Cours Langage C/C++ - Les masques

Thierry Vaira

BTS IRIS Avignon

tvaira@free.fr v0.1

Rappel : Opérateurs logique et bit à bit (1/2)

Ne pas confondre les opérateurs logiques avec les opérateurs bit à bit

```
unsigned char a = 1; unsigned char b = 0;
unsigned char aa = 20; /* non nul donc VRAI en logique */
unsigned char bb = 0xAA;

// Ne pas confondre !
/* ! : inverseur logique */
/* ~ : inverseur bit à bit */
printf("a = %u - !a = %u - ~a = %u (0x%hhX)\n", a, !a, ~a, ~a);
printf("b = %u - !b = %u - ~b = %u (0x%hhX)\n", b, !b, ~b, ~b);

printf("aa = %u (0x%hhX) - !aa = %u - ~aa = %u (0x%hhX)\n", aa, aa, !
    aa, ~aa, ~aa);
printf("bb = %u (0x%hhX) - !bb = %u - ~bb = %u (0x%hhX)\n", bb, bb, !
    bb, ~bb, ~bb);
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Rappel : Opérateurs logique et bit à bit (2/2)

- Pour les opérateurs bit à bit, il est conseillé d'utiliser la représentation en hexadécimale :

Exemple

```
a = 1 - !a = 0 - ~a = 4294967294 (0xFE)

b = 0 - !b = 1 - ~b = 4294967295 (0xFF)

aa = 20 (0x14) - !aa = 0 - ~aa = 4294967275 (0xEB)

bb = 170 (0xAA) - !bb = 0 - ~bb = 4294967125 (0x55)
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Les opérateurs de masque

- Un masque désigne des données utilisées pour des **opérations bit à bit**, permettant de modifier la (ou les) valeur(s) d'un bit (ou un groupe) en une seule opération sans modifier les autres bits.
- *Remarque :* la connaissance du binaire et de l'hexadécimal est indispensable pour manipuler des masques en informatique industrielle. On rappelle qu'un symbole hexadécimal (0 à F) représente un groupe de 4 bits (2^3 à 2^0).
- Pour réaliser des masque, on s'appuie sur les opérateurs suivants :
  - le **ET** (AND) bit à bit (&)
  - le **OU** (OR) bit à bit (|)
  - le **OU EXCLUSIF** (XOR) bit à bit (^)
  - le décalage de n bit (rang) vers la droite (»)
  - le décalage de n bit (rang) vers la gauche («)
  - l'inversion bit à bit (~)

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Tables de vérité (1/2)

Avant de pouvoir les combiner et les utiliser pour réaliser des masques bit à bit, il faut tout d'abord maîtriser leurs propriétés (table de vérité).

```
#include <stdio.h>
int main() {
   printf("Table de vérité du ET (&) :\n");
   printf("0 & 0 = %d\n", 0 & 0);
   printf("0 & 1 = %d\n", 0 & 1);
   printf("1 & 0 = %d\n", 1 & 0);
   printf("1 & 1 = %d\n\n", 1 & 1);
   printf("Table de vérité du OU (|) :\n");
   printf("0 | 0 = %d\n", 0 | 0);
   printf("0 | 1 = %d\n", 0 | 1);
   printf("1 | 0 = %d\n", 1 | 0);
   printf("1 | 1 = %d\n\n", 1 | 1);
   printf("Table de vérité du OU EXCLUSIF (^) :\n");
   printf("0 ^ 0 = %d\n", 0 ^ 0);
   printf("0 ^ 1 = %d\n", 0 ^ 1);
   printf("1 ^ 0 = %d\n", 1 ^ 0);
   printf("1 ^ 1 = %d\n\n", 1 ^ 1);
   return 0;
}
```

Tables de vérité (2/2)

Les tables de vérité des opérateurs ET, OU et OU EXCLUSIF sont :

```
Table de vérité du ET (&) :
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

Table de vérité du OU (|) :
0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

Table de vérité du OU EXCLUSIF (^) :
0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Élément neutre et absorbant

- Le plus intéressant est d'analyser maintenant leur comportement.
- Pour le **ET** bit à bit :
  - le 0 est l'élément absorbant : il y aura un 0 en sortie quelque soit la valeur de l'autre opérande
  - le 1 est l'élément neutre : la valeur de l'autre opérande est inchangée
- Pour le **OU** bit à bit :
  - le 0 est l'élément neutre : la valeur de l'autre opérande est inchangée
  - le 1 est l'élément absorbant : il y aura un 1 en sortie quelque soit la valeur de l'autre opérande
- Pour le **OU EXCLUSIF** bit à bit :
  - le 0 est l'élément neutre : la valeur de l'autre opérande est inchangée
  - le 1 est l'élément inverseur : la valeur de l'autre opérande est inversée

Principe d'un masque

- On en déduit que pour **forcer l'état d'un bit** :
  - à 0 : il faut utiliser un **ET** (&)
  - à 1 : il faut utiliser un **OU** (|)
  - à l'inverse : il faut utiliser un OU EXCLUSIF (^)
- Si on utilise un de ces opérateurs pour forcer l'état d'un bit, il faudra pour **ne pas modifier l'état des autres bits** utiliser :
  - un 1 avec un **ET** (&)
  - un 0 avec un **OU** (|)
  - un 0 avec un OU EXCLUSIF (^)

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Comportement des opérateurs », « et ~(1/2)

Avant d'aller plus loin, observons le comportement des autres opérateurs : », « et ~.

```
#include <stdio.h>
int main() {
   unsigned char valeur; int n;
   printf("Comportement de l'inverseur bit à bit :\n");
   printf("~0x00 -> 0x%02X\n", (unsigned char)~0x00);
   printf("~0x01 -> 0x%02X\n", (unsigned char)~0x01);
   printf("\nUtilisation de l'opérateur << : un chenillard\n");
   for(n=0;n<8;n++) {
      valeur = 1 << n;
      printf("valeur = %u ou 0x%02X (<< décalage de %d bit(s) à gauche)\n",
           valeur, valeur, n);
   }
   printf("\nUtilisation de l'opérateur >> : un chenillard\n");
   for(n=0;n<8;n++) {
      valeur = 0x80 >> n;
      printf("valeur = %u ou 0x%02X (>> décalage de %d bit(s) à droite)\n",
           valeur, valeur, n);
   }
   return 0;
}
```

Comportement des opérateurs », « et ~(2/2)

```
Comportement de l'inverseur bit à bit :
~0x00 -> 0xFF
~0x01 -> 0xFE
Utilisation de l'opérateur << : un chenillard
valeur = 1 ou 0x01 (<< décalage de 0 bit(s) à gauche)
valeur = 2 ou 0x02 (<< décalage de 1 bit(s) à gauche)
valeur = 4 ou 0x04 (<< décalage de 2 bit(s) à gauche)
valeur = 8 ou 0x08 (<< décalage de 3 bit(s) à gauche)
valeur = 16 ou 0x10 (<< décalage de 4 bit(s) à gauche)
valeur = 32 ou 0x20 (<< décalage de 5 bit(s) à gauche)
valeur = 64 ou 0x40 (<< décalage de 6 bit(s) à gauche)
valeur = 128 ou 0x80 (<< décalage de 7 bit(s) à gauche)
Utilisation de l'opérateur >> : un chenillard
valeur = 128 ou 0x80 (>> décalage de 0 bit(s) à droite)
valeur = 64 ou 0x40 (>> décalage de 1 bit(s) à droite)
valeur = 32 ou 0x20 (>> décalage de 2 bit(s) à droite)
valeur = 16 ou 0x10 (>> décalage de 3 bit(s) à droite)
valeur = 8 ou 0x08 (>> décalage de 4 bit(s) à droite)
valeur = 4 ou 0x04 (>> décalage de 5 bit(s) à droite)
valeur = 2 ou 0x02 (>> décalage de 6 bit(s) à droite)
valeur = 1 ou 0x01 (>> décalage de 7 bit(s) à droite)
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Forçage d'une sortie (1/2)

- En informatique industrielle, on manipule souvent des E/S TOR (Tout Ou Rien) qui permettent de commander des actionneurs.
- Par exemple avec 8 sorties TOR, on pourra commander individuellement 8 chauffages en marche/arrêt. Évidemment, pour permettre d'allumer ou éteindre un chauffage sans toucher aux autres, on devra utiliser des masques. Attention, le bit de poids faible est numérotée 0 (2^0).
- Pour forcer la sortie n°3 à l'état 1, il faudra donc utiliser un **OU** :
  - commande précédente → `xxxx` `0xxx` (la sortie n°3 est à l'état 0, les autres sorties ne doivent pas être modifiées)
  - **masque** avec un **OU** → `0000` `1000` (la sortie n°3 est forcée à l'état 1, les autres sorties ne sont pas modifiées par le 0)
  - nouvelle commande → `xxxx` `1xxx` (la sortie n°3 est passée à l'état 1, les autres sorties n'ont pas été modifiées)

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Forçage d'une sortie (2/2)

- Pour forcer la sortie n°6 à l'état 0, il faudra donc utiliser un **ET** :
  - commande précédente → `x1xx xxxx` (la sortie n°6 est à l'état 1, les autres sorties ne doivent pas être modifiées)
  - **masque** avec un **ET** → `1011 1111` (la sortie n°6 est forcée à l'état 0, les autres sorties ne sont pas modifiées par le 1)
  - nouvelle commande → `x0xx xxxx` (la sortie n°6 est passée à l'état 0, les autres sorties n'ont pas été modifiées)
- Pour inverser la sortie n°1, il faudra donc utiliser un **OU EXCLUSIF** :
  - commande précédente → `xxxx xx?x` (la sortie n°1 est à l'état 0 ou 1, les autres sorties ne doivent pas être modifiées)
  - **masque** avec un **XOR** → `0000 0010` (la sortie n°1 est sera inversée, les autres sorties ne sont pas modifiées par le 0)
  - nouvelle commande → `xxxx xx~x` (la sortie n°1 a été inversée, les autres sorties n'ont pas été modifiées)

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Commande individuelle de 8 sorties TOR (1/3)

```c
#include <stdio.h>

/* Fonction permettant de visualiser un octet sous sa forme binaire */
/* Convention usuelle : 2 signifie to (vers en français) */
void octet2bit(unsigned char octet)
{
   int rang = 0x80; // soit 1000 0000
   int i;

   for(i = 0; i < 8; i++)
   {
      if(octet & (rang >> i)) // >> décalage à droite de i bits
            printf("1");
      else  printf("0");
   }
   printf("\n");
}
```

Commande individuelle de 8 sorties TOR (2/3)

```
int main() { unsigned char masque;
   unsigned char commandes8TOR = 0xE5; // simulons un situation initiale
   printf("Forcer la sortie n°3 à l'état 1, on utilise un OU (|) :\n");
   printf("commande précédente -> "); octet2bit(commandes8TOR);
   masque = 0x08; // 0000 1000
   printf("masque avec un OU -> "); octet2bit(masque);
   commandes8TOR = commandes8TOR | masque; // ou : commandes8TOR |= masque;
   printf("nouvelle commande -> "); octet2bit(commandes8TOR); printf("\n");
   printf("Forcer la sortie n°6 à l'état 0, on utilise un ET (&) :\n");
   printf("commande précédente -> "); octet2bit(commandes8TOR);
   masque = 0xBF; // 1011 1111
   printf("masque avec un ET -> "); octet2bit(masque);
   commandes8TOR = commandes8TOR & masque; // ou : commandes8TOR &= masque;
   printf("nouvelle commande -> "); octet2bit(commandes8TOR); printf("\n");
   printf("Inverser la sortie n°1, on utilise un OU EXCLUSIF (^) :\n");
   printf("commande précédente -> "); octet2bit(commandes8TOR);
   masque = 0x02; // 0000 0010
   printf("masque avec un XOR -> "); octet2bit(masque);
   commandes8TOR = commandes8TOR ^ masque; // ou : commandes8TOR ^= masque;
   printf("nouvelle commande -> "); octet2bit(commandes8TOR);
   return 0;
}
```

Commande individuelle de 8 sorties TOR (3/3)

```
Forcer la sortie n°3 à l'état 1, on utilise un OU (|) :
commande précédente -> 11100101
masque avec un OU -> 00001000
nouvelle commande -> 11101101

Forcer la sortie n°6 à l'état 0, on utilise un ET (&) :
commande précédente -> 11101101
masque avec un ET -> 10111111
nouvelle commande -> 10101101

Inverser la sortie n°1, on utilise un OU EXCLUSIF (^) :
commande précédente -> 10101101
masque avec un XOR -> 00000010
nouvelle commande -> 10101111
```

Lecture d'une entrée (1/3)

- En informatique industrielle, on manipule souvent des E/S TOR (Tout Ou Rien) qui permettent de lire les états booléens de capteurs.
- Par exemple avec 8 entrées TOR, on pourra connaître individuellement l'état de 8 interrupteurs marche/arrêt. Évidemment, pour traiter l'état d'un bouton de manière isolée, on devra utiliser des masques. Comme il est plus facile de traiter un état logique booléen (VRAI/FAUX, MARCHE/ARRET soit 0 ou 1), on fera des décalages à gauche pour placer l'état binaire d'une entrée sur le poids faible (le bit n°0).
- Déterminons l'état logique (booléen) de l'entrée n°2 :
  - état courant → `xxxx x?xx` (quel est l'état logique de l'entrée n°2 ?)
  - **masque** avec un **ET** → 0000 0100 (on place un 0 sur les entrées qui ne nous intéressent pas et un 1 sur l'entrée n°2)
  - état binaire → 0000 0?00 (l'état de l'entrée n°2 est isolé après l'opération de masque)
  - état logique → 0000 000? (décalage de 2 bits et on a maintenant l'état logique 0 ou 1 de l'entrée n°2)

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Lecture d'une entrée (2/3)

```
int main() { unsigned char masque;
  unsigned char entrees8TOR = 0xE5; // simulons un situation initiale
  unsigned char entreeN2; // état logique de l'entrée n°2

  printf("Déterminer l'état logique de l'entrée n°2, on utilise un ET (&) :\n");
  printf("état des 8 entrées          -> "); octet2bit(entrees8TOR);
  masque = 0x04; // 0000 0100
  printf("masque avec un ET           -> "); octet2bit(masque);
  entrees8TOR = entrees8TOR & masque; // ou : entrees8TOR &= masque;
  printf("état de l'entrée n°2        -> "); octet2bit(entrees8TOR);
  entreeN2 = entrees8TOR >> 2;
  printf("état logique de l'entrée n°2 -> "); octet2bit(entreeN2); printf("\n");

  /* on peut maintenant faire des tests logiques */
  if(entreeN2)
       printf("L'entrée n°2 est à 1 !");
  else printf("L'entrée n°2 est à 0 !");
  printf("\n");
  return 0;
}
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Lecture d'une entrée (3/3)

```
Déterminer l'état logique de l'entrée n°2, on utilise un ET (&) :
état des 8 entrées            -> 11100101
masque avec un ET             -> 00000100
état de l'entrée n°2          -> 00000100
état logique de l'entrée n°2 -> 00000001

L'entrée n°2 est à 1 !
```

Il est indispensable de posséder des fonctions prêtes à l'emploi pour manipuler des E/S TOR. On va écrire une fonction `fabriquerCommandes8TOR()` qui retourne la valeur de la commande et qui reçoit en paramètres :

- l'état précédent des sorties
- le numéro de sortie (0 à 7) à commander
- l'action à réaliser (0 ou 1)

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon

Une fonction prête à l'emploi

```
// Cela rendra aussi le code source plus lisible
#define ARRET   0
#define MARCHE  1

// Pour améliorer encore la lisibilité :
typedef unsigned char OCTET; // définition du type OCTET (8 bits)

OCTET fabriquerCommandes8TOR(OCTET commandes8TORPrecedentes, int numeroSortie,
     int action)
{
   OCTET commandes8TOR, masque;

   masque = (1 << numeroSortie);
   if(action == MARCHE)
        commandes8TOR = commandes8TORPrecedentes | masque;
   else  commandes8TOR = commandes8TORPrecedentes & ~masque;

   return commandes8TOR;
}
```

Frères des Ecoles Chrétiennes Saint-Jean-Baptiste de la Salle - Avignon