Partiel du 1/04/09 - Concepts Informatiques (CI2)

1 Évolution de la mémoire

Décrivez, par des schémas similaires à ceux vus en cours et en TD, l'évolution de la pile et du tas lors de l'exécution du programme suivant, à chaque ligne de code, y compris lors de l'appel des méthodes `m1` et `m2`.

La solution de cet exercice n'est pas donnée.

2 Piles

On se donne trois piles `p1`, `p2` et `p3`. La pile `p1` contient une suite de nombres entiers positifs.

On suppose que vous avez déjà écrit une classe `Pile` qui fonctionne bien, contenant les méthodes `estVide, sommet, empiler, depiler`.

1. Écrire un algorithme pour déplacer les entiers de `p1` dans `p2`, en inversant leur ordre.
   Il suffit de depiler de p1 pour empiler dans p2 :

   ```
   while(!p1.empty()){p2.push(p1.pop());}
   ```

2. Écrire un algorithme pour déplacer les entiers de `p1` dans `p2`, en conservant leur ordre.
   On se sert de p3 comme pile "intermediaire", ainsi on inverse l'ordre deux fois.

   ```
   while(!p1.empty()){p3.push(p1.pop());
   while(!p3.empty()){p2.push(p3.pop());}
   ```

3. Écrire un algorithme pour déplacer les entiers de `p1` dans `p2` de façon à avoir dans p2 tous les nombres pairs au-dessous des nombres impairs.
   Les pairs vont dans p2, les impairs vont (temporairement) dans p3 ; ensuite on met les impairs stockés en p3 au dessus des pairs déjà en p2.

   ```
   while(!p1.empty()){
       int elem = (Integer)p1.pop();
       if (elem%2 ==0) p2.push(elem);
       else p3.push(elem);
   }
   while(!p3.empty()){p2.push(p3.pop());
   ```

4. Écrire un algorithme pour copier dans `p2` les nombres pairs contenus dans `p1`. Le contenu de `p1` après execution de l'algorithme doit être identique à celui avant exécution. Les nombres pairs doivent être dans `p2` dans l'ordre où ils apparaissent dans `p1`.
   On transfere tout dans p3 en inversant l'ordre. Ensuite, les éléments pairs vont dans p2 ; on inverse a nouveau l'ordre, donc... Tandis que tous les éléments retournent dans p1 ; on inverse à nouveau, donc...

```
    while(!p1.empty()){p3.push(p1.pop());
    while(!p3.empty()){
        int elem = (Integer)p3.pop();
        if (elem%2 ==0) p2.push(elem);
        p1.push(elem);
    }
```

3 Compilation/Décompilation

1. Exécuter pas à pas ce programme en notant dans un tableau à quatre colonnes l'évolution des variables instruction, registre et memoire[0] et memoire[1]. Chaque ligne comportera le contenu des variables après la n-ième étape.

```
public static void main (String[] args){
    boolean exit = false;
    int instruction = 1;
    int registre = 0;
    int[] memoire = new int[2];
    while(!exit){
        switch(instruction){
        case 1:
            memoire[1] = 1;
            instruction++; break;
        case 2:
            memoire[0] = 0;
            instruction++; break;
        case 3:
            if(memoire[0] < 5) instruction++;
            else instruction = 8;
            break;
        case 4:
            registre = memoire[1] * 5;
            instruction++; break;
        case 5:
            memoire[1] = registre + 1;
            instruction++; break;
        case 6:
            memoire[0]++;
            instruction++; break;
        case 7:
            instruction = 3;
            break;
```

```
            case 8:
                System.out.println("Resultat = " + memoire[1]);
                exit = true;
                break;
            }
        }
}
```

ins	reg	m0	m1
1	0	0	0
2	0	0	1
3	0	0	1
4	0	0	1
5	5	0	1
6	5	0	6
7	5	1	6
3	5	1	6
4	5	1	6
5	30	1	6
6	30	1	31
7	30	2	31
3	30	2	31
4	30	2	31
5	155	2	31
6	155	2	156
7	155	3	156
3	155	3	156
4	155	3	156
5	780	3	156
6	780	3	781
7	780	4	781
3	780	4	781
4	780	4	781
5	3905	4	781
6	3905	4	3906
7	3905	5	3906
3	3905	5	3906
8	3905	5	3906

2. Que calcule ce programme ?

Le programme calcule l'élément u_5 de la suite définie par $u_0 = 1$ et $u_{i+1} = 5 * u_i + 1$. En effet `mem[0]` fait les fonctions d'un compteur : initialisé à 0, il est incrémenté à chaque itération de la boucle 3-7, qui s'arrête quand `mem[0]` est égal à 5. Par ailleurs, `mem[1]` est initilisé à 1 et à chaque itération de la boucle `registre` prend la valeur de 5*`mem[1]` ; cette valeur,

incrémentée d'une unité, est ensuite transférée dans `mem[1]` avant l'itération suivante.

3. "Décompiler" ce programme, c'est à dire le réécrire dans un style plus naturel.

```
public class Exo3_3 {

public static void main (String[] args){
int res=1;
int registre=0;
for (int i=0; i<5; i++){
   registre = res*5;
   res=registre+1;
}
System.out.println("resultat = "+res);
}
}
```

4. Soit la suite u_n définie par

$$u_0 = 4$$
$$u_{n+1} = n * u_n - n$$

a. Ecrire une fonction récursive calculant la n-ième valeur de la suite

```
static int u(int n){
  if (n==0) return 4;
  return ((n-1)*u(n-1)-(n-1));
}
```

b. Ecrire une version itérative calculant la 10ème valeur de la suite

```
static int u(int n){
  int res = 4;
  for (int i=0; i<n; i++){
     res=i*res-i;
  }
  return(res);
}
```

qu'on appellera en passant la valeur 10 pour le paramètre n.

c. "Compiler" cette version itérative dans le même style que le programme au début de l'exercice.

```
public static void main (String[] args){
    boolean exit = false;
    int instruction = 1;
```

```
    int registre = 0;
    int[] memoire = new int[2];
    while(!exit){
        switch(instruction){
        case 1:
            memoire[1] = 4;
            instruction++; break;
        case 2:
            memoire[0] = 0;
            instruction++; break;
        case 3:
            if(memoire[0] < 10) instruction++;
            else instruction = 8;
            break;
        case 4:
            registre = memoire[1] * memoire[0];
            instruction++; break;
        case 5:
            memoire[1] = registre - memoire[0];
            instruction++; break;
        case 6:
            memoire[0]++;
            instruction++; break;
        case 7:
            instruction = 3;
            break;
        case 8:
            System.out.println("Resultat = " + memoire[1]);
            exit = true;
            break;
        }
    }
}
```

4 Cette Débordante Fonction d'Ackermann

On rappelle la défintion de la fonction d'Ackermann :

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m,n-1)) & \text{si } m > 0 \text{ et } n > 0 \end{cases}$$

Ecrire un programme qui pour un couple d'entiers (m, n), affiche une matrice M où $M[i, j]$ contiendra le nombre de fois que l'appel `Acker(i,j)` a été effectué pour calculer $A(m, n)$. On note qu'au moins l'une des dimensions de la matrice ne peut pas être prévue à l'avance, il faudra donc gérer cet aspect de l'algorithme de manière appropriée.

```
public class AckerCompte {

static void Incremente (int [][]T, int m, int n){
  if (T[m].length <= n) {
    int [] tmp = new int [n+1];
    for (int i=0; i<T[m].length; i++) tmp[i]=T[m][i];
    T[m]=tmp;
  }
  T[m][n]++;
}

static int A(int [][]T, int m, int n){
  Incremente(T, m, n);
  if (m==0) return(n+1);
  else if (n==0) return A(T, m-1, 1);
       else return(A(T, m-1,A(T, m,n-1)));
}

public static void main(String[] args){
  int m = Integer.parseInt(args[0]);
  int n = Integer.parseInt(args[1]);
  int res;
  int [][] T =  new int [m+1][n+1];
  res = A(T,m,n);
  for (int i=0; i<=m; i++) {
    for(int j=0; j<T[i].length; j++)
       System.out.print (T[i][j] + " ");
    System.out.println(" ");
  }
}
}
```