Exercices sur les masques

1 . Donner une déclaration en C de la variable **commandeChauffages**.
Réponse : **`unsigned char commandeChauffages;`**

2 . Ecrire la commande pour **activer seulement le chauffage n°3**.
Réponse :

	7	6	5	4	3	2	1	0	
	X	***X***	***X***	***X***	***0***	***X***	***X***	***X***	***= état précédent***
opérateur : |	0	0	0	0	1	0	0	0	= masque (0x08)
	X	X	X	X	1	X	X	X	= activation du chauffage n°3

En langage C :
```
commandeChauffages = commandeChauffages | 0x08;
// ou :
commandeChauffages |= 0x08;
```

L'opérateur | permet de forcer un ou plusieurs bits à 1.

3 . Ecrire la commande pour **éteindre seulement le chauffage n°6**.
Réponse :

	7	6	5	4	3	2	1	0	
	X	***1***	***X***	***X***	***X***	***X***	***X***	***X***	***= état précédent***
opérateur : **&**	1	0	1	1	1	1	1	1	= masque (0xBF)
	X	0	X	X	X	X	X	X	= activation du chauffage n°6

En langage C :
```
commandeChauffages = commandeChauffages & 0xBF;
// ou :
commandeChauffages &= 0xBF;
```

L'opérateur & permet de forcer un ou plusieurs bits à 0.

Remarque : l'écriture de valeur sans signification dans un programme est à proscrire pour des raisons de *debugage*, de maintenance et d'évolution (par exemple 0xFE ??)
4 . Suite à la remarque précédente, proposer une (ou des) solutions pour éviter d'utiliser des valeurs sans signification.
5 . Que réalise ces lignes de codes ? Compléter le tableau ci-dessous.
Réponse :

```
#define ARRET_CHAUFFAGES      0x00
#define CHAUFFAGE_3           0x08
#define CHAUFFAGE_2           0x04
#define CHAUFFAGE_1           0x02
#define CHAUFFAGE_0           0x01
```

```
commandeChauffages = ARRET_CHAUFFAGES;
commandeChauffages |= CHAUFFAGE_3;
commandeChauffages &= ~CHAUFFAGE_3;
commandeChauffages |= (CHAUFFAGE_1 | CHAUFFAGE_0);
commandeChauffages &= ~(CHAUFFAGE_1 | CHAUFFAGE_0);
commandeChauffages |= CHAUFFAGE_2;
```

7	6	5	4	3	2	1	0	en hexa	*Action*
0	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***= 0x00***	***état initial à l'arrêt***
0	0	0	0	1	0	0	0	= 0x08	activation du chauffage n°3
0	0	0	0	0	0	0	0	= 0x00	arrêt du chauffage n°3
0	0	0	0	0	0	1	1	= 0x03	activation des chauffages n°0 et 1
0	0	0	0	0	0	0	0	= 0x00	arrêt des chauffages n°0 et 1
0	0	0	0	0	1	0	0	= 0x04	activation du chauffage n°2

Le code gagne en lisibilité !

On désire maintenant connaître l'état des chauffages individuellement.
6 . Donner une déclaration en C de la variable **etatChauffages**.
Réponse : **`unsigned char etatChauffages;`**

7 . Afficher l'état logique du **chauffage n°2**.
Réponse :

	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	
	X	***X***	***X***	***X***	***X***	0/1	***X***	***X***	***= état des chauffages***
opérateur : **&**	0	0	0	0	0	1	0	0	= masque (0x04) = **`CHAUFFAGE_2`**
	0	0	0	0	0	0/1	0	0	= état du chauffage n°2
opérateur : **>>**						->	->		décalage à droite de 2 bits
	0	0	0	0	0	0	0	0/1	= état <u>booléen</u> du chauffage n°2

En langage C :

```
if( ((etatChauffage & CHAUFFAGE_2) >> 2) == 1 )
        printf("chauffage n°2 : allumé\n");
else    printf("chauffage n°2 : éteint\n");
```

8 . Ecrire la fonction **fabriquerCommandeChauffage()** qui **reçoit en paramètres** le numéro de chauffage (0 à 7) et l'action à réaliser (0 ou 1) et qui **retourne** la valeur de la commande.
Réponse :

```
#define ARRET     0
#define MARCHE    1
#define OFF       0
#define ON        1
typedef unsigned char BYTE; // définition du type BYTE (c'est un unsigned char)

unsigned char fabriquerCommandeChauffage(int numChauffage, int action)
{
      unsigned char commandeChauffages, etat, masque;

      etat = lire(); // cette fonction peut être simulée pour des tests

      masque = (1 << numChauffage);
      if(action == ON)
            commandeChauffages = etat | masque;
      else  commandeChauffages = etat & ~masque;
      return commandeChauffages;
}
```

9 . Dans la fonction suivante, déterminer le rôle et l'utilisation des deux paramètres **choix** et **cmd** :

```
typedef unsigned char BYTE; // définition du type BYTE (c'est un unsigned char)

BYTE fabriquerCommande2Chauffage(BYTE choix, BYTE cmd)
{
   BYTE etat, commande;

   etat = lire(); // cette fonction peut être simulée pour des tests

   commande = etat & ~choix ;
   commande = commande | (choix & cmd);

   return commande;
}
```

Réponse :

	7	6	5	4	3	2	1	0	
etat	X	X	X	X	X	X	X	X	
choix	0	1	0	0	1	0	0	0	0x48 : on sélectionne les bits 6 et 3
~choix	1	0	1	1	0	1	1	1	
etat & ~choix	X	0	X	X	0	X	X	X	on ne touche pas les bits non choisis
cmd	0	0	0	0	1	0	0	0	0x08 : on commande les bits 6 (= 0) et 3 (= 1)
choix & cmd	0	0	0	0	1	0	0	0	
commande | (choix & cmd)	X	0	X	X	1	X	X	X	résultat : bit 6 = 0 et bit 3 = 1

choix : les bits à 1 de cette variable indique les chauffages que l'on désire commander

cmd : les bits correspondants à choix donnent les commandes à réaliser

10 . Comparer les deux fonctions réalisées **fabriquerCommandeChauffage** et **fabriquerCommande2Chauffage**.

Réponse :

La principale différence se situe au niveau du nombre de chauffages commandés : en effet, la fonction **fabriquerCommandeChauffage** ne permet de commander qu'un seul chauffage à la fois. La fonction **fabriquerCommande2Chauffage** permet de commander (allumer et/ou éteindre) autant de chauffage que l'on sélectionne.