Basic Bitwise Operators + (Part 2) – Tutorial #9

T.K. HAREENDRAN

AVR tutorial

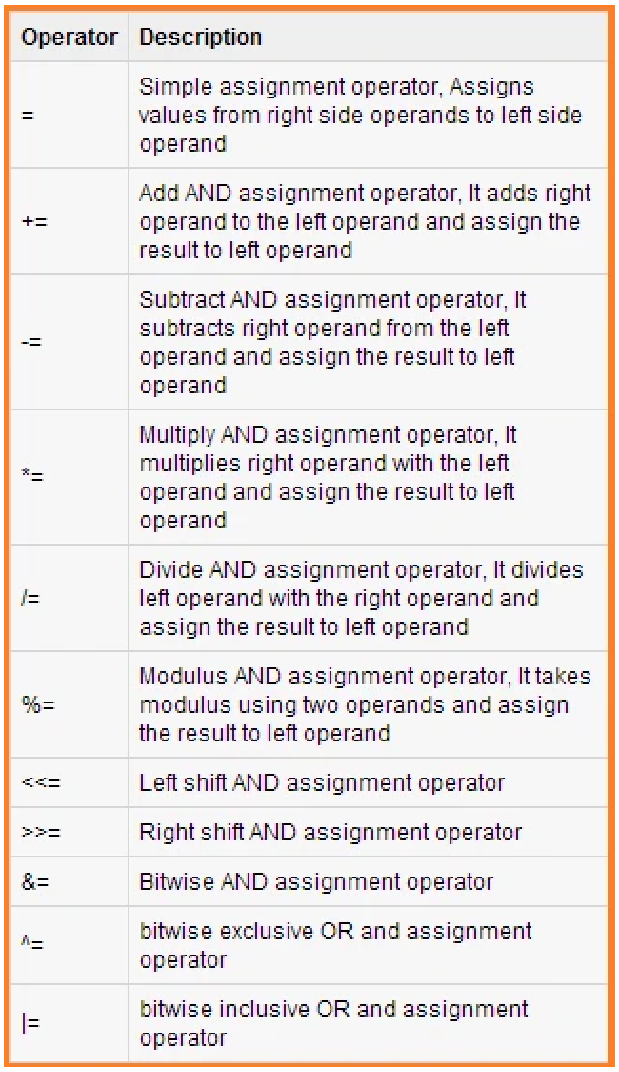
Share this:

Share More

This is the leftover part of Basic Bitwise Operators, prepared to draw your attention to some more learning points in connection with the Bitwise operation. In the previous tutorial, we learned all about decimal system, binary system, bitwise logic, assignment opeartors, bitshift operators, etc. So to recap: An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. As you may well know, C++ have many built-in operators including the following:

- Bitwise Operators
- Logical Operators
- Assignment Operators
- Arithmetic Operators

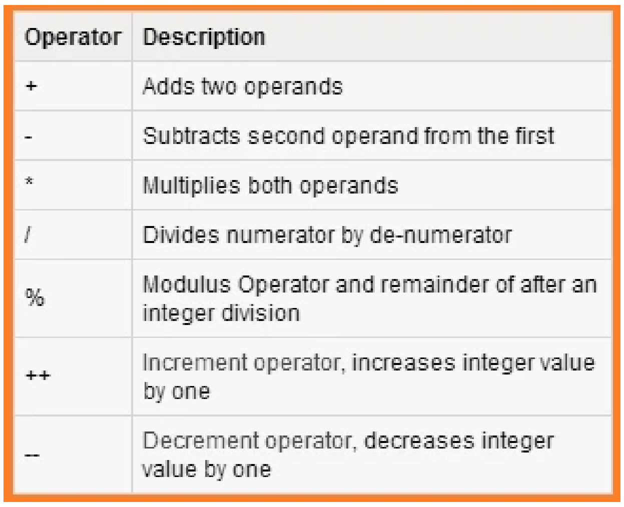
Bitwise operator works on bits and perform "bit-by-bit" operation. Supported Logical operators are AND, OR and NOT. There are following Assignment operators, and Arithmetic operators.



Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand
<<=	Left shift AND assignment operator
>>=	Right shift AND assignment operator
&=	Bitwise AND assignment operator
^=	bitwise exclusive OR and assignment operator
|=	bitwise inclusive OR and assignment operator

The assignment operator assigns a value to a variable. The statement **X = 5**assigns the integer value 5 to the variable X. And the statement X = Y assigns to variable X the value contained in variable Y. The value of X at the moment this statement is executed is lost and replaced by the value of Y. Note that the assignment operation always takes place from right to left.

A word of caution! The assignment operator (operator =, with one equal sign) is not the same as the equality comparison operator (operator ==, with two equal signs). The first one (=) assigns the value on the right-hand to the variable on its left, while the other (==) compares whether the values on both sides of the operator are equal.



Operator	Description
+	Adds two operands
-	Subtracts second operand from the first
*	Multiplies both operands
/	Divides numerator by de-numerator
%	Modulus Operator and remainder of after an integer division
++	Increment operator, increases integer value by one
–	Decrement operator, decreases integer value by one

Literally, operations of addition, subtraction, multiplication and division correspond to their respective mathematical operators. As an example , assume that variable A holds 20 and variable B holds 40, then A + B will give 60 , and A++ will give 21 . The modulus operator, represented by the % sign , gives the remainder of a division of two values. For example, **X = 11 % 3** results in variable X containing the value 2, since dividing 11 by 3 results in 3, with a remainder of 2. There are few other operators supported by C++ Language, and a forthcoming chapter will examine them one by one.

Boolean Operations?

A bit (short form of Binary Digit) is the minimum amount of information, since it only stores either value 1 or 0, which represents either Yes or No (True or False). Many operations can be performed with bits, either in alliance with other bits or themselves alone. These operations receive the name of Boolean operations, comes from the name of the mathematician George Boole. Next you have a list of the basic boolean operations. Combining these operations we can obtain any possible result from two bits.

- **AND:** This operation is performed between two bits. The result of applying the AND operation is 1 if both are equal to 1, and 0 in all other cases.
- **OR:** This operation is also performed between two bits . The result is 1 if either one of the two bits is 1, or if both are 1. If none is equal to 1 the result is 0.
- **NOT:** This operation is performed on a single bit. Its result is the inversion of the actual value of the bit.If it was set to 1 it becomes 0, and if it was 0 it becomes 1.

The operator **!** is the C++ operator for the Boolean operation NOT. The logical operators **&&** and **||** corresponds to the Boolean logical operation AND, the Boolean logical operation **OR** respectively. When using the logical operators, C++ only evaluates what is necessary from left to right to come up with the combined relational result, ignoring the rest. Look at this example **(5==5)||(5&g-
t;6)**. Here, C++ evaluates first whether 5==5 is true, and if so, it never checks whether 5>6 is true or not. This is known as "short-circuit evaluation".

Difference between the **Bit-wise AND** and the **Conditional AND** operators

The **&** operator is the “bit-wise AND” operator. This permits you to manipulate individual bits in a number. The **&&** is the “conditional AND” operator, and the && operator is evaluated in short circuit fashion. The & (unconditional) “logical AND” operator, when applied to boolean value yield a boolean value. This operator is similar to the && operator, except that the & operator is not evaluated in “short-circuit” fashion. That is, both operands are first evaluated before computing the result.

I thrashed over a lot that you may catch in later tutorials. You now know just enough to be serious and I hope this learning process hasn’t caused your brow to do too much harm to your tabletop!

Code Snippet of the “LED+Switch” Project in Part 8!

```
int main(void) {
    init();
    //Set the Data Direction Register for the LED to output
    DDRC |= (1<<5);
    //Set the Data Direction Register for the switch to input
    DDRC &= ~(1<<3);
    //Enable the pullup resistor for the switch
    PORTC|=(1<<3);
    while (1) {
        if (PINC & (1<<3))
            PORTC|=(1<<5);
        else
            PORTC&=~(1<<5);
    }
}
```

→ Part 10: **ATmega8 Advanced Guide - 8-bit timers**