TP n°4

Rappels de cours

1 Récursivité simple

Une fonction (ou une procédure) récursive est une fonction qui s'appelle elle-même. Ainsi, la fonction factorielle peut être définie de manière plus concise à l'aide d'une fonction récursive :

```
import fr.jussieu.script.Deug;

class Fact{

    static int fact_imperative(int n){
        int res = 1;
        for (int i = 1; i <= n ; i++)
          res = res * i;
        return res;
    }

    static int fact_recursive(int n){
        if (n == 0)
          return 1;
        else
          return n * fact_recursive(n-1);
    }

    static int fact_recursive_terminale(int n, int a){
        if (n <= 1)
          return a;
        else
          return fact_recursive_terminale(n-1,n*a);
    }

    public static void main (String[] args){
        int p;
        p = Deug.readInt();
        Deug.println(fact_imperative(p) + "   " + fact_recursive(p) + " " +
                     fact_recursive_terminale(p,1));
    }
}
```

Une fonction récursive f est dite terminale lorsque tout appel récursif est de la forme return f(...) ; ; autrement dit, la valeur retournée est directement la valeur obtenue par un appel récursif, sans qu'il n'y ait aucune opération sur cette valeur, c'est le cas de la fonction `fact_recursive_terminale`. Ainsi, dans le cas d'une fonction récursive terminale, le dépilement des valeurs de retour est direct, ceci aboutit à une version plus optimisée de la fonction.

NB : la fonction mathématique factorielle $n \mapsto n!$ n'est ni récursive, ni itérative, c'est l'algorithme utilisé pour la calculer qui est l'un ou l'autre.

Il faut faire attention à ce que la fonction ne boucle pas sur la même valeur, auquel cas le programme ne s'arrêterait jamais, comme dans cet exemple :

```
import fr.jussieu.script.Deug;

class Stupide{
    static int boucle(int n){
        return boucle(n);
    }



    public static void main (String[] args){
        int p;
        p = boucle(0);
        Deug.println("ce message n'arrivera jamais");
    }
}
```

Pour éviter les appels infinis :

- l'appel récursif doit toujours être fait avec un paramètre de valeur différente que l'appel initial, dans la plupart des cas ce paramètre est plus petit
- il doit toujours y avoir un cas d'arrêt, i.e. sans appel récursif.

Un algorithme récursif est plus lent qu'un algorithme itératif car il y a la gestion des appels de fonctions (empilement et dépilement du contexte).

2 Récursivité croisée

On parle de récursivité *croisée* lorsque deux fonctions s'appellent l'une l'autre récursivement.