Bitwise Operators

A **bitwise operator** performs an operation on one or more bit patterns orbinary numerals at the level of their individual bits.

In C++, there are six bitwise operators:

& (bitwise AND)
| (bitwise OR)
^ (bitwise XOR)
~ (bitwise NOT)
<< (left shift)
>> (right shift)

Bitwise AND (&):

The bitwise AND operator is represented by the ampersand (&) symbol. It takes two operands and performs the AND operation on each pair of corresponding bits. The result of an AND operation is 1 if both operands are 1, otherwise the result is 0.

For example:

0000 0101 (5 in decimal)
&
0000 0011 (3 in decimal)
_____
0000 0001 (1 in decimal)

The bitwise AND operator can be used to clear specific bits (set them to 0). For example, consider the following pseudo code:

```
int clear_rightmost_set_bit(int n)
{
  return n & (n-1);
```

```
}
```

The function clear_rightmost_set_bit() takes an integer n as input and clears the rightmost set bit in n. The expression n & (n-1) sets the rightmost set bit in n to 0.

Bitwise OR (|):

The bitwise OR operator is represented by the vertical bar or pipe symbol (|). It takes two operands and performs the OR operation on each pair of corresponding bits. The result of an OR operation is 1 if any of the operands is 1, otherwise the result is 0.

For example:

```
0000 0101  (5 in decimal)
|
0000 0011  (3 in decimal)
_____________
0000 0111  (7 in decimal)
```

The bitwise OR operator can be used to set specific bits (set them to 1). For example, consider the following pseudo code:

```
int set_rightmost_clear_bit(int n)
{
  return n | (n+1);
}
```

The function set_rightmost_clear_bit() takes an integer n as input and sets the rightmost clear bit in n. The expression n | (n+1) sets the rightmost clear bit in n to 1.

Bitwise XOR (^):

The bitwise XOR operator is represented by the caret or circumflex symbol (^). It takes two operands and performs the XOR operation on each pair of corresponding bits. The result of an XOR operation is 1 if the operands are different, otherwise the result is 0.

For example:

0000 0101 (5 in decimal)
^
0000 0011 (3 in decimal)
_____
0000 0110 (6 in decimal)

The bitwise XOR operator can be used to invert specific bits (set them to 1 if they were 0 and vice versa). For example, consider the following pseudo code:

```
int invert_rightmost_set_bit(int n)
{
  return n ^ (n-1);
}
```

The function invert_rightmost_set_bit() takes an integer n as input and inverts the rightmost set bit in n. The expression n ^ (n-1) inverts the rightmost set bit in n.

Bitwise NOT (~):

The bitwise NOT operator is represented by the tilde symbol (~). It takes one operand and performs the NOT operation on each bit. The result of a NOT operation is the complement of the operand. The bitwise NOT operator is a unary operator, which means it only takes one operand.

For example:

0000 0001 (1 in decimal)

1111 1110 (-2 in decimal)

The bitwise NOT operator can be used to toggle specific bits (set them to 0 if they were 1 and vice versa). For example, consider the following pseudo code:

```
int toggle_rightmost_set_bit(int n)
{
  return n ^ (1 << r);
}
```

The function toggle_rightmost_set_bit() takes an integer n as input and toggles the rightmost set bit in n. The expression n ^ (1 << r) toggles the rightmost set bit in n.

Left Shift (<<):

The left shift operator is represented by two less than symbols (<<). It takes two operands; the first operand is the value to be shifted and the second operand is the number of bits to shift. The left shift operator shifts the bits of the first operand to the left by the number of bits specified by the second operand.

For example:

0000 0101 (5 in decimal)
<<
2

0001 0100 (20 in decimal)

The left shift operator can be used to multiply a number by 2. For example, consider the following pseudo code:

```
int multiply_by_two(int n)
{
  return n << 1;
}
```

The function multiply_by_two() takes an integer n as input and multiplies n by 2. The expression n << 1 multiplies n by 2.

Right Shift (>>):

The right shift operator is represented by two greater than symbols (>>). It takes two operands; the first operand is the value to be shifted and the second operand is the number of bits to shift. The right shift operator shifts the bits of the first operand to the right by the number of bits specified by the second operand.

For example:

0000 0101 (5 in decimal)
>>
2

0000 0001 (1 in decimal)

The right shift operator can be used to divide a number by 2. For example, consider the following pseudo code:

```
int divide_by_two(int n)
{
  return n >> 1;
}
```

The function divide_by_two() takes an integer n as input and divides n by 2. The expression n >> 1 divides n by 2.