Résumé : Représentation des nombres sur machine.

L'énoncé de ce TP se trouve sur
`http://perso-laris.univ-angers.fr/~delanoue/istia/calcul_numerique/td2.pdf`

Exercice 1

Sous `Matlab`, on peut choisir de représenter les nombres entiers sous différents formats. Les tableaux suivants récaputilent ces types en précisant la mémoire utilisée et l'étendue des nombres représentables :

	Etendue	Type	Nombre d'octects
`int8`	-128 à 127	Signed 8-bit integer	1
`int16`	-32,768 à 32,767	Signed 16-bit integer	2
`int32`	-2,147,483,648 à 2,147,483,647	Signed 32-bit integer	4
`int64`	-9,223,372,036,854,775,808 à 9,223,372,036,854,775,807	Signed 64-bit integer	8

	Etendue	Type	Nombre d'octects
`uint8`	0 à 255	Unsigned 8-bit integer	1
`uint16`	0 à 65 535	Unsigned 16-bit integer	2
`uint32`	0 à 4 294 967 295	Unsigned 32-bit integer	4
`uint64`	0 à 18 446 744 073 709 551 615	Unsigned 64-bit integer	8

1. Lancer `Matlab` et taper le code suivant :

   ```
   a = uint8(10)
   b = 10
   ```

2. Bien que la réponse de `Matlab` soit la même pour ces deux instructions, ces deux variables sont stockées différemment. Afin de connaître de quel type est une variable, on peut utiliser la commande `whos`. De quels types sont les variables `a` et `b` ?

3. Il est possible de modifier le *format d'affichage* des nombres grâce à la commande

   ```
   format hex
   ```

   Taper les commandes suivantes

   ```
   c = uint8(10)
   d = uint16(10)
   e = uint16(255)
   ```

   Interpréter les chaînes de caractères affichées.

4. Quel est le nombre entier qui sera représenté par `00000000000190c2` (codé en non signé), vous présenterez votre solution en base 10 ?

5. Quelle propriété de l'addition permet de justifier l'égalité suivante :

$$f + (h - g) = (f + h) - g$$

6. Calculer

$$250 + 12 - 10$$

7. Taper les commandes suivantes

   ```
   format short
   f = uint8(250)
   g = uint8(10)
   h = uint8(12)
   ```

8. Evaluer avec `Matlab` les expressions suivantes `(f + h) - g` et `f + (h - g)`. Justifier les résultats obtenus.

Exercice 2 (Flottant)
Le format *double* permet de représenter (pas toujours de façon exacte) les nombres réels sous forme binaire. Le terme double provient du nom complet : nombre flottant à double précision. Il y a quelques années, les flottants (*float*) étaient codés sur 32 bits. Cependant, certaines applications en sciences et en ingénieurie ont montré que cette précision n'était pas suffisante. Il a donc été décidé de doubler le nombre de bits permettant de mieux de représenter les nombres réels.

Les 64 bits des *double* se décomposent de la façon suivante :
- 1 bit pour le signe (0 signifie positif et 1 signifie négatif),
- 11 bits pour l'exposant,
- 52 bits pour la mantisse.

$$\underbrace{a_1}_{\text{signe}}\ \underbrace{a_2 \dots a_{12}}_{\text{exposant}}\ \underbrace{a_{13} \dots a_{64}}_{\text{mantisse}}$$

où chaque a_i est un symbole pris parmis l'ensemble $\{0, 1\}$.

Dans cette représentation, on impose aussi au premier chiffre de la mantisse d'être non nul, et comme le seul chiffre non nul est 1, on ne stocke pas le premier 1 (il est tout simplement implicite). L'exposant est quant à lui stocké en addition un biais de 01111111111_2. Ce permet de stocker des exposants négatifs. Par exemple, si l'exposant est nul, alors $a_2 \dots a_{12}$ sera égale à 01111111111.

Ces informations sont suffisantes pour interpréter un nombre flottant en double précision sauvegardé sous forme binaire. Finalement, `Matlab` nous fournit une version hexadécimal de cette forme binaire via le mode `format hex`.

1. Sur le paper, écrire 16 en base 2.
2. Quels sera sa mantisse et son exposant s'il est codé en double ?
3. Taper les commandes suivantes

   ```
   format hex
   i = 16
   ```

   Vous devriez obtenir :

   ```
   4030 0000 0000 0000
   ```

   Donc

   $$\underbrace{0100}_{4}\underbrace{0000}_{0}\underbrace{0011}_{3}\underbrace{0000}_{0}\ \underbrace{0000}_{0}\underbrace{0000}_{0}\underbrace{0000}_{0}\underbrace{0000}_{0}\ \underbrace{0000}_{0}\underbrace{0000}_{0}\underbrace{0000}_{0}\underbrace{0000}_{0}\ \underbrace{0000}_{0}\underbrace{0000}_{0}\underbrace{0000}_{0}\underbrace{0000}_{0}$$

   On en déduit :

   $$\underbrace{0}_{\text{Signe}}\ \underbrace{10000000011}_{\text{Exposant - 11 digits}}\ \underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000}_{\text{Mantisse - 52 digits}}$$

   Avec le biais, on a l'exposant qui est 4 et avec le 1 implicite, la mantisse est $(1, \underbrace{0 \dots 0}_{52 \text{ fois}})_2$. Donc le nombre stocké est bien :

   $$(1, \underbrace{0 \dots 0}_{52 \text{ fois}})_2 \times 2^4 = 16$$

4. Reprendre les questions précédentes avec le nombre `48`.
5. Quel est le nombre décimal qui est représenté en double par
   0011 1111 1110 1000 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 ?
6. Ecrire 0.5 en base 2.
7. Ecrire 1/3 en base 3.

8. Ecrire 0.1 en base 2. Indication : on pourra poser la division en base 2 de 1_2 par 1010_2.
9. Taper les commandes suivantes

```
format short;
z = 0; for k=1:10 z = z +0.1; end;
```

10. Quelle est la valeur de `z` ?
11. Taper la commande

```
z == 1
```

Interpréter et justifier le résultat obtenu.

Exercice 3 (Evaluation de polynômes)
Considérons le polynôme P mis sous forme factorisée $P = (x-1)^8$.

1. Développer le polynôme P.
2. On pose $d = 0.000007237$. Calculer (à la main) la différence entre $1-d$ et $1+d$.
3. Généralement, les polynômes ne sont pas donnés sous forme factorisée. Dans la suite de cette exercice, nous allons évaluer le polynôme P en $1+d$ et $1-d$ de trois manières différentes. On interprétera (sans justifier) la différence entre les deux évaluations comme une mesure de la précision de la méthode.

   Que réalise le code `matlab` suivant ?

```
d = 0.000007237;
x = [1 - d, 1 + d]
y1 = [0,0];
p = [1 -8 28 -56 70 -56 28 -8 1];
for i = 1:length(p)
     y1 = y1 + p(i)*x.^(i - 1);
end
y1
```

4. Copier et modifier le code précédent de sorte que l'évaluation se fasse en suivant l'ordre décroissant des monômes. Notez-vous Une amélioration ?
5. Une autre façon d'évaluer un polynôme est de le mettre sous forme de Horner. Etant donné un polynôme $P = \alpha_n x^n + \cdots + \alpha_0$, cette méthode consiste à appliquer récursivement la règle de réécriture à $P \to (Q(x))x + P(0)$. Par exemple $3x^3 + 4x^2 + 2x^1 + 4$ devient successivement $(3x^2 + 4x + 2)x + 4$ puis $((3x + 4)x + 2)x + 4$.
   Proposer le polynôme P sous forme de Horner.
6. Compléter le code suivant de sorte que le polynôme P soit évalué en suivant la règle de Horner.

```
y3 = [0 0];
p = [1 -8 28 -56 70 -56 28 -8 1];
for i = 1:length(p)

end
y3
```

7. On suppose que P est de degré n. Donner le nombre de multiplications et le nombre d'additions pour la méthode d'évaluation via la forme développé et via la méthode d'Horner, laquelle est plus rapide ?
8. Dans ce cas, la méthode de Horner donne de meilleurs résultats. Dans cette question, on va proposer une figure permettant de comparer les différentes méthodes d'évaluations.

   (a) Créer un vecteur `D` composé de 1000 points entre 0.999 et 1.001.

(b) Calculer la différence entre $P(1+d) - P(1-d)$ en utilisant
- sa forme factorisée,
- une addition des valeurs des momônes (de façon croissante ou décroissante),
- la forme de Horner.

9. Utiliser la commande `plot` afin d'afficher les résultats des 3 méthodes

Vous devrez obtenir une figure comme celle ci :
`http://perso-laris.univ-angers.fr/~delanoue/istia/calcul_numerique/error.gif`
- en noir : forme factorisée
- en bleu : developpée
- rouge : Horner