TD 6 – Itération/Récursion - Appels de fonction

Concepts informatiques (CI2)

2011–2012

1 Exercice

On définit le type `Pile` suivant :

```
class Pile {
    int sommet;
    int []valeurs;
    public Pile() {
        sommet = 0;
        valeurs = new int[100];
    }
    public int lire(int n) {
     return valeurs[sommet-n-1];
    }
    public void ecrire(int n,int valeur) {
        valeurs[sommet-n-1] = valeur;
    }
    public int depile() {
        return valeurs[--sommet];
    }
    public void empile(int valeur) {
        valeurs[sommet++] = valeur;
    }
}
```

Soit le programme suivant :

```
public static void main(String []a) {
    Pile pile = new Pile();
    boolean exit = false;
    int instruction;
    int registre;
    int [] memoire = new int[3];
    instruction = 1;
    while (!exit) {
        switch (instruction) {
        case 1:
            memoire[0] = 0;
            instruction++;
            break;
        case 2:
            memoire[1] = 0;
            instruction++;
            break;
        case 3:
```

```
            if (memoire[0]==10) instruction = 8;
            else instruction++;
            break;
        case 4:
            instruction++;
            pile.empile(instruction);
            pile.empile(memoire[0]);
            instruction = 9;
            break;
        case 5:
            memoire[1] += pile.depile();
            instruction++;
            break;
        case 6:
            memoire[0]++;
            instruction++;
            break;
        case 7:
            instruction = 3;
            break;
        case 8:
            exit = true;
            instruction++;
            break;
        case 9:
            pile.ecrire(0,3*pile.lire(0));
            instruction++;
            break;
        case 10:
            registre = pile.depile();
            instruction = pile.depile();
            pile.empile(registre);
            break;
        }
    }
}
```

1. Exécuter ce programme en notant soigneusement l'évolution des variables.
2. Que calcule ce programme ? Comment le fait-il ?
3. Écrire un programme naturel équivalent.
4. Faites l'opération inverse avec le programme suivant :

```
public static int f(int n) {
    int res;
    res=g(n);
    return res;
}
public static int g(int n) {
    int res;
    res=2+n;
    return res;
}
public static void main(String[] a) {
    int i;
    int tmp;
    tmp=1;
```

```
    for(i=0;i<10;i++){
        tmp*=f(i);
    }
    System.out.println(tmp);
}
```

2 Exercice

On parle de récursivité *croisée* lorsque deux fonctions s'appellent l'une l'autre récursivement.
Les fonctions suivantes sont censées donner la parité d'un nombre entier : cela sera-t-il le cas pour toutes les valeurs entières positives ? Si ce n'est pas le cas, proposer une correction.

```
class PairImpair{

    static boolean pair (int n){
        if (n==0)
            return true;
        else
            return impair(n-1);
    }
    static boolean impair (int n){
        if (n==1)
            return true;
        else
            return pair(n-1);
    }

    public static void main (String[] args) {
        Scanner s = new Scanner(System.in);
        int p = s.nextInt();
        System.out.println("pair? "+ pair(p) + " impair? " + impair(p));
    }
}
```

3 Exercice

Dans cette exercice, toutes les méthodes demandées doivent être *récursives*.

1. Écrire une fonction `String repete( int n, String s)` renvoyant la chaîne de caractères s répétée n fois : `repete(3, "bla")` donne `"blablabla"`.
2. Écrire une fonction `void pyramide( int n, String s)` qui écrit sur la première ligne, 1 fois la chaîne `s`, sur la deuxième, 2 fois la chaîne `s`, et ainsi de suite jusqu'à la dernière ligne, où il y aura `n` fois la chaîne `s`. Ainsi `pyramide(5, "bla");` donnera

   ```
   bla
   blabla
   blablabla
   blablablabla
   blablablablabla
   ```

3. Quand on lance `pyramide(n,s)`, combien y a-t-il d'appels à `pyramide` ? Combien y a-t-il d'appels à `repete` ? Dessiner l'arbre des appels pour `n` = 3.

4. comment faire pour obtenir la pyramide «inverse» de la précédente ? Toujours en utilisant uniquement des appels récursifs...

4 Exercice

Une *sous-suite* d'un mot u est un mot w obtenu à partir de u en effaçant des lettres et en respectant l'ordre. Par exemple, ara est une sous-suite des mots $quadratique$, $marchandage$ et $alcatraz$, mais n'est pas une sous-suite de raa.
Soient deux mots u et v. On veut implémenter une méthode qui retourne la longueur maximale d'une sous-suite commune à u et v.

– Dans un premier temps, on cherche à en donner une version récursive n. On note $LSSM(i, j)$ la longueur maximale d'une sous-suite commune au mot formé des i premières lettres de u avec celui formé des j premières lettres de v.
  Que vaut $LSSM(i, 0)$? $LSSM(0, j)$?
  On note u_i la i-ème lettre du mot u. Supposons que $u_i = v_j$.
  Exprimer $LSSM(i, j)$ en fonction de $LSSM(i - 1, j - 1)$.
  Supposons maintenant que $u_i \neq v_j$.
  Exprimer $LSSM(i, j)$ en fonction de $LSSM(i - 1, j)$ et de $LSSM(i, j - 1)$.
  En déduire une méthode récursive naïve pour le problème. Pour $u = abac$ et $v = cbc$, dessiner l'arbre des appels récursifs et donner les états successifs de la pile.
– Implémenter une version itérative puis une version récursive *dynamique* de cette méthode, en mémorisant les valeurs des sous-problèmes dans un tableau à deux dimensions. Construire le tableau pour les deux versions lorsque $u = abac$ et $v = cbc$.