

**Exercice 1.** *Représentations de nombres*

**1.1.** Le nombre dont la représentation en base *dix* est 1500 s'écrit comme la somme de puissance 2 suivante :

$$\begin{aligned} 1500 &= 1024 + 256 + 128 + 64 + 16 + 8 + 4 \\ &= 2^{10} + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 \end{aligned}$$

La représentation en base *deux* du nombre est donc : 10111011100.

Une autre façon de procéder est de faire des divisions successives par 2 jusqu'à atteindre un quotient nul.

La suite des restes obtenus correspond à la suite des chiffres de la représentation binaire lue de droite à gauche.

Appliqué au nombre 1500, cela donne la suite de divisions (et de restes figurant à l'extrémité des flèches) :

$$1500 \rightarrow_0 750 \rightarrow_0 375 \rightarrow_1 187 \rightarrow_1 93 \rightarrow_1 46 \rightarrow_0 23 \rightarrow_1 11 \rightarrow_1 5 \rightarrow_1 2 \rightarrow_0 1 \rightarrow_1 0.$$

La suite des restes donne les chiffres de droite à gauche : 10111011100.

Pour obtenir la représentation en base *seize*, on constitue, à partir de la droite, des groupes de quatre chiffres dans la représentation binaire et on traduit ces groupes en leur correspondant en base *seize*. Cela donne : 101 1101 1100, traduit en base *seize* par 5DC.

Pour obtenir la représentation en base *huit*, on constitue à partir de la droite des groupes de trois chiffres dans la représentation binaire et on traduit ces groupes en leur correspondant en base *huit*. Cela donne : 10 111 011 100, traduit en base *huit* par 2734.

**1.2.** S'agissant d'une variable de type **short**, le codage est réalisé sur 16 bits.

En tant que quantité de type **short**, le nombre 1500 a donc comme représentation en machine 0000010111011100.

Le nombre  $-1500$  a comme représentation en machine la représentation en base *deux* du nombre  $2^{16} - 1500$ , c'est-à-dire  $65536 - 1500 = 64036$ .

Pour expliciter cette représentation :

1. on peut effectuer la conversion de 64036 en base *deux*, réalisée par divisions successives par 2. Dans le traitement de l'exemple, chaque flèche correspond à une division par 2 et indique le reste de la division :

$$\begin{aligned} 64036 &\rightarrow_0 32018 \rightarrow_0 16009 \rightarrow_1 8004 \rightarrow_0 4002 \rightarrow_0 2001 \rightarrow_1 1000 \rightarrow_0 500 \rightarrow_0 250 \\ &\rightarrow_0 125 \rightarrow_1 62 \rightarrow_0 31 \rightarrow_1 15 \rightarrow_1 7 \rightarrow_1 3 \rightarrow_1 1 \rightarrow_1 0. \end{aligned}$$

Cela conduit à la représentation 1111101000100100;

2. une autre manière de procéder est de rechercher le premier 1 à partir de la droite et de basculer tous les chiffres à sa gauche (changer les 0 en 1 et les 1 en 0). Dans le traitement de l'exemple donné ci-dessous, les chiffres à la gauche du premier 1 dans la représentation de 1500 sont soulignés. Ce sont les chiffres qui sont basculés pour obtenir la représentation de 1500 :

$$\underline{0000010111011100} \rightarrow 1111101000100100.$$

### Exercice 2. Affectations, opérations arithmétiques

Les instructions successives du programme ont les effets suivants :

- a vaut 5 et b vaut 2
- c prend la valeur 10 ( $2 \times 5$ )
- b prend la valeur 6 ( $5 + 1$ )
- a prend la valeur 26 ( $6 + 2 \times 10$ )
- les trois premiers `print` affiche respectivement 26, 6 et 10 (un nombre par ligne)
- a prend la valeur flottante 8.0, résultat de la division entière de 26 par 3 converti en `float`

Le premier `print` de la seconde série affiche : 8.0

- le `print` suivant affiche : 8, valeur de type `int` quotient entier de la division de 26 par 3
- dans `a / y`, l'un des opérandes est de type `float` : on fait donc une division en `float`.

On obtient donc  $26/4.0 = 6.5$ . Le dernier `print` affiche donc : 6.5

---

### Exercice 3. Lecture de nombres et suite de tests

```
import fr.jussieu.script.*;
class Exo3 {
    public static void main(String[] arg){
        Deug.print("Entrer la valeur de l'entier x : ");
        int x = Deug.readInt();
        Deug.print("Entrer la valeur de l'entier y : ");
        int y = Deug.readInt();
        // premier test : x strictement inferieur a -21
        if(x < -21) Deug.println("vrai");
        else Deug.println("faux");
        // deuxieme test : x est pair
        if(x % 2 == 0) Deug.println("vrai");
        else Deug.println("faux");
        // troisieme test : x egal au double de (y+1)
        if(x == 2 * (y + 1)) Deug.println("vrai");
        else Deug.println("faux");
        // quatrieme test : x positif ou nul et y strictement plus grand que x
        if(x >= 0 && y > x) Deug.println("vrai");
        else Deug.println("faux");
        // cinquieme test : le triple de y superieur ou egal a la moitie de x
        // Ne pas faire de division entiere .....
        if((3 * y) >= (x / 2.0f))    // autre solution : (6 * y >= x)
            Deug.println("vrai");
        else Deug.println("faux");
    }
}
```

Une autre solution, plus compacte utilisant des variables booléennes et leur impression avec la méthode `println` (les messages seront `true` et `false`) :

```
import fr.jussieu.script.*;
class Exo3V2 {
    public static void main(String[] arg){
        Deug.print("Entrer la valeur de l'entier x : ");
        int x = Deug.readInt();
```

```

    Deug.print("Entrer la valeur de l'entier y : ");
    int y = Deug.readInt();
    boolean t1, t2, t3, t4, t5;
    t1 = (x < -21) ; t2 = (x % 2 == 0); t3 = (x == 2 * (y + 1)) ;
    t4 = (x >= 0 && y > x); t5 = (6 * y >= x);
    Deug.println(t1 + " " + t2 + " " + t3 + " " + t4 + " " + t5);
}
}

```

#### Exercice 4. *Tarif d'assurance appliqué à un conducteur*

La solution donnée est une traduction brutale de l'énoncé :

```

import fr.jussieu.script.*;
class Exo4 {
    public static void main(String[] args){
        int age;           // age du client
        int permis;        // age du permis
        int accidents;     // nombre d'accidents
        boolean nouveau;   // nouveau client ?
        int annees;        // nombre d'annees d'assurance
        char tarif;        // tarif applique au client
        Deug.print("Age du conducteur ? "); age = Deug.readInt();
        Deug.print("Age du permis ? "); permis = Deug.readInt();
        Deug.print("Nombre d'accidents ? "); accidents = Deug.readInt();
        Deug.print("Nouveau client (repondre 1 si oui) ? "); int x = Deug.readInt();
        nouveau = (x == 1);
        if (!nouveau){ Deug.print("Nombre d'annees comme client ? "); annees = Deug.readInt();
        else annees = 0;
        if (age < 25 && permis < 2)
            if (accidents != 0) tarif = '0'; // pas d'assurance
            else tarif = 'A';
        else if( (age < 25 && permis >= 2) || (age >= 25 && permis < 2))
            switch(accidents) {
                case 0 : tarif = 'B'; break;
                case 1 : tarif = 'A'; break;
                default : tarif = '0';
            }
        else
            switch(accidents) {
                case 0 : tarif = 'C'; break;
                case 1 : tarif = 'B'; break;
                case 2 : tarif = 'A'; break;
                default : tarif = '0';
            }
        if(tarif == '0') Deug.println("Assurance impossible");
        else {
            if (annees >= 3) tarif++;
            Deug.println("Assurance au tarif " + tarif);
        }
    }
}
}

```

**Exercice 5.** *Calcul du nombre obtenu en lisant les chiffres à l'envers*

```
public static int miroir(int n){
    int m = 0; // nouveau nombre a priori nul
    while (n != 0) {
        // on multiplie ce qu'on a calcule et on ajoute le chiffre suivant
        // le chiffre suivant est le reste de la division par 10
        m = 10 * m + n % 10;
        n = n / 10; // on calcule le quotient
    }
    return m;
}
```

---

**Exercice 6.** *Entier exprimé sous forme d'au plus 4 carrés*

**6.1.** *Plus grande puissance de y divisant x*

```
public static int plusGrandePuissance(int x, int y){
    int n = 1, // y puissance 0 (puissance precedente divisant x)
    m = y; // y puissance 1 (puissance suivante)
    while (x % m == 0){ // la puissance suivante est-elle diviseur de x ?
        n = m; // la puissance suivante (qui divise x) devient la precedente
        m *= y; // calcul de la puissance suivante en vue de son test
    }
    return n; // on renvoie la derniere puissance de y divisant x
}
```

**6.2.** *Tester si un entier satisfait la condition de Gauss*

On commence par chercher la plus grande puissance  $z$  de 4 qui divise  $x$ . On calcule le quotient  $q$  de la division de  $x$  par ce nombre  $z$ . Le nombre  $x$  satisfait la condition de Gauss si et seulement si le reste de la division de  $q$  par 8 est égal à 7.

Cela se traduit en la fonction Java suivante :

```
public static boolean gauss (int x){
    int z = plusGrandePuissance(x, 4);
    int q = x / z;
    return(q % 8 == 7);
}
```

**6.3.** *Tester si un entier est un carre parfait*

Un entier est un carré parfait s'il est égal au carré de la partie entière de sa racine carrée. Cela conduit à la fonction suivante :

```
public static boolean carre(int x){
    int rac = (int)Math.sqrt(x);
    return (rac * rac == x);
}
```

#### 6.4. La décomposition d'un entier en somme d'au plus 4 carrés

- Dans cette première solution, la fonction `cond` renvoie le nombre d'éléments d'une décomposition minimale après avoir affiché les valeurs d'une telle décomposition.

```
public static int cond(int x){
    // si x est un carre parfait, on ecrit sa racine carre
    int rac = (int)Math.sqrt(x);
    long n; // variable pour les calculs intermediaires
    if(carre(x)){
        Deug.println(x + " est le carre de " + rac);
        return 1;
    }
    // si x ne satisfait pas la condition de Gauss, on cherche
    // une decomposition en 2 ou 3 carres
    // on affiche la premiere qu'on trouve et on s'arrete
    if(!gauss(x)){
        Deug.println(x + " ne satisfait pas la condition de Gauss ");
        // recherche d'une decomposition comme somme de 2 carres
        for(int i = 1; i <= rac; i++){
            for(int j = i; j <= rac; j++){
                n = i * i + j * j;
                if(n > x) break;
                if(n < x) continue;
                Deug.println(x + " est la somme des carres de " + i + " et " + j);
                return 2; }
        // recherche d'une decomposition comme somme de 3 carres
        for(int i = 1; i <= rac; i++){
            for(int j = i; j <= rac; j++){
                for(int k = j; k <= rac; k++) {
                    n = i * i + j * j + k * k;
                    if(n > x) break;
                    if(n < x) continue;
                    Deug.println(x + " est la somme des carres de "
                        + i + ", " + j + " et " + k);
                    return 3; }
            }
        // on cherche une decomposition comme somme de 4 carres exactement :
        // directement, dans le cas ou la condition de Gauss est satisfaite
        // et si jamais on n'avait pas trouve de decomposition plus petite sinon
        for(int i = 1; i <= rac; i++){
            for(int j = i; j <= rac; j++){
                for(int k = j; k <= rac; k++){
                    for(int l = k; l <= rac; l++) {
                        n = i * i + j * j + k * k + l * l;
                        if(n > x) break;
                        if(n < x) continue;
                        Deug.println(x + " est la somme des carres "
                            + i + ", " + j + ", " + k + " et " + l);
                        return 4; }
                    }
            }
        return -1; // necessaire bien que jamais atteint
    }
}
```

- Dans cette seconde solution, la fonction `cond` renvoie la référence d'un tableau dont la taille est le nombre d'éléments (1, 2, 3 ou 4) de la plus petite décomposition possible, le tableau contenant les valeurs d'entiers pour une telle décomposition. La solution donnée comporte l'ensemble des définitions de fonction, un programme principal. et des exemples d'exécution.

--> cat Exo6.java <== visualisation du contenu du fichier source

```
import fr.jussieu.script.*;
class Exo6 {
    public static void main(String[] arg){
        Deug.print("Entrer un entier ? ");
        int n = Deug.readInt();
        int[] t = cond(n);
        Deug.print(n + " est somme des carres de ");
        for(int i = 0; i < t.length; i++)
            Deug.print(t[i] + " ");
        Deug.println();
    }
    public static int plusGrandePuissance(int x, int y){
        int n = 1, m = y;
        while (x % m == 0){ n = m; m *= y; }
        return n;
    }
    public static boolean gauss (int x){
        int z = plusGrandePuissance(x, 4);
        int q = x / z;
        return(q % 8 == 7);
    }
    public static boolean carre(int x){
        int rac = (int)Math.sqrt(x);
        return (rac * rac == x);
    }
    public static int[] cond2(int x){
        int[] sol;
        // si x est un carre parfait, on crit sa racine carre
        int rac = (int)Math.sqrt(x);
        long n; // variable pour les calculs intermediaires
        if(carre(x)){
            sol = new int[1]; sol[0] = rac;
            return sol; }
        // si x ne satisfait pas la condition de Gauss, on cherche
        // une decomposition en 2 ou 3 carres
        if(!gauss(x)){
            Deug.println(x + " ne satisfait pas la condition de Gauss ");
            // recherche d'une decomposition comme somme de 2 carres
            for(int i = 1; i <= rac; i++){
                for(int j = i; j <= rac; j++){
                    n = i * i + j * j;
                    if(n > x) break;
                    if(n < x) continue;
                    sol = new int[2]; sol[0] = i; sol[1] = j;
                    return sol; }
            }
        }
    }
}
```

```

// recherche d'une decomposition comme somme de 3 carres
for(int i = 1; i <= rac; i++)
    for(int j = i; j <= rac; j++)
        for(int k = j; k <= rac; k++) {
            n = i * i + j * j + k * k;
            if(n > x) break;
            if(n < x) continue;
            sol = new int[3]; sol[0] = i; sol[1] = j; sol[2] = k;
            return sol;
        }
}

// recherche d'une decomposition comme somme de 4 carres
for(int i = 1; i <= rac; i++)
    for(int j = i; j <= rac; j++)
        for(int k = j; k <= rac; k++)
            for(int l = k; l <= rac; l++) {
                n = i * i + j * j + k * k + l * l;
                if(n > x) break;
                if(n < x) continue;
                sol = new int[4];
                sol[0] = i; sol[1] = j; sol[2] = k; sol[3] = l;
                return sol;
            }
return null; // jamais atteint mais indispensable
}

}

--> java Exo6          <== demande d'execution du programme Java
Entrer un entier ? 98765
98765 ne satisfait pas la condition de Gauss
98765 est somme des carres de 13 314
--> java Exo6
Entrer un entier ? 119025
119025 est somme des carres de 345
--> java Exo6
Entrer un entier ? 6786543
6786543 est somme des carres de 1 1 621 2530
--> java Exo6
Entrer un entier ? 89076
89076 ne satisfait pas la condition de Gauss
89076 est somme des carres de 4 16 298
-->

```