Informatique: TD2

MP2I Lycée Pierre de Fermat

La calculatrice est **interdite** pour ce TD !

Exercice 1.

Hexadécimal

Q1. Donnez l'écriture décimale des entiers suivants :

a) $\overline{100101}^2$ b) $\overline{253}^7$ c) $\overline{2AF}^{16}$

Q2. Écrivez n en base B pour :

a) $n = 100, B = 2$ b) $n = 91, B = 7$ c) $n = 3, B = 3$ d) $n = 59, B = 16$.

Q3. Soit B une base quelconque. Écrivez n en base B pour :

a) $n = B^k$ avec $k \in \mathbb{N}$ b) $n = B^k - 1$ avec $k \in \mathbb{N}$

Rappel : une utilité de la base 16 (l'hexadécimal) est que l'on peut passer facilement de la base 16 à la base 2 et inversement : un chiffre en base 16 correspond à 4 chiffres en base 2.

Q4. Combien de chiffres hexadécimaux faut-il pour représenter un octet ? Et pour un entier 32 bits ?

Q5. Remplir le tableau suivant :

Dec	15	735	493		
Bin	1111	0010 1101 1111		1010 0110 0001	
Hex	f	2df			d4c

Exercice 2.

Addition des entiers non-signés

Dans cet exercice, on considère des entiers non-signés sur 8 bits.
Effectuer les additions suivantes **en binaire**, et dire lorsqu'elles causent un dépassement d'entier :

a) 95 + 132 b) 12 + 167 c) 7 + 64 d) 184 + 72

Exercice 3. *Lecture en base B*

On considère l'algorithme suivant de lecture en base B :

Algorithme 1 : Lecture depuis une base

Entrée(s) : B une base, $a_{l-1} \dots a_0$ un mot sur l'alphabet $[\![0, B-1]\!]$
Sortie(s) : $n \in \mathbb{N}^*$ tel que $n = \overline{a_{l-1} \dots a_0}^B$

```
1 n ← 0;
2 i = 0;
  // Invariant: ...
3 tant que i < l faire
4 |   n ← n + a_i B^i;
5 |   i ← i + 1;
6 retourner n
```

Q1. Montrez la correction de cet algorithme à l'aide d'un bon invariant de boucle.
Indication : Déterminez ce que vaut n à chaque étape de l'algorithme.

Q2. En fonction du nombre de chiffres l lus, quel est le nombre total de multiplications effectuées par l'algorithme ? On considèrera que le calcul de puissance est fait naïvement, i.e. que calculer x^k demande $k-1$ multiplications.

Q3. Modifier l'algorithme pour qu'il effectue moins de multiplications. On pourra introduire une variable p vérifiant l'invariant suivant : "$p = 10^i$". Combien de multiplications exécute t-il au total en fonction de l ?

On s'intéresse maintenant à un deuxième algorithme, effectuant moins de multiplications, appelé **méthode de Horner**. Cette méthode permet en général d'évaluer la valeur d'un polynôme en un point, on étudie ici une version qui s'applique directement à la lecture d'un nombre en base B :

Algorithme 2 : Horner

Entrée(s) : B une base, $a_{l-1} \dots a_0$ un mot sur l'alphabet $[\![0, B-1]\!]$
Sortie(s) : $n \in \mathbb{N}^*$ tel que $n = \overline{a_{l-1} \dots a_0}^B$

```
1 n ← 0;
2 i = l − 1;
  // Invariant: ...
3 tant que i ≥ 0 faire
4 |   n ← Bn + a_i;
5 |   i ← i − 1;
6 retourner n
```

Q4. Montrer la correction de cet algorithme. Quel est le nombre total de multiplications qu'il effectue en fonction de la longueur l du mot en entrée ?

Q5. En s'inspirant de l'algorithme précédent, proposer un algorithme d'évaluation de polynôme économe en multiplications :

Algorithme 3 : Eval

Entrée(s) : $a_{l-1}, \dots, a_0$ coefficients d'un polynôme P, $x_0 \in \mathbb{R}$
Sortie(s) : $P(x_0) = \sum_{i=0}^{l-1} a_i x_0^i$

Exercice 4. *Opérations bit à bit*

Lorsque l'on considère des entiers écrits en base 2, on doit avoir une vision duale : d'une part celle des entiers, d'autre part celle des mots sur l'alphabet $\{0,1\}$.
Par exemple, l'octet 01001101 représente l'entier 77 ET est une suite de 8 valeurs booléennes.

On rappelle les trois opérateurs booléens bits à bits : &, | et $\sim$. Ces trois opérateurs font respectivement un ET, un OU et un NON bit à bit. Par exemple sur 4 bits :

$$1001\&1100 = 1000, 1001|1100 = 1101 \text{ et } \sim 1001 = 0110$$

On fixe $l \in \mathbb{N}$ un nombre de bits sur lequel les nombres sont écrits. On note $\mathbb{N}_l$ l'ensemble des entiers représentables sur l bits non-signés. Les opérateurs bit-à-bit agissent techniquement sur les mots binaires, i.e. sur les suites de 0 et de 1, mais on étend assez naturellement leur action au entiers de $\mathbb{N}_l$: pour n et m deux entiers ayant pour écritures binaires respectives $a_{l-1}\dots a_0$ et $b_{l-1}\dots b_0$, on aura :

$$n\&m = \overline{(a_{l-1}\dots a_0)\&(b_{l-1}\dots b_0)}^2$$

et idem pour | et $\sim$. Autrement dit, pour calculer $n\&m$, on écrit n et m en binaire sur l bits, on calcule le ET bit à bit, et l'on interprète le résultat comme un entier en base 2.

Q1. En se plaçant sur $l = 8$ bits, donner les résultats des calculs suivants :

1. $100\&52$
2. $31\&64$
3. $95|44$
4. $31|64$
5. ~ 216
6. $(\sim 100)|100$

Q2. Pour $x \in \mathbb{N}_l$, que vaut $x| \sim x$? Et $x\& \sim x$?

Q3. Montrez que pour tout $x, y \in \mathbb{N}_l$, $x\&y \leq \min(x, y)$. Sur 8 bits, trouvez un cas où l'inégalité est stricte, et un où il y a égalité mais où $x \neq y$.

On rappelle également les opérateurs de ***décalage*** : >> et << :

$$\forall u = a_{l-1}\dots a_0 \in \{0,1\}^*, \forall k \in \mathbb{N}, u << k = a_{l-1-k}\dots a_0 00\dots 0$$

avec k 0 ajoutés à droite de u et k bits perdus à gauche, et

$$\forall u = a_{l-1}\dots a_0 \in \{0,1\}^*, \forall k \in \mathbb{N}, u >> k = 00\dots 0 a_{l-1}\dots a_k$$

avec k 0 ajoutés à gauche de u et k bits perdus à droite. On étend l'action de ces opérateurs aux entiers de $\mathbb{N}_l$.

Q4. Sur 8 bits, donner le résultat des calculs suivants :

1. $7 << 3$
2. $73 << 5$
3. $32 >> 3$
4. $55 >> 2$

Q5. Pour $a \in \mathbb{N}_l$ un entier et $k \in \mathbb{N}$, exprimer $a >> k$ et $a << k$ en fonction de a, k et l.

Q6. Pour $n \in \mathbb{N}_l$, donner une formule permettant d'obtenir le i-ème bit de n en n'utilisant que les opérateurs bit-à-bit et les opérateurs de décalage.

Q7. Pour $n \in \mathbb{N}_l$, donner une formule permettant de changer le i-ème bit de n en un 1.

Q8. Pour $n \in \mathbb{N}_l$, donner une formule permettant d'inverser le i-ème bit de n, i.e. transformer un 1 en 0 et inversement. (*Vous pouvez sauter cette question et y revenir après avoir fait l'exercice 5 si vous n'y arrivez pas.)*

Exercice 5.

Ou exclusif

Le but de cet exercice est d'écrire un algorithme d'addition en binaire n'utilisant que des opérations booléennes. On introduit un nouvel opérateur booléen : le ***OU exclusif***, aussi appelé ***xor***, noté $\oplus$ (Cf Fig. 1).

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 1 – Table de vérité du ***xor*** logique

Q1. Rappelez les tables de valeurs des opérateurs booléens && || et !.

Q2. Pour a et b deux booléens, exprimez $a \oplus b$ en n'utilisant que les opérateurs && || et !.

Q3. Montrer que le xor est associatif.

En C, il n'existe pas de symbole pour le xor logique. En revanche, on peut faire des xor bit à bit avec l'accent circonflexe `^`.

Q4. On se place sur 8 bits. Combien vaut :

a) 9 ^ 20 b) 91 ^ 107 c) 255 ^ 183

Q5. De manière générale, sur l bits, pour n un entier non-signé représentable, combien vaut :

a) n ^ 0 ? b) n ^ $(2^l - 1)$? c) n ^ n ?

Q6. Pour n, m représentables sur l bits, à quelle condition a-t-on que n ^ $m = n + m$?

Q7. Que fait le code suivant (on suppose que x et y ont été définies plus haut) :

```
1 x = x^y;
2 y = x^y;
3 x = x^y;
```

Exercice 6.

Addition binaire

Cet exercice est à faire après l'exercice 5, et vous pouvez utiliser librement le OU exclusif.

Soient a, b, c des booléens. On note s et c' les booléens qui représentent l'addition $a + b + c$, avec s le résultat et c' la retenue. Par exemple pour $a = b = c = 1$, on a $s = 1$ et $c' = 1$.

Q1. Exprimez s et c' en fonction de a, b, c en n'utilisant que des opérateurs booléens.

L'idée de l'algorithme d'addition est d'utiliser les formules précédentes pour additionner deux nombres binaires $a_{k-1} \dots a_0$ et $b_{k-1} \dots b_0$ en faisant des additions bit à bit et en propageant la retenue.

Q2. De combien de bits a t-on besoin pour stocker le résultat d'une addition de deux nombres à k bits ?

Q3. Écrivez un algorithme d'addition binaire :

Algorithme 4 : Addition binaire

Entrée(s) : $a = a_{k-1} \dots a_0$ et $b = b_{k-1} \dots b_0$ deux mots binaires
Sortie(s) : $s = s_k \dots s_0$ le résultat de l'addition de a et b

Exercice 7.

Encodage de jeux

On veut encoder une grille de morpion par un entier non signé 32 bits. On veut que notre encodage nous permette de récupérer l'état de la grille facilement en manipulant les bits.

On suppose qu'on a deux joueurs : le joueur X et le joueur O. Une case peut contenir un X, un O ou rien. On assigne à chaque case un nombre entre 0 et 3 selon son contenu :

$$\begin{array}{lcl} \text{X} & \mapsto & 2 \\ \text{O} & \mapsto & 3 \\ \text{rien} & \mapsto & 0 \end{array}$$

On remarque que le nombre 1 ne correspond à rien. Ensuite, on numérote les 9 cases d'une grille comme suit :

0	1	2
3	4	5
6	7	8

Enfin, en notant a_i le nombre correspondant au contenu de la case i, on encode une grille G par :

$$\sum_{i=0}^{8} a_i 4^i$$

Q1. Donnez l'écriture en bases 2, 4, 16 et 10 de l'encodage des grilles suivantes (vous pouvez utiliser la calculatrice pour cette question uniquement) :

a)

b)

		O
X	X	O
		X

c)

O	X	O
X	X	O
	O	X

Q2. Si n est l'encodage d'une grille et $0 \leq i \leq 8$, exprimez à l'aide des opérateurs >> et & une formule calculant un booléen indiquant si la case i de la grille est remplie ou non.

Q3. Soit n l'encodage d'une grille, et $0 \leq i \leq 8$. On suppose que la case i de la grille encodée par n est remplie. Exprimez à l'aide des opérateurs >> et & une formule calculant un booléen indiquant si la case i de la grille contient un O ou un X.

Q4. Soit n l'encodage d'une grille, et $0 \leq i \leq 8$. On suppose que la case i de la grille encodée par n est vide. Exprimez à l'aide des opérateurs << et | une formule calculant l'encodage de la grille obtenue en rajoutant le symbole X dans la case i

Q5. Idem pour le symbole O.

Q6. Expliquez, selon le même principe, comment on pourrait encoder une grille de Sudoku. De combien de bits aurait-on besoin ?

Exercice 8. *Quantités de longueur variable*

Les fichiers MIDI servent à encoder des morceaux de musiques sous la forme de suites d'évènements : appuyer sur une note, relâcher une note, baisser le volume, etc... Dans un fichier midi, pour chaque évènement, on stocke le delta de temps (dans une unité arbitraire) depuis l'évènement précédent. Ces durées sont des entiers, et dans la plupart des fichiers elles tiennent sur un ou deux octets, mais rien n'empêche a priori que ces durées prennent des grandes valeurs.

Le format MIDI stocke donc les nombres selon une convention différente des entiers à longueur fixée comme nous avons pu voir avec les types int, long int, etc... Cet encodage s'appelle Variable Length Quantity (ou VLQ), et a comme avantage qu'il utilise peu d'octets pour stocker des petits nombres. Dans les fichiers midi, les petits nombres ayant plus de chances d'apparaître que des très grands nombres, l'encodage VLQ permet d'économiser de la mémoire, en adaptant le nombre d'octets utilisés selon le nombre à encoder.

Pour obtenir l'encodage VLQ d'un entier $n \in \mathbb{N}$, on commence par l'écrire en base 2, en rembourrant avec des 0 après le bit de poids fort pour obtenir une longueur multiple de 7 :

$$n = \overline{x_{7B-1}x_{7B-2}\dots x_1x0}^2$$

B représente le nombre de blocs de 7 bits utilisés pour écrire n :

Bloc n°	Bits
$B-1$	$x_{7B-1}\dots x_{7B-7}$
$B-2$	$x_{7B-8}\dots x_{7B-14}$
$\dots$	$\dots$
1	$x_{13}\dots x_7$
0	$x_6\dots x_0$

Enfin, on forme B octets en préfixant chaque bloc de 7 bits par un bit : 0 pour le dernier bloc et 1 pour tous les autres. On écrit donc :

$$\underline{1}x_{7B-1}\dots x_{7B-7} \quad \underline{1}x_{7B-8}\dots x_{7B-14} \quad \dots \quad \underline{1}x_{13}\dots x_7 \quad \underline{0}x_6\dots x_1x_0$$

Les bits soulignés désignent les bits de préfixe ajoutés. Par exemple, pour encoder 7631 en VLQ :

$$7631 = \overline{0111011\ 1001111}^2$$

L'encodage de 7631 en VLQ sera donc 10111011 01001111, c'est à dire en hexadécimal `0xBB4F`.

Q1. Donnez l'encodage en VLQ des entiers suivants comme suite de bits puis comme suite d'octets. Montrez les étapes de vos calculs.

1. 585
2. 53
3. 2^{12}
4. `0x2ac9f`

Q2. Décodez les nombres suivants qui ont été mis sous format VLQ. Montrez les étapes de vos calculs

1. `0x8829`
2. `0xa154`
3. `0x80a154`

Q3. Expliquez à quoi sert le bit de préfixe des octets dans l'encodage et pourquoi on ne peut pas s'en passer.

Le reste de l'exercice est à faire en C, et utilise la notion de tableau, que nous allons voir au chapitre 3 : vous pouvez le faire si vous savez déjà faire des tableaux en C ou si vous voulez prendre de l'avance sur le cours.

On s'intéresse maintenant à l'implémentation d'algorithmes de codage et de décodage du format VLQ en C. Le but est d'écrire les deux fonctions suivantes :

```
1 /* Écrit dans le tableau t la suite des octets représentant n dans l'encodage
2  * VLQ, et renvoie le nombre d'octets ayant été réellement utlisés */
3 int vlq_encode(long unsigned int n, unsigned char* t);
4
5 /* Interprète les `size` premiers octets pointés par t
6  * comme une valeur encodée en VLQ, et renvoie l'entier lu */
7 long unsigned int vlq_decode(unsigned char* t, int size);
```

Le type `unsigned char` représente un octet.

Q4. Implémentez les deux fonctions, et testez-les bien.