

## CC hexad\_ecimal

### Question 1 :

Première étape, créer le fichier c avec le code à l'intérieur :

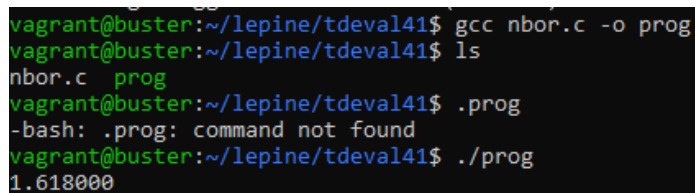
```
nano nbor.c
```

Ensuite, il faut compiler le programme

```
gcc nbor.c -o prog
```

Pour finir par l'exécuter : `./prog`

Ce qui nous donne :



```
vagrant@buster:~/lepine/tdeval41$ gcc nbor.c -o prog
vagrant@buster:~/lepine/tdeval41$ ls
nbor.c  prog
vagrant@buster:~/lepine/tdeval41$ ./prog
-bash: ./prog: command not found
vagrant@buster:~/lepine/tdeval41$ ./prog
1.618000
```

### Question 2 :

On remarque que l'on cast l'int en un float, mais il est alors normal de se demander comment on arrive sur le nombre d'or.

En réalité, on stock dans n une valeur en décimal : 1070537376

Lors du printf, on vient récupérer, non pas la valeur en décimal, mais la valeur binaire stocké à l'adresse de n afin de pouvoir la transformer en float. Or, la représentation d'un floatant se fait via la norme IEEE-754, c'est-à-dire que :

$$(1070537376)_{10} = (111111110011110001101010100000)_2 = (1.61800003052)_{\text{IEEE-754}}$$

C'est bien se dernier nombre qui est affiché.

Site de conversion utilisé :

Décimal -> binaire : <https://www.rapidtables.com/convert/number/decimal-to-binary.html>

Binaire -> floatant : <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

### Question 3 :

On fait l'inverse ici, on souhaite avoir (2021)<sub>IEEE-754</sub>

A l'aide du site de conversion, on remarque que ça équivaut à  $(01000100111111001010000000000000)_2$

C'est-à-dire :  $(1157406720)_{10}$

Aperçu du résultat :

```
#include <stdio.h>
int main() {
    int n = 1157406720;
    printf("%f\n",*(float*)&n);
}
```

```
vagrant@buster:~/lepine/tdeval41$ nano nbor.c
vagrant@buster:~/lepine/tdeval41$ gcc nbor.c -o prog
vagrant@buster:~/lepine/tdeval41$ ./prog
2021.000000
```

### Question 4 :

On sait que  $(1070537376)_{10} = (3FCF1AA0)_{16}$

Enter decimal number

10

ConvertResetSwap

Hex number

16

Avec la commande `hexedit prog` (prog est le code c compilé), on affiche le code hexadécimal de l'exécutable, puis on recherche 3F CF 1A A0

[illegible]

Une fois chose faite et après avoir pris une aspirine contre le mal de tête, il suffit de remplacer cette valeur par la valeur hexadécimale de  $(1157406720)_{10}$  c'est-à-dire :

$(44FCA000)_{16}$

```
FF E0 66 0F 1F 44 00 00 C3 0F 1F 80 00 00 00 00 80 3D 39 2F 00 00 00 75 2F 55 48 83 3D F6 2E 00 00 00 48 89 E5 74 0C 48 ..  
B8 3D 1A 2F 00 00 E8 2D FF FF FF E8 68 FF FF FF C6 05 11 2F 00 00 01 5D C3 0F 1F 80 00 00 00 00 C3 0F 1F 80 00 00 00 00 .=  
E9 7B FF FF FF 55 48 89 E5 48 83 EC 10 C7 45 FC 00 A0 FC 44 48 8D 45 FC F3 0F 10 00 F3 0F 5A C0 48 8D 3D AD 0E 00 00 B8 ..{  
01 00 00 00 E8 CF FE FF FF B8 00 00 00 00 C9 C3 0F 1F 84 00 00 00 00 00 41 57 49 89 D7 41 56 49 89 F6 41 55 41 89 FD 41 ...  
54 4C 8D 25 60 2C 00 00 55 48 8D 2D 60 2C 00 00 53 4C 29 E5 48 83 EC 08 E8 63 FE FF FF 48 C1 FD 03 74 1B 31 DB 0F 1F 00 TL  
4C 89 FA 4C 89 F6 44 89 EF 41 FF 14 DC 48 83 C3 01 48 39 DD 75 EA 48 83 C4 08 5B 5D 41 5C 41 5D 41 5E 41 5F C3 0F 1F 00 L.  
C3 00 00 00 48 83 EC 08 48 83 C4 08 C3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
```

On enregistre puis on relance l'exécutable :

```
vagrant@buster:~/lepine/tdeval41$ hexedit prog  
vagrant@buster:~/lepine/tdeval41$ vagrant@buster:~/lepine/tdeval41$ ./prog  
2021.000000
```

### Question 5 :

On cherche à réaliser un programme permettant de rentrer une valeur, la transformer en binaire, puis en décima, puis hexa, et enfin, de modifier le code hexadécimal de l'exécutable.

```
from decimal import Decimal  
  
value = input('Entrez la valeur de sortie souhaitée : ') # on demande la valeur  
r  
  
value_in_dec = Decimal(value.replace(',', '.')) # valeur en décimal  
value_in_bin = bin(value_in_dec) # valeur en binaire  
  
# ...
```

Plus le temps de terminer ...