CHAPTER

9

C Operators

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
- Misc Operators

This tutorial will explain the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

Arithmetic Operators

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

Try the following example to understand all the arithmetic operators available in C programming language:

```
#include <stdio.h>

main()
{
   int a = 21;
   int b = 10;
   int c ;

   c = a + b;
   printf("Line 1 - Value of c is %d\n", c );
   c = a - b;
   printf("Line 2 - Value of c is %d\n", c );
   c = a * b;
   printf("Line 3 - Value of c is %d\n", c );
   c = a / b;
   printf("Line 4 - Value of c is %d\n", c );
   c = a % b;
   printf("Line 5 - Value of c is %d\n", c );
   c = a++;
   printf("Line 6 - Value of c is %d\n", c );
   c = a--;
   printf("Line 7 - Value of c is %d\n", c );

}
```

When you compile and execute the above program, it produces the following result:

```
Line 1 - Value of c is 31

Line 2 - Value of c is 11

Line 3 - Value of c is 210

Line 4 - Value of c is 2

Line 5 - Value of c is 1

Line 6 - Value of c is 21

Line 7 - Value of c is 22
```

Relational Operators

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.

Try the following example to understand all the relational operators available in C programming language:

```
#include <stdio.h>

main()
{
   int a = 21;
   int b = 10;
   int c ;

   if( a == b )
   {
      printf("Line 1 - a is equal to b\n" );
   }
   else
   {
      printf("Line 1 - a is not equal to b\n" );
   }
   if ( a < b )
   {
      printf("Line 2 - a is less than b\n" );
   }
   else
   {
      printf("Line 2 - a is not less than b\n" );
   }
   if ( a > b )
   {
      printf("Line 3 - a is greater than b\n" );
   }
   else
   {
      printf("Line 3 - a is not greater than b\n" );
   }
   /* Lets change value of a and b */
   a = 5;
   b = 20;
   if ( a <= b )
```

```
   {
      printf("Line 4 - a is either less than or equal to  b\n" );
   }
   if ( b >= a )
   {
      printf("Line 5 - b is either greater than  or equal to b\n" );
   }
}
```

When you compile and execute the above program, it produces the following result:

```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to  b
Line 5 - b is either greater than  or equal to b
```

Logical Operators

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 non-zero, 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.

Try the following example to understand all the logical operators available in C programming language:

```
#include <stdio.h>

main()
{
   int a = 5;
   int b = 20;
   int c ;

   if ( a && b )
   {
      printf("Line 1 - Condition is true\n" );
   }
   if ( a || b )
   {
```

```
      printf("Line 2 - Condition is true\n" );
   }
   /* lets change the value of  a and b */
   a = 0;
   b = 10;
   if ( a && b )
   {
      printf("Line 3 - Condition is true\n" );
   }
   else
   {
      printf("Line 3 - Condition is not true\n" );
   }
   if ( !(a && b) )
   {
      printf("Line 4 - Condition is true\n" );
   }
}
```

When you compile and execute the above program, it produces the following result:

```
Line 1 - Condition is true

Line 2 - Condition is true

Line 3 - Condition is not true

Line 4 - Condition is true
```

Bitwise Operators

Bitwise operator works on bits and performs bit-by-bit operation. 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

Assume if A = 60; and B = 13; now in binary format they will be as follows:

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13, 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 -60, which is 1100 0011
<<	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

Try the following example to understand all the bitwise operators available in C programming language:

```
#include <stdio.h>

main()
{

   unsigned int a = 60;   /* 60 = 0011 1100 */
   unsigned int b = 13;   /* 13 = 0000 1101 */
   int c = 0;

   c = a & b;       /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );

   c = a | b;       /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;       /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );

   c = ~a;          /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );

   c = a << 2;     /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );

   c = a >> 2;     /* 15 = 0000 1111 */
```

```
   printf("Line 6 - Value of c is %d\n", c );
}
```

When you compile and execute the above program, it produces the following result:

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

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 operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - 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

Try the following example to understand all the assignment operators available in C programming language:

```c
#include <stdio.h>

main()
{
   int a = 21;
   int c ;

   c =  a;
   printf("Line 1 - =  Operator Example, Value of c = %d\n", c );

   c +=  a;
   printf("Line 2 - += Operator Example, Value of c = %d\n", c );

   c -=  a;
   printf("Line 3 - -= Operator Example, Value of c = %d\n", c );

   c *=  a;
   printf("Line 4 - *= Operator Example, Value of c = %d\n", c );

   c /=  a;
   printf("Line 5 - /= Operator Example, Value of c = %d\n", c );

   c  = 200;
   c %=  a;
   printf("Line 6 - %= Operator Example, Value of c = %d\n", c );

   c <<=  2;
   printf("Line 7 - <<= Operator Example, Value of c = %d\n", c );

   c >>=  2;
   printf("Line 8 - >>= Operator Example, Value of c = %d\n", c );

   c &=  2;
   printf("Line 9 - &= Operator Example, Value of c = %d\n", c );

   c ^=  2;
   printf("Line 10 - ^= Operator Example, Value of c = %d\n", c );

   c |=  2;
   printf("Line 11 - |= Operator Example, Value of c = %d\n", c );

}
```

When you compile and execute the above program, it produces the following result:

```
Line 1 - =  Operator Example, Value of c = 21
Line 2 - += Operator Example, Value of c = 42
Line 3 - -= Operator Example, Value of c = 21
Line 4 - *= Operator Example, Value of c = 441
Line 5 - /= Operator Example, Value of c = 21
Line 6 - %= Operator Example, Value of c = 11
Line 7 - <<= Operator Example, Value of c = 44
Line 8 - >>= Operator Example, Value of c = 11
```

```
Line 9 - &= Operator Example, Value of c = 2
Line 10 - ^= Operator Example, Value of c = 0
Line 11 - |= Operator Example, Value of c = 2
```

Misc Operators ↦sizeof & ternary

There are few other important operators including **sizeof** and **? :** supported by C Language.

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

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

Try the following example to understand the operator precedence available in C programming language:

```
#include <stdio.h>

main()
{
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;       // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n",  e );

   e = ((a + b) * c) / d;     // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

   e = (a + b) * (c / d);   // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );

   e = a + (b * c) / d;      //  20 + (150/5)
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );

   return 0;
}
```

When you compile and execute the above program, it produces the following result:

```
Value of (a + b) * c / d is : 90

Value of ((a + b) * c) / d is  : 90

Value of (a + b) * (c / d) is  : 90

Value of a + (b * c) / d is  : 50
```