Contrôle final d'algorithmique/Java - INF220 **Nom :**

Les exercices sont traitables de manière indépendante, donc si vous bloquez sur un, passez au suivant, vous y reviendrez plus tard. **Le dernier exercice est volontairement plus difficile que les autres.**
Des temps indicatifs pour chaque exercice sont donnés entre parenthèses. Les pourcentages d'importance des exercices ne sont qu'indicatifs : selon vos résultats, il est possible que les dernières questions ne soient pas prises en compte pour la note, ou avec un coefficient plus faible.

Exercice 1 – Structures de données (15%)
Q1/ Reliez chaque ensemble de données (à gauche) à la structure de données (à droite) la plus adaptée pour stocker ces données, en imaginant que vous avez l'idée saugrenue de faire un logiciel pour stocker ces données. Ajoutez une phrase d'explication pour chaque lien. Chaque ensemble de données ne doit être relié qu'à une seule structure de données, et vice-versa.

a) Votre intégrale des épisodes de *Friends* rangés sur l'étagère dans l'ordre de diffusion aux USA (la série est terminée depuis 2004)

b) La liste des joueurs d'une partie de Monopoly

c) Les tee-shirts pliés rangés les uns au-dessus des autres dans votre placard

d) La liste de tous les fichiers et dossiers de votre disque dur

e) Votre collection de romans rangés par ordre alphabétique des auteurs sur l'étagère (vous en achetez et en prêtez souvent)

- arbre
- file
- liste
- pile
- tableau

Justification pour a) : _____

_____

Justification pour b) : _____

_____

Justification pour c) : _____

_____

Justification pour d) : _____

_____

Justification pour e) : _____

_____

Q2/ Un magasin veut utiliser une structure de données légère pour stocker, pour chaque produit alimentaire qu'il vend, les numéros de lots de produits qui sont dans ses rayons. Utilise-t-il une pile ou une file ? Justifiez !

Exercice 2 – Afficheur Leet speak (30%)

Q1/ Les geeks utilisent un codage des caractères habituels par des caractères ressemblants, en remplaçant par exemple S par 5 ou K par |<. Complétez les 20 trous (de la forme : _____) dans le code Java ci-dessous, afin qu'il permette d'afficher à l'écran la phrase : £'4£60 (' 357 7R0|° |=R415 !

On dispose des fonctions suivantes en Java :

- `System.out.println(a)` affiche la chaîne de caractères *a* puis retourne à la ligne
- `System.out.print(a)` affiche la chaîne de caractères *a*
- `a.length()` renvoie le nombre de caractères de la chaîne *a*
- `Caractere(a,i)` renvoie une chaîne de caractères qui contient le *i*-ième caractère de la chaîne *a* (la numérotation des caractères commence à 1)
- `ChainesEgales(a,b)` renvoie le booléen True si les chaînes *a* et *b* sont égales, le booléen False sinon (on ne peut pas utiliser simplement *a*==*b* pour tester l'égalité de deux chaînes de caractères *a* et *b*)

```
import java.io.*;

public class LeetSpeak{
   public static void ________(String[] arg){
     // Affichage en Leet speak de la phrase voulue :
     _____________("L'_________ C'EST TROP FRAIS !");
   }

   public static _________ afficheLeetSpeak(____________ chaine){
      for(____________;i<_____________;i_________){
         // Affichage de chaque caractere en Leet speak
         System____________(ConvertitCaractere(____________(chaine,_____)));
      }
   }

   public static _________ ConvertitCaractere(________ ___________________){
     _________ carac=caractere;
     // Conversion de chaque caractere en son equivalent Leet speak
     if (ChainesEgales(carac,"___")){carac="4";}
     if (ChainesEgales(carac,"C")){carac="(";}
     if (ChainesEgales(carac,"___")){carac="3";}
     if (ChainesEgales(carac,"F")){carac="|=";}
     if (ChainesEgales(carac,"G")){carac="6";}
     if (ChainesEgales(carac,"I")){carac="1";}
     if (ChainesEgales(carac,"L")){carac="£";}
     if (ChainesEgales(carac,"___")){carac="0";}
     if (ChainesEgales(carac,"___")){carac="|°";}
     if (ChainesEgales(carac,"S")){carac="5";}
     if (ChainesEgales(carac,"T")){carac="7";}
     return _________;
   }

     [… Ici sont situées les déclarations des fonctions Caractere et
     ChainesEgales …]
}
```

Q2/ Écrivez en Java une fonction **ecritUnCaractereParLigne** qui prend en entrée une chaîne de caractères *a* et l'affiche caractère par caractère, en retournant à la ligne après chaque caractère :

Exercice 3 – Minimum d'un tableau d'entiers (25%)

Rappels sur les tableaux d'entiers en Java
- `int[]` est le type Java des tableaux d'entiers
- `t[i]` est la *i*-ième case du tableau *t* (la case du début du tableau étant la case numéro 0)
- `t.length` est le nombre de cases du tableau *t*

On souhaite écrire en Java une fonction **minimumTableauAPartirDe** qui prend en entrée un tableau *tab* d'entiers, et un entier *i* et qui renvoie la valeur minimum du tableau à partir de la *i*-ième case (en incluant cette case, et en commençant la numérotation à 0).
Par exemple, **`minimumTableauAPartirDe`**`(new int[]{9,8,3,2,4,6,5,6,9},3)` renvoie 2.

Q1/ Combien renvoie **`minimumTableauAPartirDe`**`(new int[]{9,8,3,2,4,6,5,6,9},4)` ?

Q2/ Écrivez en Java une fonction **PremierSuperieurAuSecond** qui prend en entrée un tableau *tab* d'entiers et qui renvoie le booléen True si la première case de tab est strictement supérieure à la seconde, le booléen False sinon.

Q3/ En utilisant une boucle, écrivez en Java la fonction **`minimumTableauAPartirDe`**.

Q4/ En utilisant le principe que le minimum d'un tableau à partir de la *i*-ième case est soit la *i*-ième case (si elle est plus petite que toutes les cases qui suivent), soit le minimum du tableau à partir de la (*i*+1)-ième case (si ce minimum est plus petit que la *i*-ième case), écrivez en Java, sans utiliser de boucle, la fonction **`minimumTableauAPartirDe`**, grâce à la récursivité.

Exercice 4 – Copies de piles et files (15%)

Q1. Supposons qu'on a deux variables de type `Queue` *F* et *G*, telles que *G* est vide (elle vient d'être créée)
Écrivez une suite d'instructions qui permet, sans faire d'affectation directe (`G=F;`) de copier la file *F* à l'intérieur de la file *G* (c'est-à-dire transférer tous les éléments de *F* dans *G*)

Q2. Même question si *F* et *G* sont de type `Stack` (c'est-à-dire des piles d'entiers). Comme précédemment, *G* est vide.

On rappelle les fonctions sur les piles et les files :
- type file d'entiers : `Queue`
- créer une file d'entiers *F* : `F = creeFile();`
- enfiler un entier *a* à la fin de la file *F* : `enfile(F,a)` (ne renvoie rien)
- défiler l'entier *a* du début de la file *F* : `defile(F)` (renvoie l'entier *a*)
- tester si la file *F* est vide : `estVide(F)` (renvoie un booléen)
- type pile d'entiers : `Stack`
- créer une pile d'entiers *P* : `P = creePile();`
- empiler un entier *a* en haut de la pile *P* : `empile(P,a)` (ne renvoie rien)
- dépiler l'entier *a* en haut de la pile *P* : `depile(P)` (renvoie l'entier *a*)
- tester si la pile *P* est vide : `estVide(P)` (renvoie un booléen)

Exercice 5 – Élévation rapide à la puissance (15%)

Pour calculer a^b avec l'algorithme naïf de puissance, on effectue *b*-1 multiplications (à chaque fois, on multiplie par *a*). Il existe pourtant une façon astucieuse d'effectuer le calcul de a^b en faisant moins de multiplications.
L'idée de base de l'optimisation du calcul est la suivante : pour calculer a^4, une fois que j'ai calculé $a^2=a*a$, au lieu de calculer $a^3=a*a$, puis $a^4=a^3*a$, comme dans l'algorithme naïf, je peux plutôt calculer directement $a^4=a^2*a^2$ en utilisant la valeur de a^2 que je viens de calculer. Ainsi, je ne ferai au total que deux multiplications au lieu de trois.
Autre exemple : pour calculer a^{20}, je peux dire qu'il s'agit simplement de $a^{10}*a^{10}$ (une multiplication), et j'obtiens a^{10} en remarquant que c'est égal à a^5*a^5 (une autre multiplication), a^5 étant obtenu en remarquant qu'il est égal à $(a^2*a^2)*a$ (deux multiplications), a^2 étant lui même obtenu par l'opération $a*a$ (une multiplication). Soit au total 5 multiplications au lieu de 19.

Généralisez ces remarques en écrivant en Java un algorithme récursif rapide d'élévation à la puissance : il faudra distinguer le cas où *b* est pair de celui où *b* est impair.