Bit operations cheatsheet

Bit Operations Cheatsheet

Here's a list of all the Bit Operations available in C++:

Bitwise AND (&)

The bitwise AND operator compares each bit of the first operand to the corresponding bit of the second operand. If both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

Example:

```
int a = 7; // 111 in binary
int b = 10; // 1010 in binary
int c = a & b; // c = 2 (0010 in binary)
```

Bitwise OR (|)

The bitwise OR operator compares each bit of the first operand to the corresponding bit of the second operand. If either bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

Example:

```
int a = 7; // 111 in binary
int b = 10; // 1010 in binary
int c = a | b; // c = 15 (1111 in binary)
```

Bitwise XOR (^)

The bitwise XOR operator compares each bit of the first operand to the corresponding bit of the second operand. If the bits are different (one is 0 and the other is 1), the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

Example:

```
int a = 7; // 111 in binary
int b = 10; // 1010 in binary
int c = a ^ b; // c = 13 (1101 in binary)
```

Bitwise NOT (~)

The bitwise NOT operator takes each bit of the operand and flips it. If the bit is 0, it becomes 1. If the bit is 1, it becomes 0.

Example:

```
int a = 7; // 111 in binary
int b = ~a; // b = -8 (1000 in binary)
```

Left Shift (<<)

The left shift operator takes the bits of the first operand and shifts them to the left by the number of bits specified by the second operand. The bits that are shifted out of the left side of the operand are discarded, and the new bits on the right side are set to 0.

Example:

```
int a = 7; // 111 in binary
int b = a << 2; // b = 28 (11100 in binary)
```

Right Shift (>>)

The right shift operator takes the bits of the first operand and shifts them to the right by the number of bits specified by the second operand. The bits that are shifted out of the right side of the operand are discarded, and the new bits on the left side are set to 0 if the operand is unsigned. If the operand is signed, the new bits on the left side are set to the value of the sign bit (the leftmost bit).

Example:

```
int a = 7; // 111 in binary
int b = a >> 1; // b = 3 (11 in binary)
```

Bitwise AND Assignment (&=)

The bitwise AND assignment operator performs a bitwise AND operation on the two operands and stores the result in the first operand.

Example:

```
int a = 7; // 111 in binary
int b = 10; // 1010 in binary
a &= b; // a = 2 (0010 in binary)
```

Bitwise OR Assignment (|=)

The bitwise OR assignment operator performs a bitwise OR operation on the two operands and stores the result in the first operand.

Example:

```
int a = 7; // 111 in binary
int b = 10; // 1010 in binary
a |= b; // a = 15 (1111 in binary)
```

Bitwise XOR Assignment (^=)

The bitwise XOR assignment operator performs a bitwise XOR operation on the two operands and stores the result in the first operand.

Example:

```
int a = 7; // 111 in binary
int b = 10; // 1010 in binary
a ^= b; // a = 13 (1101 in binary)
```

Left Shift Assignment (<<=)

The left shift assignment operator performs a left shift operation on the first operand by the number of bits specified by the second operand, and stores the result in the first operand.

Example:

```
int a = 7; // 111 in binary
a <<= 2; // a = 28 (11100 in binary)
```

Right Shift Assignment (>>=)

The right shift assignment operator performs a right shift operation on the first operand by the number of bits specified by the second operand, and stores the result in the first operand.

Example:

```
int a = 7; // 111 in binary
a >>= 1; // a = 3 (11 in binary)
```