TD n° 12 : corrigé

EXERCICE 1 (*Entiers 4 bits*)

1. On peut représenter les entiers relatifs de la plage $[\![-2^{n-1}; 2^{n-1}-1]\!]$, ici $[\![-8, 7]\!]$.
2. Les entiers relatifs 0, 1, −1, 2, −5 et 7 sont respectivement représentés par 0000, 0001, 1111, 0010, 1011 et 0111.
3. Les entiers relatifs représentés par 1000 et 0101 sont respectivement −8 (qui est le plus petit entier relatif représentable) et 5.

EXERCICE 2 (*Flottants 8 bits*)

1. Le décalage vaut 3 et les exposants possibles sont ceux de la plage $[\![-2; 3]\!]$.
2. Les représentations des nombres 1, −1, 2, $\frac{1}{2}$ et −5 sont respectivement 0|011|0000, 1|011|0000, 0|100|0000, 0|010|0000 et 1|101|0100.
3. Le plus petit nombre strictement supérieur à 1 est 0|011|0001 et sa valeur est $1 + \frac{1}{16} = 1.0625$.
4. Le flottant représenté par 1|110|0101 est −10.5.
5. La représentation du plus petit nombre strictement positif (normalisé) est 0|001|0000 de valeur 0.25 et la représentation du plus grand nombre strictement positif (normalisé) est 0|110|1111 de valeur 15.5.
6. On a $3.125 < \pi < 3.25$ et il n'y a aucun flottant 8-bits strictement entre 3.125 (de représentation 0|100|1001) et 3.25 (de représentation 0|100|1010). Le flottant le plus proche de π est donc 3.125 de représentation 0|100|1001.

EXERCICE 3 Avec des flottants 64 bits c'est −323. On peut l'obtenir avec ce bout de programme C :

C

```c
int k = 0;
  double p = 1.0;
  while (p > 0) {
    k--;
    p /= 10;
  }
  k++;  // On dépasse de 1
```

EXERCICE 4 (*Flottants 16 bits*)

1. Le décalage est de $2^4 - 1$ soit 15. On peut représenter les entiers de 0 à $2^5 = 31$ sur 5 bits, les entiers 0 et 31 étant réservés. On peut donc représenter des exposants entre −14 et 15.
2. - 0 est représenté par 0|00000|0000000000.
   - $1 = +1.0 \times 2^0$. Donc l'exposant décalé est $15 = 01111_2$.
     1 est représenté par 0|01111|0000000000.
   - −1 est représenté par 1|01111|0000000000.
   - $2 = 1.0 \times 2^1$. Donc l'exposant décalé est $16 = 10000_2$.
     2 est représenté par 0|10000|0000000000.
   - $7 = 1.11_2 \times 2^2$. Donc l'exposant décalé est $17 = 10001_2$.
     7 est représenté par 0|10001|1100000000.

- $-259 = -(256+3) = -(01|0000|0011_2) = -1.00000011_2 \times 2^8$. Donc l'exposant décalé est $23 = 1|0111_2$ et -259 est représenté par $1|10111|0000001100$.

3. La représentation de 1 est $0|01111|0000000000$. Le plus petit nombre représentable suivant est celui dont la représentation est $0|01111|0000000001$. C'est $1 + \frac{1}{2^{10}} = 1.0009765625$.
4. Le plus petit nombre (normalisé) strictement positif est représenté par $0|00001|0000000000$. C'est $2^{-14} \approx 6.103515625 \times 10^{-5}$. Le plus grand nombre (normalisé) est celui de représentation $0|11110|1111111111$. C'est $1.1111111111_2 \times 2^{15} = 11111111111 \times 2^5 = (2^{11}-1) \times 2^5 = 2^{16} - 2^5 = 65536 - 32 = 65504$.
5. C'est le nombre 3.140625. On peut vérifier que c'est la meilleure approximation de π pour cette représentation.

EXERCICE 5 (*Associativité, réflexivité et distributivité*)

1. C
```c
bool test_associativite(double a, double b, double c) {
  return (a + b) + c == a + (b + c);
}
```
2. On peut prendre par exemple $a = 0.1, b = 0.2, c = 0.3$.
3. Elle est de 24,62% comme on peut le calculer avec le petit programme suivant :
C
```c
int nb = 0;
  for (int a = 0; a <= 100; a++) {
    for (int b = 0; b <= 100; b++) {
      for (int c = 0; c <= 100; c++) {
        if (!test_associativite(a / 10.0, b / 10.0, c / 10.0)) {
          nb++;
        }
      }
    }
  }
  double proba = nb / (101. * 101. * 101.);
```
4. Non, $NaN \neq NaN$.
5. Non. Le même exemple $a = 0.1, b = 0.2, c = 0.3$ le montre.
6. Non. Par exemple $0.2 * (0.1 + 0.3) \neq 0.2 * 0.1 + 0.2 * 0.3$.