C - Operators

An **operator** is a symbol that tells the compiler to perform specific mathematical or logical functions. By definition, an **operator** performs a certain operation on operands. An operator needs one or more operands for the operation to be performed.

Depending on how many operands are required to perform the operation, operands are called as unary, binary or ternary operators. They need one, two or three operands respectively.

- **Unary operators** – ++ (increment), -- (decrement), ! (NOT), ~ (compliment), & (address of), * (dereference)
- **Binary operators** – arithmetic, logical and relational operators except !
- **Ternary operators** – The ? operator

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

We will, in this chapter, look into the way each operator works. Here, you will get an overview of all these chapters. Thereafter, we have provided independent chapters on each of these operators that contain plenty of examples to show how these operators work in C Programming.

Arithmetic Operators

We are most familiar with the arithmetic operators. These operators are used to perform arithmetic operations on operands. The most common arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/).

In addition, the modulo (%) is an important arithmetic operator that computes the remainder of a division operation. Arithmetic operators are used in forming an arithmetic expression. These operators are binary in nature in the sense they need two operands,

and they operate on numeric operands, which may be numeric literals, variables or expressions.

For example, take a look at this simple expression −

```
a + b
```

Here "+" is an arithmetic operator. We shall learn more about arithmetic operators in C in a subsequent chapter.

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

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

Relational Operators

We are also acquainted with relational operators while learning secondary mathematics. These operators are used to compare two operands and return a boolean value (true or false). They are used in a boolean expression.

The most common relational operators are less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), equal to (==), and not equal to (!=). Relational operators are also binary operators, needing two numeric operands.

For example, in the Boolean expression −

```
a > b
```

Here, ">" is a relational operator.

We shall learn more about with relational operators and their usage in one of the following chapters.

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the 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 the 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 the 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 the 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 the condition becomes true.	(A <= B) is true.

Logical Operators

These operators are used to combine two or more boolean expressions. We can form a compound Boolean expression by combining Boolean expression with these operators. An example of logical operator is as follows −

```
a >= 50 && b >= 50
```

The most common logical operators are AND (&&), OR(||), and NOT (!). Logical operators are also binary operators.

Show Examples

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

We will discuss more about Logical Operators in C in a subsequent chapter.

Bitwise Operators

Bitwise operators let you manipulate data stored in computer's memory. These operators are used to perform bit-level operations on operands.

The most common bitwise operators are AND (&), OR (|), XOR (^), NOT (~), left shift (<<), and right shift (>>). Here the "~" operator is a unary operator, while most of the other bitwise operators are binary in narure.

Bitwise operator works on bits and perform 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 A = 60 and B = 13 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 following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Show Examples

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 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 = 15 i.e., 0000 1111

Assignment Operators

As the name suggests, an assignment operator "assigns" or sets a value to a named variable in C. These operators are used to assign values to variables. The "=" symbol is defined as assignment operator in C, however it is not to be confused with its usage in mathematics.

The following table lists the assignment operators supported by the C language −

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C

+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the 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

Hence, the expression "a = 5" assigns 5 to the variable "a", but "5 = a" is an invalid expression in C.

The "=" operator, combined with the other arithmetic, relational and bitwise operators form augmented assignment operators. For example, the += operator is used as add and assign operator. The most common assignment operators are =, +=, -=, *=, /=, %=, &=, |=, and ^=.

Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Show Examples

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	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 and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a 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.

Show Examples

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

Other Operators in C

Apart from the above, there are a few other operators in C that are not classified into any of the above categories. For example, the increment and decrement operators (++ and --) are unary in nature and can appear as a prefix or postfix to the operand.

The operators that work with the address of memory location such as the address-of operator (&) and the dereference operator (*). The sizeof operator (sizeof) appears to be a keyword but really an operator.

C also has the type cast operator (()) that forces the type of an operand to be changed. C also uses the dot (.) and the arrow (->) symbols as operators when dealing with derived data types such as struct and union.

The C99 version of C introduced a few additional operators such as auto, decltype.

A single expression in C may have multiple operators of different type. The C compiler evaluates its value based on the operator precedence and associativity of operators. For example, in the following expression –

```
a + b * c
```

The multiplication operand takes precedence over the addition operator.

We shall understand these properties with examples in a subsequent chapter.

Many other programming languages, which are called C-family languages (such as C++, C#, Java, Perl and PHP) have an operator nomenclature that is similar to C.