Université Paris 7 - Denis Diderot
CI2 — Partiel 3 mars 2007 — Correction

Sur les copies d'examen, même si ce n'est pas précisé il vaut mieux, en général, commencer chaque exercice sur une nouvelle page. Et surtout écrire comme un titre (en gros, souligné, encadré, en couleur, ...) le numéro de l'exercice.

Exercice 1 — *Appels par valeurs, appels par références*

L'exécution commence par la fonction `main`.

```
int x=5 ;
```

On crée, dans le bloc d'activation de la fonction `main`, une variable `x`, contenant l'entier 5.

```
System.out.println("dans main : x="+x) ;
```

```
dans main : x=5
```

```
f1(x) ;
```

On appelle la fonction `f1` sur l'entier contenu dans `x`, soit 5.

```
static int f1(int x){
```

On crée donc un bloc d'activation pour `f1`, contenant une variable `x` (qui n'a rien à voir avec celle de `main`), qui contient 5.

```
System.out.println("dans f1 : x="+x) ;
```

```
dans f1 : x=5
```

```
x= x+1 ;
```

La variable `x` du bloc d'activation de `f1` (et non pas de celui de `main`) est incrémentée.

```
System.out.println("dans f1 : x="+x) ;
```

```
dans f1 : x=6
```

Retour dans `main`. La variable `x` de `main` contient toujours 5.

```
System.out.println("dans main : x="+x) ;
```

```
dans main : x=5
```

```
IntRef xRef= new IntRef (x) ;
```

On crée un objet de type `IntRef`, dont le champ `valeur` contient 5. On entre l'adresse de cet objet dans la variable `xRef` de la fonction `main`.

```
System.out.println("dans main : xRef.valeur="+xRef.valeur) ;
```

```
dans main : xRef.valeur=5
```

```
f1(xRef.valeur) ;
```

On appelle `f1`, sur l'entier contenu dans le champ `valeur` de l'objet pointé par la variable `xRef`. Donc on crée un bloc d'activation contenant une variable `x` contenant 5. D'où :

```
dans f1 : x=5
dans f1 : x=6
```

Et l'on retourne dans `main`, où la variable `xRef` pointe toujours vers le même objet, qui n'a pas été modifié.

```
System.out.println("dans main : xRef.valeur="+xRef.valeur) ;
```

```
dans main : xRef.valeur=5
```

f2(xRef) ;

On appelle la fonction `f2`, sur l'objet pointé par `xRef`.

static int f2(IntRef xRef){

On crée un bloc d'activation pour `f2`, contenant une variable `xRef`, qui contient l'adresse de l'objet créé dans `main`.

xRef.valeur= xRef.valeur+1 ;

On incrémente le champ `valeur` de cet objet.

return xRef.valeur ;

On le renvoie en résultat. Retour dans `main`, qui jette ce résultat. Mais comme la variable `xRef` de `f2` pointait vers le même objet que la variable `xRef` de `main`, `xRef.valeur` vaut maintenant 6.

System.out.println("dans main : xRef.valeur="+xRef.valeur) ;

```
dans main : xRef.valeur=6
```

xRef.valeur= f2(xRef) ;

De même, `f2` incrémente le champ valeur de l'objet pointé par `xRef`, qui contient donc 7. `f2` renvoie cette valeur en résultat, et on la réentre dans le champ valeur de l'objet (ce qui est inutile).

System.out.println("dans main : xRef.valeur="+xRef.valeur) ;

```
dans main : xRef.valeur=7
```

x= f2(xRef) ;

Ici, c'est dans `x` que l'on entre le résultat de `f2`. On a donc désormais 8 dans `x` et `xRef.valeur`.

System.out.println("dans main : xRef.valeur="+xRef.valeur) ;
System.out.println("dans main : x="+x) ;

```
dans main : xRef.valeur=8
dans main : x=8
```

int[] T= **new** int[2] ;
T[0]= 2 ;
T[1]= 4 ;

On crée un tableau, dont on entre l'adresse dans la variable `T`. On entre 2 dans la première case, et 4 dans la seconde.

f1(T[0]) ;

On appelle `f1` sur l'entier contenu dans la première case du tableau, soit 2. C'est-à-dire que l'on crée un bloc d'activation contenant une variable `x` qui contient 2. Ce bloc est disjoint du tableau. `f1` incrémente cette variable `x` dans son bloc, mais ne touche pas au tableau puisqu'elle ne connaît pas son adresse.

```
dans f1 : x=2
dans f1 : x=3
```

Retour dans `main`, où le tableau pointé par `T` n'a pas été modifié.

System.out.println("dans main : T[0]="+T[0]) ;

```
dans main : T[0]=2
```

f3(T,T[0],T[1]) ;

On appelle `f3`, avec, comme arguments l'adresse du tableau, et les entiers contenus dans sa première et sa deuxième case.

static void f3(int[] T,int x, int y){

On crée donc un bloc d'activation pour `f3`, contenant une variable `T` pointant vers le tableau créé dans `main`, et deux variables `x` et `y` contenant respectivement 2 et 4.

T[1]=x ;
T[0]=y ;

La première case du tableau contient désormais 4 et la seconde 2.

System.out.println("dans f3 : T[0]="+T[0]) ;

`dans f3 : T[0]=4`

Retour dans `main`. Le contenu du tableau pointé par `T` a été modifié par `f3`, vu que c'est l'adresse de ce tableau qui a été passée à `f3`.

```
System.out.println("dans main : T[0]="+T[0]) ;
```

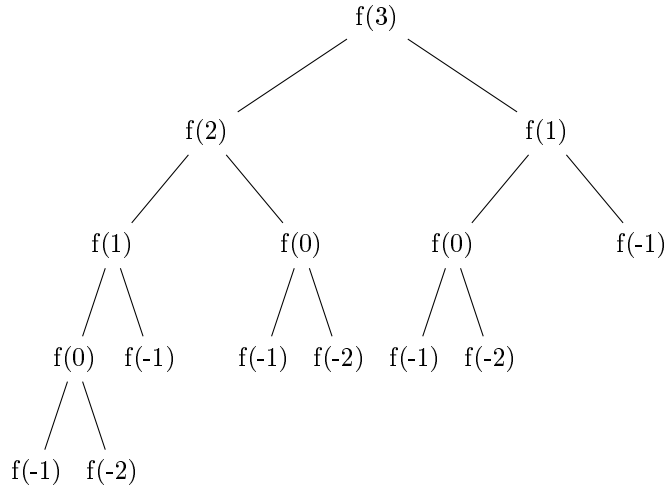
`dans main : T[0]=4`

Exercice 2 — ***Fonction récursive***

1. Version récursive naïve :

```
public static int naif(int x) {
  if (x<0) return 1 ; else return 3*naif(x-1)-naif(x-2) ;
  }
```

2. Arbre des appels :



3. Comme le code évalue en premier l'opérande de gauche de la soustraction (ce qui n'est pas obligatoirement le cas avec tous les langages de programmation), la suite des appels est : f(3), f(2), f(1), f(0), f(-1), f(-2), f(-1), f(0), f(-1), f(-2), f(1), f(0), f(-1), f(-2), f(-1)

4. Version itérative avec tableau :

```
public static int avectableau(int x) {
  if (x<0) return 1 ;
  int T[]=new int[x+3] ; // Ne doit pas être exécuté si par exemple x vaut -5
  T[0]=1 ; T[1]=1 ;
  for (int i=0 ;i<=x ;i++) T[i+2]=3*T[i+1]-T[i] ;
  return T[x+2] ;
  }
```

On stocke $f(i)$ dans la $i+2^{\text{ème}}$ case du tableau, les cases 0 et 1 contenant $f(-2)$ et $f(-1)$.

5.

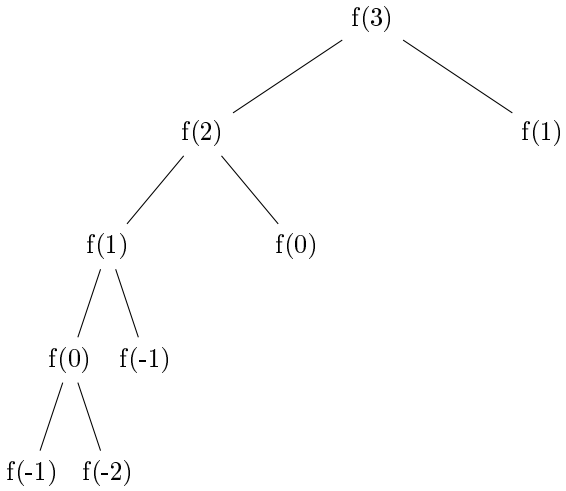
1	1	2	5	13	34

6. Version « programmation dynamique » :

```
public static int dyn_aux(int x,int T[],boolean deja_calcule[]) {
  if (x<0) return 1 ;
  if (deja_calcule[x]) return T[x] ;
  T[x]=3*dyn_aux(x-1,T,deja_calcule)-dyn_aux(x-2,T,deja_calcule) ;
  deja_calcule[x]=true ;
  return T[x] ;
  }
public static int dyn(int x) {
  if (x<0) return 1 ;
  int T[]=new int[x+1] ;
  boolean deja_calcule[]=new boolean[x+1] ;
```

```
    for (int i=0;i<=x+1;i++) deja_calcule[i]=false;
    return dyn_aux(x,T,deja_calcule);
  }
```

7. Le tableau `T` est rempli dans l'ordre `T[0], T[1], T[2]`... Ce qui ressemble au cas itératif. L'arbre des appels récursifs ressemble à celui de la question 2, après élagage :



L'ordre des appels est donc f(3), f(2), f(1), f(0), f(-1), f(-2), f(-1), f(0), f(1)

8. La version naïve va être catastrophiquement plus lente que les autres, dès que x va être un peu grand, car elle refait de nombreuses fois les mêmes calculs (de l'ordre de 2^x appels récursifs...). Ensuite, la version itérative a des chances d'être un peu meilleure que la version dynamique, puisqu'elle ne fait pas d'appels récursifs (on perd du temps à créer puis détruire les blocs d'activations).

Exercice 3 — *Empilement de boîtes*

1. On obtient 16 jetons à la fin :

			Bleue 1				
		Bleue 3	Rouge 3	Rouge 4			
	Bleue 5	Rouge 5	Rouge 5	Rouge 5	Rouge 9		
Bleue 7	Rouge 7	Rouge 7	Rouge 7	Rouge 7	Rouge 7	Rouge 16	16

2. – $f(0) = 0$
   – $f(1) = 1$
   – pour $n > 1$: $f(n) = f(n-2) + n$
3.
```
static int frec(int n) {
    if (n==0||n==1) return n;
    return n+frec(n-2);
}
```

4.
```
static int fit(int n) {
    int r=0;
    for (int i=n;i>0;i=i-2) r=r+i;
    return r;
}
```