C for system

1 Environnement C

Récupérez le fichier hello.c, compilez le et testez le.

```
gcc -o  hello hello.c
hello
```

ou

```
gcc  hello.c
a.out
```

Modifiez le message du *printf* et tester votre programme.

2 PrintBin

En C il n'existe pas de fonction dans la librairie standard pour afficher un nombre en binaire. Écrire en un programme *printBin.c* d'affichage d'un entier non signé sous sa forme binaire. Pour cela il faudra utiliser des opérateurs binaires comme | & ~^ » «

```
#include<stdio.h>
int main(){
   unsigned val = 0x3F0E;
   ...
   //0000 0000 0000 0000 0011 1111 0000 1110
```

Pour afficher les 0 ou 1 utilisez la fonction putchar('O') ou putchar('1')

Votre programme doit s'adapter au type utilisé unsigned char, unsigned, short unsigned, lonq unsigned

3 Décalage circulaire

Écrire un programme qui, à partir d'un entier naturel, effectue une opération de décalage circulaire à gauche de n bits. (le résultat sera affiché en héxadécimal.) exemple : U decal G

unsigned U = 0xA2453F0E (10100010 01000101 00111111 00001110)

G=8 (décalage circulaire à gauche de 8 bits)

résultat -> 453F0EA2 (01000101 00111111 00001110 10100010)

Dans une première version les décalages circulaires s'effectueront bit par bit.

L'affichage s'effectue par un

```
printf("%X\n",result)
```

Écrire une nouvelle version en faisant le décalage par bloc (sans itération dans votre programme)

4 Complément à 2

Le complément à 2 sur les entiers signés permet d'effectuer les additions avec les nombres négatifs. Il est utilisé pour coder les entiers négatifs

Pour coder (-4) :

— on prend le nombre positif 4 : 00000100 ;

— on inverse les bits : 11111011 ;

— on ajoute 1 : 11111100.

Si l'on doit transformer un nombre en son complément à deux sans faire l'addition, la solution est de garder tous les bits depuis la droite jusqu'au premier 1 (compris) puis d'inverser tous les suivants.

— Prenons par exemple le nombre 20 : 00010100.

— On inverse la partie de gauche après le premier un : **11101**100.

— On garde la partie à droite telle quelle : (00010**100**).

— Et voici -20 : 11101100.

```
char vingt=20;
...
// A completer  .... complement à 2
...
printf("-20  %hhd\n", vingt)
```

5 Ensemble de booléens

Un élément de type unsigned (sur 4 octets) est sur 32 bits et peut donc contenir 32 booléens. (tableau de 32 booléens).

Pour cet exercice, nous utiliserons une variable globale, même si cela n'est pas souhaitable.

```
unsigned bools;
```

Écrire la fonction initBools(unsigned char val) qui initialise les 32 booléens contenus dans *bools* à False (0) ou Vrai(1) (val==0 False).

```
unsigned bools;
int main(){
initBools( (unsigned char) 0 );
}
```

Reprendre le programme printBin pour en faire une fonction printBools qui affiche les bits contenu dans le variable *bools*.

```
unsigned bools;
int main(){
initBools( (unsigned char) 1 );
printBools();
// 11111111111111111111111111111111
}
```

Écrire la fonction int setBools(int nb, unsigned char val) qui met False 0 ou True 1 (val) dans le booléen contenu dans *bools* à la position nb. Le booléen numéro 0 est mis dans bit le plus à droite (de poids le plus faible). La fonction retourne 0 en cas de succès.

```
unsigned bools;
int main(){
initBools( (unsigned char) 0 );
printBools()
// 00000000000000000000000000000000
setBools( 2,(unsigned char)1);
printBools()
// 00000000000000000000000000000100
}
```

Écrire la fonction unsigned char isBools(int nb) qui retourne True ou False 0 si le booléen contenu dans *bools* à la position nb est vrai ou faux.

```
unsigned bools;
int main(){
initBools( (unsigned char) 0 );
printBools()
// 00000000000000000000000000000000
setBools( 2,(unsigned char)1);
printBools()
// 00000000000000000000000000000100
printf("isbools %u\n",isBools(2)!=0);
// isbools 1
}
```

6 La multiplication

Écrire un programme qui saisit deux entiers op1 et op2 (non-signés et < 256) puis calcule le produit op1 * op2 en utilisant les opérateurs manipulant leur représentation binaire. (sans utiliser *, /, %) Vous pouvez utiliser les opérateurs de décalage « et » ainsi que l'addition.

Exemple

op1 <- 00010001 (17)
op2 <- 00000101 (5)
produit 01010101 (85)

On va, pour cela, utiliser les puissances de 2.

op2 = 5 = 101 = $4(2^2) + 1(2^0)$

sachant d'un décalage à gauche op1 est une multiplication par 2 :

17*5 = 17*4 + 17. avec 17*4 = 17 « 2

17*5 = (17 « 2) + 17

Algorithme :

op2	op1		produit intermédiaire
0000 0101	00010001		0
extraction de bit	multiple de 2		
– - - - - -1	00010001	->	00010001 +
– - - - - 0-	00100010	->	0 +
– - - - 1 –	01000100	->	01000100 +
			=======
			01010101 (résultat)

7 La division

Écrire un programme qui saisit deux entiers op1 et op2 (non-signés et < 256) puis calcule le résultat de la division entière op1 // op2 en utilisant les opérateurs manipulant leur représentation binaire. (sans utiliser *, /, %) Vous pouvez utiliser les opérateurs de décalage « et » ainsi que l'addition et soustraction.

Exemple

op1 <- 00010110 (22)
op2 <- 00000101 (3)

Algorithme :

op1 0001 0110	op2 00000011	result 0	traitement
0		0	op1 < op2
00		00	op1 < op2
0001		000	op1 < op2
0001 0		0000	op1 < op2
0001 01			op1 >= op2
0000 10		00001	op1=op1-op2 (5-3=2)
0000 101			op1 >= op2
0000 010		000011	op1=op1-op2 (5-3=2)
0000 0100			op1 >= op2
0000 0001		00000111	op1=op1-op2 (4-3=1)