Consolidation des bases de la programmation

TP 1 : Tableaux et boucles

CR à rédiger : un texte `.txt` avec les réponses aux questions. Une réponse **ne doit pas dépasser 2 lignes** sur une feuille A4 (taile de police : 12, marges classiques). Lors du contrôle TP, vous imprimez votre fichier et il sera noté.

1 Tableaux, boucles et adresses IP

Exercice 1.1 Ecrire un programme `Ruby` qui demande à l'utilisateur de saisir un nombre naturel n et qui affiche le tableau $[1, -2, 3, -4, 5, -6 \cdots \pm n]$. Attention : si n est pair, alors la dernière valeur doit être négative (la dernière valeur devrait être $-n$). Sinon, la dernière valeur devrait être n. Par exemple, si $n = 6$, le programme devrait afficher :

```
[1,-2,3,-4,5,-6]
```

Si $n = 7$, le programme devrait afficher

```
[1,-2,3,-4,5,-6,7]
```

Exercice 1.2 Modifier le programme `Java` ci-dessous pour résoudre l'exercice précédent en `Java`. La valeur de n doit saisie dans le programme (voir ligne 11 ci-dessous). C'est le seul programme à réaliser en Java au TP1.

```
1 import java.util.Arrays;
2 public class Exo1{
3     public static void plus5(int[] tab){
4         int n = tab.length;
5         for(int i=0;i<n;i++){
6             tab[i] = tab[i] + 5;
7         }
8     }
9     public static void main(String[] args){
10        int[] tab;
11        int n = 7;
12        tab = new int[n];
13        for(int i=0;i<n;i++){
14            if(i<3){
15                tab[i] = 7;
16            }else{
17                tab[i] = i+1;
18            }
19        }
20        plus5(tab);
21        System.out.println(Arrays.toString(tab));
22    }
23 }
```

Exercice 1.3 Écrire un programme Ruby qui demande à l'utilisateur de saisir une série de nombres. Lorsque l'utilisateur saisit 0, le programme affiche la moyenne des nombres déjà saisis.
Indication 1 : Voir la session `irb` ci-dessous.

```
irb(main):012:0> a=Array.new()
=> []
irb(main):013:0> a<<55
=> [55]
irb(main):014:0> a<<9
=> [55, 9]
irb(main):015:0> a<<88
=> [55, 9, 88]
irb(main):016:0> a.length()
=> 3
```

Indication 2 : Voir la session `irb` ci-dessous.

```
irb(main):028:0> nb=8
=> 8
irb(main):029:0> while (nb!=0) do nb=gets().to_i(); end
6
8
9
0
=> nil
```

Question 1	Est-ce qu'il aurait été possible de résoudre l'exercice 1 et 2 sans utiliser de tableau ?	☐
Question 2	Est-ce qu'il aurait été possible de résoudre l'exercice 3 sans utiliser de tableau ?	☐

Exercice 1.4 Écrire un programme qui demande à l'utilisateur de saisir une adresse IP. Le programme devrait vérifier si celle-ci est correcte et ensuite l'afficher en binaire (ex., `12323.to_s(2)` renvoie `"11000000100011"`). Il faut vérifier si l'adresse comporte bien 4 champs. Il faut vérifier que chaque champs représente un nombre entre 0 et 255. En plus, la première adresse IP qu'on considère comme correcte pour cet exercice est : 1.0.0.0. La dernière adresse correcte pour cet exercice est : 223.255.255.255 (à partir de 224.0.0.0, on rentre dans des adresses multicast qu'on ignore dans cet exercice).
Indication 1 : utiliser un tableau avec 4 cases. Vous pouvez l'obtenir grâce à la routine `split`, comme illustré dans l'exemple `irb` ci-dessous.

```
irb(main):001:0> "10.0.0.5".split(".")
=> ["10", "0", "0", "5"]
irb(main):002:0>
```

2 Tableaux, matrices, arithmétique

Exercice 2.1 Écrire un programme Ruby qui demande à l'utilisateur de saisir un nombre naturel n et qui affiche la table de multiplication $n \times n$. Par exemple, si $n = 4$, le programme pourrait afficher :

```
[1,2,3,4]
[2,4,6,8]
[3,6,9,12]
[4,8,12,16]
```

Exercice 2.2 Écrire un programme `Ruby` qui demande à l'utilisateur de saisir un nombre naturel n et qui affiche les valeurs :

$$1!,\quad 2!,\quad 3!\quad ,\ldots n!$$

Par exemple, si $n = 5$, le programme devrait afficher :

$$1, \quad 2, \quad 6, \quad 24, \quad 120$$

Dans une deuxième étape, calculer la somme ci-dessous (*attention :* 0! vaut 1).

$$S = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

Finalement, afficher $T = S - $`Math::E`

Question 3 Quelle valeur de T avez vous trouvé ? □

Question 4 Quelle est le nombre de multiplications nécessaires pour réaliser les taches ci-dessus ? Est-ce qu'il toujours nécessaire d'utiliser un tableau ? □

Exercice 2.3 Implémenter un programme qui permet d'afficher tous les nombres premiers inférieurs à une valeur n saisi par l'utilisateur. Attention : (i) il faut faire appel au crible Ératosthène décrit ci-après, (ii) il ne faut pas utiliser de multiplication.

Le crible d'Ératosthène est un procédé qui permet de trouver tous les nombres premiers inférieurs à un certain entier naturel donné n. L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier. En supprimant tous les multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier, et qui sont donc les nombres premiers. On commence par rayer les multiples de $p = 2$. Pour illustration, pour "rayer" les multiples de $p = 2$, on pourrait utiliser un code comme :

```
p=2
multiple_p=p+p
while (multiple_p<n) do
    tab[multiple_p]=""
    multiple_p=multiple_p+p
end
```

Ensuite, on raye les multiples du plus petit entier restant, e.g., après $p = 2$, on passe à $p = 3$, puis à $p = 5$, etc.

Exercice 2.4 Implémenter une routine `pgdc(`a, b`)` qui permet de renvoyer (sans afficher) le Plus Grand Diviseur Commun (PGDC) de a et b. Exemple : `pgdc(6,15)` devrait retourner 3.

Indication : penser à l'algorithme d'Euclid décrit en pseudo-code par :

```
while not (a == b)
    if a > b
       a = a - b
    else
       b = b - a
    end if
end while
return a
```

Exercice 2.5 Implémenter un programme qui permet de construire une matrice M de taille $n \times n$ avec n introduit par l'utilisateur. M_{ij} vaut 1 si $pgdc(i, j)$ renvoie 1. M_{ij} vaut 0 si $pgdc(i, j)$ renvoie une valeur supérieure à 1 ; vous observez que $M_{ij} = 1$ uniquement si i et j sont premiers entre eux (i est premier avec j si

$pgdc(i, j) = 1$). Afficher cette matrice.

Exercice 2.6 Soit un entier n saisit par l'utilisateur. Écrire un programme qui affiche le nombre de valeurs $i \in [1..n]$ premiers avec n.

Question 5 Quelle valeur devrait le programme afficher si n est premier ? □

3 Fichiers d'entrée et de sortie

Exercice 3.1 Écrire un programme qui permet de lire une matrice $n \times n$ à partir d'un fichier `in.txt` et l'écrire dans un fichier `out.txt`. Le fichier d'entrée contient une ligne avec une valeur de n, suivie par n lignes (chaque ligne avec n valeurs). Par exemple,

```
3
6 7 9
3 2 3
9 3 0
```

Le fichier de sortie contient juste les n lignes avec des éléments séparés par ",". Par exemple :

```
6,7,9
3,2,3
9,3,0
```

Pour lire la valeur de n, il est possible d'appeler `f.gets().to_i` sur un objet f de classe `File`. Cet objet est construit avec `f=File.open("in.txt", "r")`. Pour lire les valeurs d'une ligne du fichier dans un tableau, on utilise `f.gets().split()`.

Exercice 3.2 Écrire un programme qui permet de lire les coordonnées (x, y) de n points (villes) à partir d'un fichier `in.txt`. La valeur de n est indiquée par la première ligne. Les autres n lignes contient n coordonnées (x, y). L'exemple ci-dessous indique 5 points. La premier point est situé aux coordonnées $(1, 1)$; le deuxième aux coordonnées $(4, 5)$, etc.

```
5
1 1
4 5
8 9.2
2.6 4.2
1.5 2.2
```

Construire une matrice d de distances $n \times n$. L'élément d_{ij} représente la distance entre le point i et le point j. Par exemple, la distance entre le point 0 (coordonnées $x = 1$ et $y = 1$) et le point 1 (coordonnées $x = 4$ et $y = 5$) est $\sqrt{3^2 + 4^2} = 5$. (pour quoi ?). **Écrire cette matrice** dans un fichier `out.txt` (le même format que pour l'exercice précédent).

Exercice 3.3 Notons les points (villes) par $0, 1, 2 \ldots n-1$. Soit le trajet(circuit) $0 \to 2 \to 1 \to 3 \cdots \to n-1 \to 0$. Construire un tableau $t = [0,\ 2,\ 1,\ 3,\ 4,\ 5,\ \ldots n-1,\ 0]$ associé à ce trajet. Écrire une routine `distTrajet(t)` qui permet de calculer la distance totale du trajet t ($t_0 \to t_1 \to t_2 \cdots \to t_{n-1}$). Comme indiqué dans le cours, pour aller de $t[0]$ à $t[1]$, il faut parcourir une distance $d_{t[0],t[1]}$; pour continuer, de $t[1]$ à $t[2]$, il y a une distance de $d_{t[1],t[2]}$, etc.