**Rajmani Kumar,**
**Lecturer, Dept. of BCA**
**S.U.College, Hilsa (Nalanda)**
**Patliputra University, Patna**

**BCA-2nd Year** **Paper-III**

# OPERATORS & EXPRESSIONS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Increment and decrement operators
- Conditional operators
- Misc Operators

## Arithmetic operator:

These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus.

Following table shows all the arithmetic operators supported by C language. Assume variable A holds 10 and variable B holds 20 then:

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands | A + B will give 30 |
| - | Subtracts second operand from the first | A – B will give -10 |
| * | Multiplies both operands | A * B will give 200 |
| / | Divides numerator by de-numerator | B / A will give 2 |
| % | Modulus Operator and remainder of after an integer division | B % A will give 0 |
| ++ | Increments operator increases integer value by one | A++ will give 11 |
| -- | Decrements operator decreases integer value by one | A–will give 9 |

## Relational Operators:

These operators are used to compare the value of two variables.

Following table shows all the relational operators supported by C language. Assume variable A holds 10 and variable B holds 20, then:

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

## Logical Operators:

These operators are used to perform logical operations on the given two variables.

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then:

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are nonzero, then condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true. |

## Bitwise Operators

Bitwise operator works on bits and performs bit-by-bit operation. Bitwise operators are used in bit level programming. These operators can operate upon *int* and *char* but not on *float* and *double*.

**Showbits( )** function can be used to display the binary representation of any integer or character value.

Bit wise operators in C language are; & (bitwise AND), | (bitwise OR), ~ (bitwise OR), ^ (XOR), << (left shift) and >> (right shift).

The truth tables for &, |, and ^ are as follows:

| ***p*** | ***q*** | **p & q** | **p \| q** | **p ^ q** |
|---|---|---|---|---|
| *0* | *0* | 0 | 0 | 0 |
| *0* | *1* | 0 | 1 | 1 |
| *1* | *1* | 1 | 1 | 0 |
| *1* | *0* | 0 | 1 | 1 |

The Bitwise operators supported by C language are explained in the following table. Assume variable A holds 60 (00111100) and variable B holds 13 (00001101), then:

| **Operator** | **Description** | **Example** |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12, which is 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) will give 61, which is 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49, which is 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -61, which is 1100 0011 in 2's complement form. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 0000 1111 |

## Assignment Operators:

In C programs, values for the variables are assigned using assignment operators.

There are following assignment operators supported by C language:

| **Operator** | **Description** | **Example** |
|---|---|---|
| = | Simple assignment operator, Assigns values from right side operands to left side operand | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right | C -= A is equivalent to C = C |

| | | |
|---|---|---|
| | operand from the left operand and assign the result to left operand | – A |
| *= | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator | C &= 2 is same as C = C & 2 |
| ^= | bitwise exclusive OR and assignment operator | C ^= 2 is same as C = C ^ 2 |
| \|= | bitwise inclusive OR and assignment operator | C \|= 2 is same as C = C \| 2 |

## INCREMENT AND DECREMENT OPERATOR

In C, ++ and – are called increment and decrement operators respectively. Both of these operators are unary operators, i.e, used on single operand. ++ adds 1 to operand and – subtracts 1 to operand respectively. For example:

*Let a=5 and b=10*

*a++; //*a becomes 6

*a--; //*a becomes 5

*++a; //*a becomes 6

*--a; //*a becomes 5

When i++ is used as prefix(like: ++var), ++var will increment the value of var and then return it but, if ++ is used as postfix(like: var++), operator will return the value of operand first and then only increment it. This can be demonstrated by an example:

```
#include <stdio.h>
int main()
{
int c=2,d=2;
printf("%d\n",c++); //this statement displays 2 then, only c incremented by 1 to 3.
Printf("%d",++c); //this statement increments 1 to c then, only c is displayed.
Return 0;
}
```

Output
*2*
*4*

## Conditional Operators (? :)

Conditional operators are used in decision making in C programming, i.e, executes different statements according to test condition whether it is either true or false.
Syntax of conditional operators;
*conditional_expression?expression1:expression2*
If the test condition is true (that is, if its value is non-zero), expression1 is returned and if false expression2 is returned.
Let us understand this with the help of a few examples:

```
int x, y ;

scanf ( "%d", &x ) ;

y = ( x> 5 ? 3 : 4 ) ;
```

This statement will store 3 in y if x is greater than 5, otherwise it will store 4 in y.
The equivalent if statement will be,

```
if ( x > 5 )

        y = 3 ;

else

        y = 4 ;
```

## Misc Operators:

There are few other operators supported by c language.

| Operator | Description | Example |
|---|---|---|
| sizeof() | It is a unary operator which is used in finding the size of data type, constant, arrays, structure etc. | sizeof(a), where a is integer, will return 4. |
| & | Returns the address of a variable. | &a; will give actual address of the variable. |
| * | Pointer to a variable. | *a; will pointer to a variable. |

## Operators Precedence in C

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

| Category | Operator | Associativity |
|---|---|---|
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)* &sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | <<>> | Left to right |
| Relational | <<= >>= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |

## Expressions

An expression consists of a combination of operators, operands, variables & function calls. An expression can be arithmetic, logical or relational. Here are some expressions:

*a+b – arithmetic operation*

*a>b- relational operation*

*a == b - logical operation*

*func (a,b) – function call*

*4+21*

*a*(b + c/d)/20*

*q = 5*2 x =*

*++q % 3*

*q > 3*

As you can see, the operands can be constants, variables, or combinations of the two. Some expressions are combinations of smaller expressions, called subexpressions. For example, c/d is a subexpression of the sixth example.

An important property of C is that every C expression has a value. To find the value, you perform the operations in the order dictated by operator precedence.