Opérateurs bit à bit en langage C

Par Emmanuel Delahaye

Date de publication : 28 janvier 2008

Dernière mise à jour : 7 mars 2013

Cet article a pour but de présenter les opérateurs bit à bit et leurs utilisations en langage C.

Votre avis et vos suggestions sur cet article nous intéressent !

Alors après votre lecture, n'hésitez pas : ***Commentez*** ♪

I - Description des opérateurs bits à bits....3
I-A - Introduction....3
I-B - NOT (NON)....3
I-C - AND (ET)....3
I-D - OR (OU)....4
I-E - XOR (OU exclusif)....4
I-F - SHR (Décalage à droite)....4
I-G - SHL (Décalage à gauche)....4
II - Usages des opérateurs bits à bits....6
II-A - Manipulations de l'état des bits d'une variable....6
II-B - Positionner un bit à 1....6
II-C - Positionner un bit à 0....6
II-D - Tester la valeur d'un bit....7
II-E - Conclusion....7

I - Description des opérateurs bits à bits

I-A - Introduction

Toute donnée informatique est stockée en mémoire sous la forme d'une combinaison de bits. Par exemple un entier valant 10 (base 10) implémenté par une mémoire d'une largeur de 16-bit contient :

```
0000 0000 0000 1010
```

Soit en hexadécimal :

```
000A
```

Les opérateurs bits permettent de modifier et de tester un ou plusieurs bits d'une donnée. Ces opérateurs sont :

- NOT (NON) ;
- AND (ET) ;
- OR (OU) ;
- XOR (OU exclusif) ;
- SHR (décalage à droite) ;
- SHL (décalage à gauche).

I-B - NOT (NON)

L'opérateur unaire NOT inverse l'état d'un bit selon le tableau suivant :

A	NOT A
0	1
1	0

L'opérateur C est ~. Il agit sur chaque bit de la valeur :

```
unsigned a = 1;
unsigned b = ~a; /* b == 1111111111111110  (en supposant 16-bit) */
```

I-C - AND (ET)

L'opérateur binaire AND combine l'état de 2 bits selon le tableau suivant :

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

L'opérateur C est &. Il agit sur chaque bit de la valeur :

```
unsigned a = 0xF0F0;
unsigned b = 0x00FF;
unsigned c = a & b; /* c == 0000000011110000 soit 0x00F0 */
```

I-D - OR (OU)

L'opérateur binaire OR combine l'état de 2 bits selon le tableau suivant :

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

L'opérateur C est |. Il agit sur chaque bit de la valeur :

```
unsigned a = 0xF0F0;
unsigned b = 0x00FF;
unsigned c = a | b; /* c == 1111 0000 1111 1111 soit 0xF0FF */
```

I-E - XOR (OU exclusif)

L'opérateur binaire OR combine l'état de 2 bits selon le tableau suivant :

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

L'opérateur C est ^. Il agit sur chaque bit de la valeur :

```
unsigned a = 0xF0F0;
unsigned b = 0x00FF;
unsigned c = a ^ b; /* c == 1111 0000 0000 1111 soit 0xF00F */
```

I-F - SHR (Décalage à droite)

L'opérateur binaire SHR a pour opérande de gauche la valeur initiale et pour opérande de droite le nombre de bits à décaler à droite. Les bits de poids faibles sont perdus et les bits de poids forts entrés (à gauche) sont à 0. **Ce n'est pas une rotation**.

L'opérateur C est >>. Il agit sur chaque bit de la valeur :

```
unsigned a = 0xF0F0;
unsigned b = 2;
unsigned c = a >> b; /* c == 0011 1100 0011 1100 soit 0x3C3C */
```

I-G - SHL (Décalage à gauche)

L'opérateur binaire SHL a pour opérande de gauche la valeur initiale et pour opérande de droite le nombre de bits à décaler à gauche. Les bits de poids forts sont perdus et les bits de poids faibles entrés (à droite) sont à 0. **Ce n'est pas une rotation**.

L'opérateur C est `<<`. Il agit sur chaque bit de la valeur :

```
unsigned a = 0xF0F0;
unsigned b = 2;
unsigned c = a << b; /* c == 1100 0011 1100 0000 soit 0xC3C0 */
```

II - Usages des opérateurs bits à bits

II-A - Manipulations de l'état des bits d'une variable

Si la variable est entière et non signée, il est possible d'utiliser les opérateurs bits pour mettre un ou des bits à 0 ou à 1. Les usages connus sont :

- compression de données ;
- commande et état de registres matériels ;
- etc.

II-B - Positionner un bit à 1

Le principe est de combiner la valeur avec un masque grâce à l'opérateur OU. En effet, comme l'indique la table de vérité, les bits à 0 du masque vont laisser la valeur initiale inchangée, alors les bits à 1 vont s'imposer.

```
/* mettre à 1 le bit 4 de b : */
   unsigned a = 0x000F; /* 0000 0000 0000 1111 */
   unsigned b = 0x0010; /* 0000 0000 0001 0000 */
   unsigned c = a | b;  /* 0000 0000 0001 1111 soit 0x001F */

   printf ("%04X OU %04X = %04X\n, a, b, c);
```

Pour fabriquer le masque, il suffit d'utiliser un 1 que l'on décale à gauche de la valeur correspondante au poids du bit. Par exemple :

```
Bit 0  : 1u << 0  = 0000 0000 0000 0001
Bit 2  : 1u << 2  = 0000 0000 0000 0100
Bit 15 : 1u << 15 = 1000 0000 0000 0000
```

Comme pour toute manipulation de bits (y compris avec des constantes), on utilise des valeurs non signées (d'où le 'u').

II-C - Positionner un bit à 0

Le principe est de combiner la valeur avec un masque grâce à l'opérateur ET. En effet, comme l'indique la table de vérité, les bits à 1 du masque vont laisser la valeur initiale inchangée, alors les bits à 0 vont s'imposer.

```
/* mettre à 0 le bit 3 de b : */
   unsigned a = 0x000F; /* 0000 0000 0000 1111 */
   unsigned b = 0xFFF7; /* 1111 1111 1111 0111 */
   unsigned c = a & b;  /* 0000 0000 0000 0111 soit 0x0007 */

   printf ("%04X OU %04X = %04X\n, a, b, c);
```

Pour fabriquer le masque, il suffit d'utiliser un 1 que l'on décale à gauche de la valeur correspondante au poids du bit, puis on inverse les bits avec l'opérateur NON. Par exemple :

```
Bit 0  : ~(1u << 0)  = 1111 1111 1111 1110
Bit 2  : ~(1u << 2)  = 1111 1111 1111 1011
Bit 15 : ~(1u << 15) = 0111 1111 1111 1111
```

II-D - Tester la valeur d'un bit

Le principe est d'évaluer le résultat entre la valeur à tester d'une part et un masque à 0, sauf le bit à tester, avec l'opérateur AND. Les bits a 0 restent à 0. Le bit à 1 passe à 1 si la valeur lue est 1, sinon, il reste à 0. Si le résultat est 0, le bit est donc à 0. S'il n'est pas 0, il est à 1.

```
/* tester l'état du bit 2 de a : */
   unsigned a = 0x000F; /* 0000 0000 0000 1111 */

   if (a & (1u << 2))
   {
      puts("bit 2 = 1");
   }
   else
   {
      puts("bit 2 = 0");
   }

   /* on peut aussi directement récupérer la valeur 0 ou 1
      à l'aide des propriétés de l'opérateur logique ! : */
   printf ("bit 2 = %d\n, !!(a & (1u << 2)));
```

II-E - Conclusion

Je laisse au lecteur le soin de refaire ces exercices, et trouver le moyen de positionner et tester plusieurs bits d'une même variable.

Ces **macros** permettent une manipulation aisée des bits d'un entier jusqu'à 32-bit.