Assignment Operators in C

In C language, the assignment operator stores a certain value in an already declared variable. A variable in C can be assigned the value in the form of a literal, another variable, or an expression.

The value to be assigned forms the right-hand operand, whereas the variable to be assigned should be the operand to the left of the "**=**" symbol, which is defined as a simple assignment operator in C.

In addition, C has several augmented assignment operators.

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

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

Simple Assignment Operator (=)

The **=** operator is one of the most frequently used operators in C. As per the ANSI C standard, all the variables must be declared in the beginning. Variable declaration after the first processing statement is not allowed.

You can declare a variable to be assigned a value later in the code, or you can initialize it at the time of declaration.

You can use a literal, another variable, or an expression in the assignment statement.

```
int x = 10; // declaration with initialization
int y;      // declaration
y = 20;     // assignment later
int z = x + y; // assign an expression
int d = 3, f = 5; // definition and initializing d and f.
char x = 'x'; // the variable x has the value 'x'.
```

Once a variable of a certain type is declared, it cannot be assigned a value of any other type. In such a case the C compiler reports a type mismatch error.

In C, the expressions that refer to a memory location are called "lvalue" expressions. A lvalue may appear as either the left-hand or right-hand side of an assignment.

On the other hand, the term rvalue refers to a data value that is stored at some address in memory. A rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.

Variables are lvalues and so they may appear on the left-hand side of an assignment. Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side. Take a look at the following valid and invalid statements –

```
int g = 20; // valid statement
10 = 20;    // invalid statement
```

Augmented Assignment Operators

In addition to the **=** operator, C allows you to combine arithmetic and bitwise operators with the **=** symbol to form augmented or compound assignment operator. The augmented operators offer a convenient shortcut for combining arithmetic or bitwise operation with assignment.

Example 1

For example, the expression "a += b" has the same effect of performing "a + b" first and then assigning the result back to the variable "a".

</> Open Compiler

```
#include <stdio.h>

int main(){

   int a = 10;
   int b = 20;

   a += b;
   printf("a: %d", a);

   return 0;
}
```

Output

Run the code and check its output –

```
a: 30
```

Example 2

Similarly, the expression "a <<= b" has the same effect of performing "a << b" first and then assigning the result back to the variable "a".

</> Open Compiler

```c
#include <stdio.h>

int main(){

   int a = 60;
   int b = 2;

   a <<= b;
   printf("a: %d", a);

   return 0;
}
```

Output

Run the code and check its output –

```
a: 240
```

Example 3

Here is a C program that demonstrates the use of assignment operators in C –

</> Open Compiler

```c
#include <stdio.h>

int 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 );

   return 0;
}
```

Output

When you compile and execute the above program, it will produce 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
```