TP Arithmétique et Calcul

Calcule Haute Performance, Simulation.

Réalisé par :

BENAMROUCHE Nadjib

Préambule: les headers:

<float.h>: cette bibliothèque contient des constantes prédéfinies tel que:

- ➢ FLT_MIN: le plus petit flottant positif normalisé.
- ➢ FLT_EPSILON: différence entre la valeur réelle 1.0 et le plus petit flottant immédiatement supérieur.
- ➢ FLT_RADIX: base de l'exposant dans la représentation d'un flottant.
- ➢ FLT_MANT_DIG: taille de la mantisse dans la représentation d'un réel flottant.

Exercice 10.3:

Le nombre 1/10 est un réel il n'admet pas une représentation flottante exacte avec la norme IEEE_754 pour le représenter on utilise la notion d'arrondi.

La norme IEEE_754 avec simple précision Binary 32: 1 bit de signe, 8 bits d'exposant, 23 bits de mantisse.

Représentation de 1/10 = 0 | 01111111 | 00001100110011001100110.

Pour obtenir le plus petit nombre flottant positif normalisé on a utilisé la constante FLT_MIN prédéfinie dans la bibliothèque float.h. Après la division successif de x sur 2 au départ à chaque division ça demenie l'exposant de x jusqu'à arriver à e = -126 (cet exposant est réservé pour la plage des nombre sous normaliser). A l'itération 124 on obtient notre premier nombre dénormalisé.

La précision de la représentation B-32 = 24 (nombres de bits de la mantisse) d'où on continue à diviser sur deux 12 fois. Cette division engendre une perte de bits dû au décalage à droite des bits de la mantisse. Pour la multiplication des bits se décalant vers la gauche, on rajoute des zéros à la place des 12 bit perdus.

Résultat final après multiplication x = 0.099976.

```
x =   0.00000000 ,i = 122
Avec l'ecriture rationnel en notation scientifique de x = 1.880791e-38
x =   0.00000000 ,i = 123
Avec l'ecriture rationnel en notation scientifique de x = 9.403954e-39
************************* Avec precision ****************************
La valeur apres multiplication x = 0.099976
Avec l'ecriture rationnel en notation scientifique de x = 9.997559e-02
nadjib@nadjib-HP-Laptop-15s-eq0xxx:~/Bureau/tparith/exo10_3$ make run_fast
```

Remarque:

Pour exécuter le code: make run (avec -O0 comme option de compilation sans optimisation). Pour utiliser l'option -Ofast (compilation avec optimisation) on utilise: make run_fast.

Explication: Pour le l'option -Ofast on obtient pas les mêmes résultats car les calculs se font d'une manière optimisée, il y a une perte d'information et un manque de pression des calculs.

```
x =   0.00000000 ,i = 122
Avec l'ecriture rationnel en notation scientifique de x = 1.880791e-38
x =   0.00000000 ,i = 123
Avec l'ecriture rationnel en notation scientifique de x = 0.000000e+00
************************* Avec precision ******************************
La valeur apres multiplication x = 0.000000
Avec l'ecriture rationnel en notation scientifique de x = 0.000000e+00
nadjib@nadjib-HP-Laptop-15s-eq0xxx:~/Bureau/tparith/exo10_3$
```

Exercice 10.4:

Dans le cas de l'arrondi vers plus l'infini, on obtient automatiquement une boucle infinie car la condition d'arrêt ne sera jamais satisfaite l'arrondi de x reste toujours supérieur à zéro.

```
2 raised to the power 4786975) = 1.401299e-45
2 raised to the power 4786976) = 1.401299e-45
2 raised to the power 4786977) = 1.401299e-45
2 raised to the power 4786978) = 1.401299e-45
2 raised to the power 4786979) = 1.401299e-45
```

Exercice 10.5:

Mathématiquement (1/x)*x = 1 quelque soit x réel. L'objectif de l'exercice est de définir la valeur de (1/x)*x toute on varianons la valeur de x et l'arrondi de cette opération.

Pour le cas d'arrondi au plus près, on obtient la première valeur différente de 1 l'itération = 41 avec une valeur de (1/x)*x = 9.999999e-01. Pour les autres types d'arrondi à la troisième itération on obtient des résultats différents de 1 soit un 9.999999e-01 pour les arrondis vers 0 et -inf ou bien 1.000001e+00 pour l'arrondi vers plus l'infinie.

Avec les arrondis vers " +inf, -inf, 0 " on perd de la précision rapidement, mais dans le cas d'arrondi au plus près (par défaut) la précision des calculs diminue lentement.

Remarque:

Le passage en double précision binary 64 (type de variable double) on gagne de la précision de calcul car les variables sont stockés sur 64 bits cela apparaître dans

l'exemple de la division successive de (1/10) par deux puis la multiplication de résultat fois deux (exo 10_3).

- L'exécution du code: Pour exécuter le code il faut juste modifier le type d'arrondi dans le programme (choisir un et mettre les autres en commentaire) et le lancer directement avec le make run.

```
Avec l'arrondi z = 1.000000e+00 , x = 35
Avec l'arrondi z = 1.000000e+00 , x = 36
Avec l'arrondi z = 1.000000e+00 , x = 37
Avec l'arrondi z = 1.000000e+00 , x = 38
Avec l'arrondi z = 1.000000e+00 , x = 39
Avec l'arrondi z = 1.000000e+00 , x = 40
Avec l'arrondi z = 9.999999e-01 , x = 41
```

```
./exo5.o
Avec l'arrondi z = 1.000000e+00 , x = 1, z de type float
Avec l'arrondi z = 1.000000e+00 , x = 2, z de type float
Avec l'arrondi z = 1.000001e+00 , x = 3, z de type float
```

```
gcc -g -std=c99 -Wall -O0 exo5.c -o exo5.o -lm
./exo5.o
Avec l'arrondi z = 1.000000e+00 , x = 1, z de type float
Avec l'arrondi z = 1.000000e+00 , x = 2, z de type float
Avec l'arrondi z = 9.999999e-01 , x = 3, z de type float
nadjib@nadjib-HP-Laptop-15s-eq0xxx:~/Bureau/tparith/exo10_5$
```

Exercice 10.6:

L'objectif de cette exercice est le même que l'exercice 10.5 est on obtient les résultat suivant:

1. L'arrondi au plus près: x = 7.
2. L'arrondi vers plus l'infinie x = 27.
3. L'arrondi vers zéro: x = 27.
4. L'arrondi vers plus l'infinie x = 29. D'où dans ce cas l'arrondi vers plus l'infinie est plus précis que les autres arrondies.

Le type d'arrondi influence sur les résultats de calculs car il y'a une différence entre les calculs mathématiques et les calculs machines.

Exercice 10.10:

INFINITY c'est l'infini prédéfinie dans la bibliothèque math.h.
Avec std=-c99 comme option , les résultats de calculs obtenue sont:

- log(+0): on obtient moins l'infini " -inf " et c'est la limite de Log quand x converge vers 0+ donc c'est une valeur approximative (nombre dénormalisé).
- log(-0): on obtient un overflow " nan " et c'est un résultat logique car la fonction Log n'est pas définie mathématiquement sur l'intervalle]-inf,0].
- log(+1): on obtient " 0 " et c'est la valeur de Log(1).

- log(-inf): on obtient un overflow “ nan ” et c’est un résultat logique car la fonction Log n’est pas définie mathématiquement sur l'intervalle]-inf,0].

```
nadjib@nadjib-HP-Laptop-15s-eq0xxx:~/Bureau/tparith/exo10_10$ ./exo10.o
La valeur de Log(+0) = -inf, -inf
La valeur de Log(+0) = -nan, -nan
La valeur de Log(+1) = 0.000000, 0.000000e+00
La valeur de Log(-1) = -nan, -nan
La valeur de Log(-inf) = -nan, -nan
La valeur de Log(+inf) = inf, inf
nadjib@nadjib-HP-Laptop-15s-eq0xxx:~/Bureau/tparith/exo10_10$
```

Exercice 10.11:

1. Les fonctions sin et cos sont définies sur R et leurs images sont comprises entre 1 et -1.
2. Les valeurs de sin(inf) et cos(inf) sont indéfinies car on peux pas prédire le comportement des ces fonctions, on obtient la valeur nan (not a number).
3. Le domaine de définition des fonctions arcsin et arccos est [-1;1], et l'image de la fonction arccos [0,pi] et pour arcsin [-pi/2;pi/2].
4. On essaye d’utiliser des valeurs hors le domaine de définition de ces fonctions et on obtient Nan (Not A Number).

Exercice de la fonction IEEE 754:

L'exécution de cette fonction se fait avec un Makefile. On a utilisé la compilation séparée pour faciliter l’utilisation de cette fonction dans les autres exercices.