

Bit Masking

Por Ariel Parra.

¿Qué son los bits?

Un bit es un valor booleano que puede tener dos valores, 0 y 1. Estos se usan para representar números con la base 2, por ejemplo un número en notación decimal es el 5 y en notación binaria sería el 101, esto se debe que este número es la representación de $2^2 + 2^0$, donde se cuenta de derecha a izquierda.

Números Binarios

A la combinación de 8 bits se le conoce como Byte.

El termino de hasta la derecha se le conoce como el bit menos significativo (Least Significant Bit, "LSB") y el de hasta la izquierda es el bit más significativo (Most Significant Bit, "MSB").

También existen números negativos donde el MSF dicta el signo 1 para negativo y 0 para positivo, aunque esto solo significa que el MSF sera un número negativo al que se le suman los demas números, por ejemplo en el byte 10000000 el 1 significa que esta negativo y esta en la octava posición por lo que este seria $2^8 = -128$, con un byte con signo se puede ir desde -128 a 127 ya que pasamos por el cero. la forma de tener un valor sin signo en c es usando el tipo de dato unsigned el cual quita el bit de signo.

Asignación de binarios en C++

Se puede trabajar los numeros enteros como binarios ,aunque si se quiere poner un numero binario directamente hay cuatro maneras distintas de asignar un valor binario en C++:

- Con la notación hexadecimal "0x".
- Con la notación octal "0".
- Con una funcion de conversion, `stol(c_string,NULL,BASE)`
- Con el ISO C99 (no estándar) y la notacion "0b".

```
int main() {  
    //229 en decimal  
    unsigned int hexa = 0xE5;  
    unsigned int octal = 0345;  
    string num = "11100101";  
    unsigned int funcion = stol(num.c_str(),null,2);  
    unsigned int binario = 0b11100101;  
    return 0;  
}
```

¿Qué es el Bit Masking?

Es el proceso de modificación y utilización de representaciones binarias de números o cualquier otro dato se conoce como bitmasking. Usando una máscara, múltiples bits en un byte, word, etc. pueden ser estar encendidos o apagado, o también puede ser invertido de encendido a apagado en un solo bit.

Operadores de manipulación

Símbolo	Operador
&	bitwise AND
	bitwise inclusive OR
^	bitwise XOR (exclusive OR)
<<	left shift
>>	right shift
~	bitwise NOT (unario)

Bitwise operations

Estos son operadores de manipulación de bits, muy similares a los operadores booleanos, pero no se deben confundir con los operadores lógicos o relacionales.

Bitwise	Logico	Bitwise	Logico
<code>a & b</code>	<code>a && b</code>	<code>a ^ b</code>	<code>a != b</code>
<code>a b</code>	<code>a b</code>	<code>~a</code>	<code>!a</code>

Ejemplos:

11001000	11001000	11001000	11001000
& 10111000	11111000	^ 11001001	~
-----	-----	-----	-----
= 10001000	= 11111000	= 00000001	= 00110111

Tabla de verdad

bit a	bit b	a & b (a AND b)	a b (a OR b)	a ^ b (a XOR b)	~a (NOT a)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Bitshifting

Estos operadores recorren la cantidad de bits dadas hacia una direccion, izquierda o derecha y dependiendo del tamaño del número binario, si es un número negativo o es mayor o igual al número total de bits este podrá resultar en comportamiento indefinido.

Right shift (>>)

Este sirve como una division truncada hacia abajo (floor), siendo el valor que se shiftea un exponente del numero 2. Ejemplo: Si la variable val contiene al numero en binario 11100101 (229), entonces $\text{val} \gg 1$ ($\text{val} / 2^1$) producirá el resultado 01110010 (114) y $\text{val} \gg 2$ ($\text{val} / 2^2$) producirá 00111001 (57).

```
int main(){
    unsigned int val = 0b11100101;
    val = val >> 1; // binario: 0111001
    cout<<"\nAfter right shift by 1: "<< val;
    val = val >> 2; // binario: 00111001
    cout<<"\nAfter right shift by 2: "<< val;
    return 0;
}
```

Left shift (<<)

Se sirve como multiplicación por 2 siendo el valor que se shifta el exponente, Ejemplo: si la variable val contiene al número en binario 00000111 (7), entonces $\text{val} \ll 1$ ($\text{val} * 2^1$) producirá el resultado 00001110 (14) y $\text{val} \ll 2$ ($\text{val} * 2^2$) dará 00011100 (28).

```
int main(){
    unsigned int val = 0b00000111;
    val = val << 1; // binario: 00001110
    cout<<"\nAfter left shift by 1: "<< val;
    val = val << 2; // binario: 00011100
    cout<<"\nAfter left shift by 2: "<< val;
    return 0;
}
```

Operadores de asignación

```
val=11001000;
```

Symbol	Operator	Ejemplo
&=	bitwise AND assignment	val &= 0b10111000;
=	bitwise inclusive OR assignment	val = 0b11001000;
^=	bitwise exclusive OR assignment	val ^= 0b11001000;
= ~	bitwise NOT (unario)	val = ~val;
<<=	left shift assignment	val <<= 1;
>>=	right shift assignment	val >>= 2;

Problemas

1805A: We Need the Zero

1527A: And Then There Were K

Referencias

<https://yewtu.be/watch?v=qq64FrA2UXQ>

<https://www.scaler.com/topics/data-structures/bit-masking/>

<https://www.learn-c.org/en/Bitmasks>

<https://www.geeksforgeeks.org/what-is-bitmasking/>

[https://en.wikipedia.org/wiki/Mask_\(computing\)](https://en.wikipedia.org/wiki/Mask_(computing))

https://en.wikipedia.org/wiki/Bitwise_operation

https://en.wikipedia.org/wiki/Bitwise_operations_in_C

https://en.wikipedia.org/wiki/Binary_number

<https://www.includehelp.com/c/how-to-assign-binary-value-in-a-variable-directly.aspx>