Université Paris Diderot – Paris 7 — Concepts informatiques (CI2)
Examen du 17 mai 2011 — Durée : 2 heures 30 minutes
Documents, calculettes, ordinateurs et téléphones non autorisés.

Exercice 1

Après les définitions Java suivantes :

```
class N { public int v; }
public static void f1(int a,int b,int c,int d) {
   d = b * b - 4 * a * c;
}
public static void f2(int a,int b,int c,int d) {
   b *= b; d = b - 4 * a * c;
}
public static void f3(int []t) {
   t[3] = t[1] * t[1] - 4 * t[0] * t[2];
}
public static void f4(int []t) {
   t[1] *= t[1]; t[3] = t[1] - 4 * t[0] * t[2];
}
public static void f5(N na,N nb,N nc,N nd) {
   nd.v = nb.v * nb.v - 4 * na.v * nc.v;
}
public static void main(String[] args){
  int a,b,c,d;
  int [] t = new int[4];
  N na = new N(); N nb = new N(); N nc = new N(); N nd = new N();
      .........
```

Que produisent les séquences suivantes :

1.1.
```
a = 1; b = 2; c = 3; d = 10; f1(d, c, b, a);
System.out.println(a + " " + b + " " + c + " " + d);
```

1.2.
```
a = 1; b = 2; c = 3; d = 10; f2(a, b, c, d);
System.out.println(a + " " + b + " " + c + " " + d);
```

1.3.
```
t[0] = 1; t[1] = 2; t[2] = 3; t[3] = 10; f3(t);
System.out.println(t[0] + " " + t[1] + " " + t[2] + " " + t[3]);
```

1.4.
```
t[0] = 1; t[1] = 2; t[2] = 3; t[3] = 10; f4(t);
System.out.println(t[0] + " " + t[1] + " " + t[2] + " " + t[3]);
```

1.5.
```
na.v = 1; nb.v = 2; nc.v = 3; nd.v = 10; f5(na, nb, nc, nd);
System.out.println(na.v + " " + nb.v + " " + nc.v + " " + nd.v);
```

Exercice 2

On rappelle qu'en C++ la notation `int &x` utilisée dans l'en-tête d'une fonction, indique que le paramètre `x` est transmis par référence et non par valeur.

Après les définitions suivantes

```
void g1(int &a,int &b, int &c, int &d) {
   c += d; b += c; a += b;
}
void g2(int &a, int b, int &c, int d) {
   a += b; b += c; c += d; d += a;
}
```

2.2. Quelles sont le valeurs de `a`, `b`, `c` et `d` après exécution de :

```
a = 1; b = 2; c = 3; d = 4; g1(a, b, c, d);
```

2.2. Quelles sont le valeurs de `a`, `b` et `c` après exécution de :

```
a = 1; b = 2; c = 3; g2(a, a, b, c);
```

Exercice 3

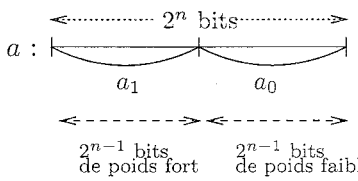
On considère les définitions de fonctions suivantes :

```
static void g(int m, int n, boolean x){
   if(x) {
       System.out.println(m + n); if(m > 0) g(m - 4, n + 1, x);
       }
   else {
       f(m - 3, n + 1); g(n + 3, m - 2, !x);
       }
}
static void f(int m, int n){
    if(m > n) {
       g(m - 2, n, false); f(m - 2, n);
       }
    else
       g(n - m, m + 2, true);
}
}
```

Décrire sous forme d'arbre la suite des appels induits par l'appel initial `g(9, 4, false)` et indiquer ce qui est affiché.

Exercice 4

L'objet de cet exercice est d'écrire sous forme récursive l'algorithme de Karatsuba permettant de multiplier deux nombres, plus rapidement que par la multiplication traditionnelle, particulièrement pour les grands nombres. Ici nous considérerons des nombres écrits en base deux et dont la représentation possède un nombre de bits égal à une puissance de 2 (donc de la forme 2^n, comme par exemple $32 = 2^5$).
Un tel nombre a peut être découpé en deux parties selon le schéma suivant :



Par exemple, pour $a = \overbrace{1100010010001101}$

$n = 16 = 2^4$, $a_1 = 11000100$, $a_0 = 10001101$

a_0 et a_1 de longueur $8 = 2^3$

et on a alors : $a = a_1 \times 2^{n-1} + a_0$.
Étant donnés deux nombres a et b de 2^n bits, le calcul de $p = a \times b$ peut être effectuée comme suit :

1. calculer $x = a_0 \times b_0$, $y = a_1 \times b_1$ et $z = \mid a_1 - a_0 \mid \times \mid b_1 - b_0 \mid$;
2. calculer $p = y \times 2^n + (x + y - signe(a_1 - a_0) \times signe(b_1 - b_0) \times z) \times 2^{n-1} + x$

On supposera que les fonctions suivantes sont fournies :

- `signe(int a)` : qui renvoie 1 ou -1 selon que $a \geq 0$ ou $a < 0$
- `abs(int a)` : qui renvoie la valeur absolue de a
- `puis2(int k)` : qui renvoie 2^k

Écrire une fonction `mult(int a, int b, int n)` récursive qui, étant donnés deux nombres a et b de n chiffres, renvoie la valeur de $a \times b$, en ne faisant que des multiplications de chiffres, des divisions (`/`) et des modulo (`%`) par des puissances de deux et des appels aux fonctions fournies.

Exercice 5

5.1. Expliquer pourquoi le *backtracking* peut être utilisé pour rechercher une solution (s'il en existe une) ou toutes les solutions $(x_1, \ldots, x_n)$ d'une équation de la forme

$$a_1 \times x_1 + \ldots + a_n \times x_n = q,$$

avec $\forall i, 0 \leq x_i \leq p$, p et q étant des entiers strictements positifs

Il n'est pas demandé d'écrire complétement l'algorithme (mais ce n'est pas interdit) mais d'en tracer les grandes lignes.

5.2. Quel arbre serait construit pour trouver toutes les solutions de l'équation

$3x_1 + x_2 + 4x_3 = 10$ avec $p = 4$.

Indication : cet arbre penche d'un côté .....