Architecture des ordinateurs : TD1

Université de Tours

Département informatique de Blois

Représentation de l'information & arithmétique des ordinateurs

*
* *

Problème 1

1. Donner la représentation des nombres suivants en complément à 2 selon un nombre de bits $k \in 4\mathbb{N}$ (multiple de 4) que l'on précisera.

   (a) $\langle 42 \rangle_7$

   (b) $\langle 101010 \rangle_8$

   (c) $\langle -2018 \rangle_{10}$

   (d) $\langle -CA7 \rangle_{16}$

2. Donner la représentation IEEE 754 en base 2 sous 32 bits des nombres en base décimale et la représentation décimale des nombres sous représentation IEEE 754 :

   (a) $\langle -2^{120} \rangle_{10}$

   (b) $\langle 13,658203125 \rangle_{10}$

   (c) $\langle 263,3 \rangle_{10}$

   (d) $\langle 0\ 10000100\ 01010000000000000000000 \rangle_{I3E}$

Problème 2

On considère le programme suivant permettant le calcul d'une suite $(u_n)_{n \in \mathbb{N}}$:

```
float u_n(int n) {
   return n == 0 ? (float) 1.0 / 3 : 4 * u_n(n-1) - 1;
}
```

1. Modélisation mathématique :

   (a) Démontrer que : $\forall n \in \mathbb{N}, u_n = \frac{1}{3}$.

   (b) Démontrer que le nombre $\frac{1}{3}$ ne peut pas être représenté de manière exacte à l'aide de la norme IEEE 754.

2. Donner la représentation IEEE 754 de $\frac{1}{3}$ au sein du programme.

3. L'appel `u_n(42)` retourne la valeur 1.9215359×10^{17}.

   (a) Expliquer briévement la différence entre le résultat théorique et celui donné par le programme. Selon vous, et sans le démontrer, quelle type de croissance suit cette erreur (logarithmique, linéaire, quadratique, exponentielle, autre) ?

(b) Sur la base d'une erreur d'approximation $\delta > 0$, déterminer l'expression de l'erreur Δ_n commise par la méthode `u_n()` au rang n.

(c) Calculer l'erreur $\delta_{\max}$ puis appliquer le calculer de l'erreur Δ_{42} commise par la suite au rang $n = 42$ sur la base d'une erreur $\delta = \delta_{\max}$. Le résultat vous semble t-il cohérent par rapport au résultat rendu par l'ordinateur ? Argumentez votre réponse.

Problème 3

Soit $(n, k) \in \mathbb{N}^2$ avec $n \geq k$. On cherche ici à calculer les coefficients binomiaux C_n^k $\left(\text{ou } \binom{n}{k}\right)$ de la formule du binôme de Newton. On rappelle que :

$$C_n^k = \frac{n!}{k!(n-k)!}$$

1. On considère le programme C donné en annexe.

   Que pouvez-dire de cette méthode de calcul. Argumentez vos propos à l'aide de la sortie programme.

2. Démontrer que : $\ln\left(C_n^k\right) = \sum_{i=1}^{k} \ln(n+1-i) - \ln(i)$.

3. La propriété utilisée à la question 2. est appellée *réduction logarithmique*.

   (a) En ce basant sur cette propriété, proposer un progamme `double binomial(int n, int k)` en C ou en Java permettant le calcul efficace des coefficients binomiaux. On précise que l'on donne accès aux fonctions de la bibliothèque `math`, `log(x)` et `exp(x)`, retournant respectivement les valeurs $\ln(x)$ et e^x.

   (b) Donner la notation IEEE 754 du nombre maximal N pouvant être représenté avec un `double`. On précise que $N \approx 1,79 \times 10^{308}$.

   Déterminer à l'aide des valeurs données par l'annexe la valeur maximale n pour laquelle le programme du 3.a peut-être appellé sans comettre d'erreur.

Problème 4

Le but de cet exercice est de permettre à l'ordinateur de calculer la racine d'un nombre de manière efficace, rapide tout en limitant l'erreur d'approximation commise.
Pour se faire, on peut utiliser l'algorithme de Newton.

1. Donner la suite $(x_n)_{n\in\mathbb{N}}$ associée à l'algorithme de Newton permettant de calculer $\sqrt{x}$, pour tout $x \in \mathbb{R}^+$.

2. Appliquer l'algorithme pour le calcul de $\sqrt{2}$, on pourra s'arrêter à $n = 3$ et on donnera la forme fractionnaire.

3. Représenter la valeur $\sqrt{2}$ en half-precision IEEE 754. Ce format inclue :

   - 1 bit de signe
   - 5 bits d'exposant
   - 10 bits de mantisse
   - Un biais $\varepsilon = 15$

4. Calculer l'erreur approximation $\delta_{\max}$ commise sous ce format.

Problème 5

Soit un entier positif $x = \langle x_{n-1} ... x_0 \rangle_2$ et $k \in [\![0, n-1]\!]$. On considère l'opérateur de *décalage à gauche* $\ll$ définie tel que :

$$x \ll k = \left\langle y_i \middle| \forall i \in [\![0, n-1+k]\!], \begin{cases} y_i = 0 & \text{si } i < k \\ y_i = x_{i-k} & \text{sinon} \end{cases} \right\rangle$$

1. Calculer $4 \ll 1$, $42 \ll 3$, $12 \ll 2$.
2. Démontrer que $x \ll k = 2^k \times x$.
3. Comment effectuer $x \times 5$ à l'aide des opérateurs $\ll$ et $+$? Et $x \times 7$?
4. Déterminer un algorithme permettant de réaliser $x \times n$ à l'aide du nombre minimal opérateurs $\ll$ et $+$ pour tout $n \in \mathbb{N}^*$.

Annexe

Coefficients binomiaux : Problème 1

```c
#include <stdio.h>

int factorielle(int n) {
    return n == 0 ? 1 : n * factorielle(n - 1);
}
int binomial(int n, int k) {
    return factorielle(n)/(factorielle(k) * factorielle(n - k));
}

int main() {
    for(int n = 0; n < 15; n++) {
        for(int k = 0 ; k < n+1; k++) {
            printf("%u \t", binomial(n, k));
        }
        printf("\n");
    }
    return 0;
}
```

```
--- Sortie programme : affichage ---
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
1    6    15   20   15   6    1
1    7    21   35   35   21   7    1
1    8    28   56   70   56   28   8    1
1    9    36   84   126  126  84   36   9    1
1    10   45   120  210  252  210  120  45   10   1
1    11   55   165  330  462  462  330  165  55   11   1
1    12   66   220  495  792  924  792  495  220  66   12   1
1    4    24   88   221  399  532  532  399  221  88   24   4    1
1    0    1    5    14   29   44   50   44   29   14   5    1    0    1
```

Valeurs de $\ln\left(C_n^k\right)$ ***: Problème 1***

n	$\ln\left(C_n^k\right)$
1026	707.4762607243114
1027	708.1684346687844
1028	708.8615818493444
1029	709.5537576845151
1030	710.2469048650747
1031	710.93908258361
1032	711.6322297641697
1033	712.3244093587805
1034	713.0175565393406
1035	713.7097380027809

Pour $k = n/2$.