour l'appel du constructeur `Pile`.
   La deuxième erreur est une erreur de logique, L4 et L6. Chaque moitié de la pile initiale est composée de `n//2` éléments et non `n`.
   Pour une meilleure lisibilité, on a repris l'ensemble de la fonction.

```
1 def scinder_jeu(p,n):
2     m1=Pile()
3     m2=Pile()
4     for i in range(n//2):
5         m1.empile(p.depile())
6     for i in range(n//2):
7         m2.empile(p.depile())
8     return m1,m2
```

4. On écrit une fonction `recombiner` (qui s'appuie sur le fait que les deux piles passées en paramètres ont la même taille).

```
def recombiner(m1,m2):
    res=Pile()
    while not m1.est_vide():
        res.empile(m1.depile())
        res.empile(m2.depile())
    return res
```

5. On écrit la fonction `faro(p,n)` demandée, en s'appuyant sur les fonctions déjà écrites.

```
def faro(p,n):
    m1,m2=scinder_jeu(p,n)
    return recombiner(m1,m2)
```

6. On a rajouté deux cas d'usage pour couvrir les deux cas demandés.

```
p3=Pile()
for k in [1,2,3]:
    p3.empile(k)
p4=Pile()
for k in [4,2,3]:
    p4.empile(k)
assert not identiques(p3,p4)
p5=Pile()
for k in [1,2,3]:
    p5.empile(k)
p6=Pile()
for k in [1,2,3]:
    p6.empile(k)
assert identiques(p5,p6)
```

7. On écrit la fonction `ordre_faro(n)` demandée.

```
def ordre_faro(n):
    pile=produire_jeu(n)
    temoin=produire_jeu(n)
    pile=faro(pile,n)
    k=1
    while not identiques(pile,temoin):
        k=k+1
        pile=faro(pile,n)
    return k
```

Exercice 3 (8 points)

Cet exercice porte sur les réseaux, les protocoles réseau, les bases de données relationnelles et les requêtes SQL.

Partie A : Configuration réseau dans la DMZ

1. Une adresse IPv4 se copose de quatre octets (écris sous forme décimale et séparés par des points).
2. L'adresse IP du serveur web de la DMZ sera 172.16.0.1 et celle du serveur de base de données 172.16.0.2.
3. La commande `ping` permet de tester la connectivité entre deux ordinateurs.
4. Ici le problème est que la passerelle par défaut du poste PC_A1 n'est pas sur le même réseau que ce poste. Le plus simple est probablement de configurer la passerelle par défaut à 192.168.1.254 qui est une autre interface du routeur A, inteface située sur le réseau 192.168.1.0/24 comme l'interface du poste PC_A1.

Partie B : routage

5. En utilisant les tables de routage, le chemin de PC_A1 à Serveur_impression est PC_A1->Routeur A-> Routeur B-> Routeur C->Routeur D->Serveur_impression.
6. Si le lien C-D est coupé, ce chemin devient PC_A1->Routeur A-> Routeur B-> Routeur C et le paquet ne parvient pas à destination.
7. On complète la table de routage du routeur C avec le protocole RIP.

Routeur C		
Destination	Prochain saut	Métrique
172.16.0.0	10.0.2.2	2
192.168.0.0	10.0.2.2	2
192.168.1.0	10.0.2.2	2
192.168.2.0	10.0.3.2	1
192.168.3.0		
10.0.0.0	10.0.2.2	1
10.0.1.0	10.0.2.2	1
10.0.2.0	-	-
10.0.3.0	-	-
10.0.4.0	-	-
10.0.5.0	10.0.3.2	1
0.0.0.0	10.0.2.2	

8. Avec le protocole RIP, le chemin de PC_A1 à Serveur_impression est PC_A1->Routeur A->Routeur B->Routeur C->Routeur D->Serveur_impression.
9. Ce choix n'est pas le meilleur choix possible si on prend en considération les débits des liaisons, puisque la liaison C->D n'a qu'un débit de 10 Mb/s, alors qu'en passant par E on a des débits 100 fois plus importants, de 1 Gb/s.
10. Avec une coupure du lien C->D, les lignes modifiées dans la table de routage du routeur C sont proposées ci-dessous

192.168.2.0	10.0.4.2	2
10.0.3.0		
10.0.5.0	10.0.4.2	1

et le nouveau chemin de PC_A1 à Serveur_impression devient PC_A1->Routeur A->Routeur B->Routeur C->Routeur E->Routeur D->Serveur_impression.

Partie C : Exploitation de la base de données

11. La requête `SELECT titre_parution FROM parution;` permet d'obtenir la liste de tous les titres parus.
12. La requête `SELECT num_parution, numero FROM page WHERE mise_en_forme='Arial,12' ORDER BY num_parution;` permet d'obtenir les numéros de parution et les numéros de page des pages en Arial 12, triés par numéro de parution.

13. La requête `SELECT num_image, titre_image, poids FROM image WHERE poids >1000;` permet d'obtenir la liste des images de poids supérieur à 1000 Ko.
14. La requête
    ```
    SELECT num_parution
    FROM parution
    JOIN page ON parution.num_parution=page.num_parution
    JOIN comporte_image ON comporte_image.id_page=page.id_page
    JOIN image ON image.num_image=comporte_image.num_image
    WHERE titre_image LIKE '%Appolo%';
    ```
    renverra les numéros des parutions où figure une image dont le titre contient la chaîne 'Appolo'.
15. La requête `INSERT INTO image VALUES (2923,'Volcans du massif central','',400,400,1430);` insère dans la table `image` une image de numéro 2923, de 400 pixels de côté, pesant 1430 Ko et de titre 'Volcans du massif central', en laissant le champ `descritptif` vide.
16. La requête `INSERT INTO texte VALUES (2754,'Vulcania',"Parc d'attraction",250);` convient.
17. La requête `DELETE FROM texte WHERE num_texte=2034;` ne fait rien si la relation `comporte_texte` ne contient aucune référence à ce numéro de texte (sinon il y aurait une erreur de violation de contrainte de référence).
18. La requête `DELETE FROM comporte_texte WHERE num_texte=2034;` convient.