

C - OPERATORS

C - OPERATORS

C - OPERATORS

C - OPERATORS

C - OPERATORS

Course Title :- Structured Programming Language Sessional

Course Code :- CSE-122 [SECTION-B]

Level Term: 1-II-A(G₁) & 1-II-B(G₃,G₄)

Khandaker Jannatul Ritu, Lecturer(CSE), BAIUST

Topics

- 1) **Arithmetic operators [%, ++, --]**
- 2) Assignment operators [+= , -=, *=, /=, %=, &= , |= , ^=]
- 3) Comparison operators/relational Operators [== , != , < , > , <= , >=]
- 4) Logical operators [&&, ||, !]
- 5) Conditional or Ternary Operator (?:) in C
- 6) Bitwise operators
- 7) **Shift operators (left shift [<<] , right shift [>>])**
- 8) Math functions(abs(n), sqrt(n), pow(a, x), log(n), log10(n), sin(n), cos(n), tan(n), round(n), trunc(n), ceil(n), floor(n))

Arithmetic Operators

1. Binary Arithmetic Operators in C

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	$x \% y$
++	Increment	Increases the value of a variable by 1	$++x$
--	Decrement	Decreases the value of a variable by 1	$--x$

Operators

```
int main(){
    int a = 10, b = 4, res;
    // printing a and b
    printf("a is %d and b is %d\n", a, b);
    res = a + b;    // addition
    printf("a + b is %d\n", res);
    res = a - b;    // subtraction
    printf("a - b is %d\n", res);
    res = a * b;    // multiplication
    printf("a * b is %d\n", res);
    res = a / b;    // division
    printf("a / b is %d\n", res);
    res = a % b;    // modulus
    printf("a %% b is %d\n", res);
}
```

a is 10 and b is 4
a + b is 14
a - b is 6
a * b is 40
a / b is 2
a % b is 2

2. Unary Arithmetic Operators in C

The unary arithmetic operators operate or work with a single operand. In C, we have two unary arithmetic operators which are as follows:

Operator	Symbol	Operation	Example
Decrement Operator	--	Decreases the integer value of the variable by one.	i--, --i
Increment Operator	++	Increases the integer value of the variable by one.	i++, ++i

Increment Operator in C

The '++' operator is used to increment the value of an integer. It can be used in two ways:

1. Pre-Increment

When placed before the variable name (also called the pre-increment operator), its value is incremented instantly. Consider the example:

```
a = ++x;
```

This example can be expanded to

```
a = (x = x + 1);
```

2. Post Increment

When it is placed after the variable name (also called post-increment operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement. For example:

```
a = x++;
```

It can be expanded to

```
a = x;
```

```
x = x + 1;
```

Increment operator(x++, ++x)



```
graph TD; A[Increment operator(x++, ++x)] --> B[Pre-Increment(++x)]; A --> C[Post-Increment(x++)];
```

Pre-Increment(++x)

Example-1

```
int x = 5;  
++x;  
printf("%d", x);
```

Output: 6

Example-2

```
int x = 5;  
printf("%d", ++x);
```

Output: 6

Post-Increment(x++)

Example-3

```
int x = 5;  
x++;  
printf("%d", x);
```

Output: 6

Example-4

```
int x = 5;  
printf("%d", x++);
```

Output: 5

Decrement Operator in C

The '--' operator is used to decrement the value of an integer. Just like the increment operator, the decrement operator can also be used in two ways:

1. Pre-Decrement

When placed before the variable name (also called the **pre-decrement** operator), its value is decremented instantly. For example, --x.

2. Post Decrement

When it is placed after the variable name (also called **post-decrement** operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement. For example, x--.

```
int main()
{
    int a = 10, b = 4, res;

    printf("Post Increment and Decrement\n");
    res = a++;
    printf("a is %d and result is %d\n", a, res);
    res = a--;
    printf("a is %d and result is %d\n", a, res);
    printf("\nPre Increment and Decrement\n");
    res = ++a;
    printf("a is %d and result is %d\n", a, res);
    res = --a;
    printf("a is %d and result is %d\n", a, res);
}
```

Output

Post Increment and Decrement
a is 11 and result is 10
a is 10 and result is 11

Pre Increment and Decrement
a is 11 and result is 11
a is 10 and result is 10

Decrement operator(x--, --x)



```
graph TD; A[Decrement operator(x--, --x)] --> B[Pre-Decrement(--x)]; A --> C[Post-Decrement(x--)];
```

Pre-Decrement(--x)

Example-1

```
int x = 5;  
--x;  
printf("%d", x);
```

Output: 4

Example-2

```
int x = 5;  
printf("%d", --x);
```

Output: 4

Post-Decrement(x--)

Example-3

```
int x = 5;  
x--;  
printf("%d", x);
```

Output: 4

Example-4

```
int x = 5;  
printf("%d", x--);
```

Output: 5

Example of Increment Operator

// C Program to illustrate the increment of both type

```
int main()
{
    int a = 5;
    int b = 5;

    // PREFIX
    int prefix = ++a;
    printf("Prefix Increment: %d\n", prefix);

    // POSTFIX
    int postfix = b++;
    printf("Postfix Increment: %d", postfix);
}
```

Prefix Increment: 6

Postfix Increment: 5

Example of Decrement Operator

```
int main(){
    int a = 5;
    int b = 5;

    // PREFIX
    int prefix = --a;
    printf("Prefix = %d\n", prefix);

    // POSTFIX
    int postfix = b--;
    printf("Postfix = %d", postfix);
}
```

Output

Prefix = 4

Postfix = 5

Multiple Operators in a Single Expression

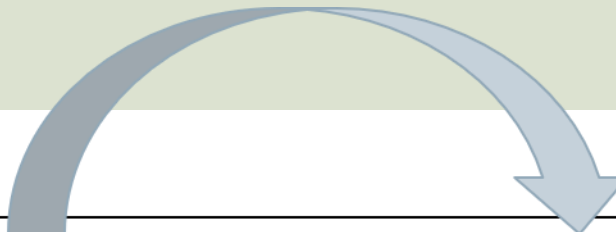
Till now, we have only seen expressions in which we have used a single operator in a single expression. What happens when we use multiple operators in a single expression? Let's try to understand this with the help of the below example.

```
// C program to demonstrate the use
#include <stdio.h>
int main(){
    int var;
    // expression with multiple operators
    var = 10 * 20 + 15 / 5;
    printf("Result = %d", var);
}
```

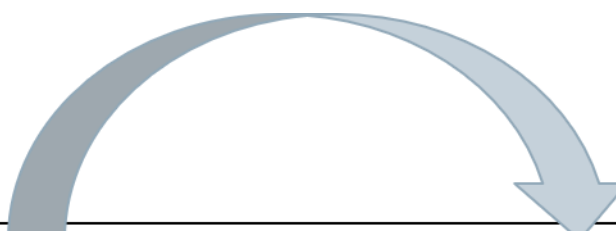
Increment Operator	Decrement Operator
Increment Operator adds 1 to the operand.	Decrement Operator subtracts 1 from the operand.
The Postfix increment operator means the expression is evaluated first using the original value of the variable and then the variable is incremented(increased).The	The Postfix decrement operator means the expression is evaluated first using the original value of the variable and then the variable is decremented(decreased).
Prefix increment operator means the variable is incremented first and then the expression is evaluated using the new value of the variable.	Prefix decrement operator means the variable is decremented first and then the expression is evaluated using the new value of the variable.
Generally, we use this in decision-making and looping.	This is also used in decision-making and looping.

Assignment Operators

Operator	Example	Same As
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$
%=	$x \% = 3$	$x = x \% 3$
&=	$x \& = 3$	$x = x \& 3$
=	$x = 3$	$x = x 3$
^=	$x \wedge = 3$	$x = x \wedge 3$
>>=	$x >> = 3$	$x = x >> 3$
<<=	$x << = 3$	$x = x << 3$



If $x = 13$; $x = x + 3$
 $= 13 + 3 = 16$



If $x = 7$; $x = x * 2$
 $= 7 * 2 = 14$

Code [Assignment Operators]

```
int main(){
    int a = 10;
    printf("Value of a is %d\n", a);
    a += 10;
    printf("Value of a is %d\n", a);
    a -= 10;
    printf("Value of a is %d\n", a);
    a *= 10;
    printf("Value of a is %d\n", a);
    a /= 10;
    printf("Value of a is %d\n", a);
}
```

```
int x = 5;
x %= 3;
printf("%d", x);
x = 5;
x &= 3;
printf("%d", x);
x = 5;
x |= 3;
printf("%d", x);
x = 5;
x ^= 3;
printf("%d", x);
```

```
int main()
{
    int x = 11;
    x = x + 2; // 11+2=13
    x = x + 5; // 13+5=18
    x = x - 3; // 18-3=15
    x = x * 1; // 15*1=15
    x = x * 20; // 15*20=300
    printf("%d", x); // 300
}
```

Comparison/relational Operators

Operator	Name	Example	Description
==	Equal to	$x == y$	Returns 1 if the values are equal
!=	Not equal	$x != y$	Returns 1 if the values are not equal
>	Greater than	$x > y$	Returns 1 if the first value is greater than the second value
<	Less than	$x < y$	Returns 1 if the first value is less than the second value
>=	Greater than or equal to	$x >= y$	Returns 1 if the first value is greater than, or equal to, the second value
<=	Less than or equal to	$x <= y$	Returns 1 if the first value is less than, or equal to, the second value

Comparison/relational Operators

```
int x = 5, y = 5;  
printf("%d", x == y); // returns 1 (true) because 5 is equal to 5
```

```
int x = 5, y = 3;  
printf("%d", x != y); // returns 1 (true) because 5 is not equal to 3
```

```
int x = 5, y = 3;  
printf("%d", x > y); // returns 1 (true) because 5 is greater than 3
```

```
int x = 5, y = 3;  
printf("%d", x < y); // returns 0 (false) because 5 is not less than 3
```

```
int x = 5, y = 3;  
printf("%d", x >= y);
```

```
int x = 5, y = 3;  
printf("%d", x <= y);
```

Logical Operators

Operator	Name	Example	Description
&&	Logical and	$x < 5 \ \&\& \ x < 10$	Returns 1 if both statements are true
	Logical or	$x < 5 \ \ x < 4$	Returns 1 if one of the statements is true
!	Logical not	$!(x < 5 \ \&\& \ x < 10)$	Reverse the result, returns 0 if the result is 1
!	Logical not	!x	If x = true Then !x will be false.

Conditional or Ternary Operator (?:) in C

```
variable = (condition) ? Expression2 : Expression3;
```

```
char operator = '+';  
int num1 = 8;  
int num2 = 7;  
int result = (operator == '+') ? (num1 + num2) : (num1 - num2);  
printf("%d", result);  
// Output: 15
```

```
int age;  
printf("Enter your age: ");  
scanf("%d", &age);  
// ternary operator to find if a person can vote or not  
(age >= 18) ? printf("You can vote") : printf("You cannot vote");
```

Bitwise Operators in C

X	Y	$X \& Y$	$X Y$	$X \wedge Y$	$\sim X$	$\sim Y$
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Bitwise Operators in C(left shift, right shift)

256	128	64	32	16	8	4	2	1
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Left Shift (<<)

value = $a \ll n$

$$= a * 2^n$$

Step-1: find binary of 13

32	16	8	4	2	1
0	0	1	1	0	1

$$= (001101)_2$$

Q-1: Find $13 \ll 1$

Solution: Given, $a = 13$

$n = 1$

Shift 13 by 1 move left.

Step-2: shift every bit of 011010 by 1 move left

0	0	1	1	0	1
0	1	1	0	1	0

$$= (011010)_2$$

Step-3: integer of 011010

32	16	8	4	2	1
0	1	1	0	1	0

$$= (16+8+1) = 26$$

So, $13 \ll 1 = 26$

Another way, $13 * 2^1 = 26$

Bitwise Operators in C(left shift, right shift)

256	128	64	32	16	8	4	2	1
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Right Shift (>>)

value = a >> n

= a / 2^n

Q-1: Find $13 >> 1$

Solution: Given, a = 13

n = 1


Shift 13 by 1 move right.

Step-1: find binary of 13

32	16	8	4	2	1
0	0	1	1	0	1

= $(001101)_2$

Step-2: shift every bit of 011010 by 1 move right



0	0	1	1	0	1
0	0	0	1	1	0

= $(000110)_2$

Step-3: integer of 000110

32	16	8	4	2	1
0	0	0	1	1	0

= $(4+2) = 6$

So, $13 >> 1 = 6$

Another way, $13 / 2^1 = 6$

```
// a = 5(00000101), b = 9(00001001)
```

```
unsigned char a = 5, b = 9;
```

```
// The result is 00000001
```

```
printf("a = %d, b = %d\n", a, b);
```

```
printf("a&b = %d\n", a & b);
```

```
// The result is 00001101
```

```
printf("a|b = %d\n", a | b);
```

```
// The result is 00001100
```

```
printf("a^b = %d\n", a ^ b);
```

```
// The result is 11111010
```

```
printf("~a = %d\n", a = ~a);
```

```
// The result is 00010010
```

```
printf("b<<1 = %d\n", b << 1);
```

```
// The result is 00000100
```

```
printf("b>>1 = %d\n", b >> 1);
```

Output

a = 5, b = 9

a&b = 1

a|b = 13

a^b = 12

~a = 250

b<<1 = 18

b>>1 = 4

Maths function

- **Absolute** → `abs(n)` → `abs(-90) = 90` `abs(90) = 90`
- **Square root** → `sqrt(n)` → `sqrt(100) = 10`
- **Power** → `pow(a, x)` → `pow(2, 5) = 32` `pow(5, 2) = 25`
- **Log** → `log(n)`, `log10(n)` → `log(100) = 2`
- **Trigonometry** → `sin(n)`, `cos(n)`, `tan(n)` → `sin(100) = ?` `cos(100) = ?` `tan(100) = ?`
- **Round** → `round(n)` → `round(7.4) = 7` `round(7.5) = 8` `round(7.9) = 8`
- **Truncate** → `trunc(n)` → `trunc(7.411) = 7` `trunc(7.678) = 7`
- **Ceiling** → `ceil(n)` → `ceil(7.411) = 8` `ceil(2.7) = 3`
- **Floor** → `floor(n)` → `floor(7.411) = 7` `floor(2.7) = 2`

```
#include<stdio.h>
```

```
int main(){
```

```
    ///absolute value : abs(n)
```

```
    int value = -5;
```

```
    printf("absolute value: %d\n", abs(value));
```

```
    ///square root : sqrt(n)
```

```
    double number = 101;
```

```
    printf("Square root: %.2lf\n", sqrt(number));
```

```
    ///power function: pow(a,b)
```

```
    double a=5, b=2;
```

```
    printf("power(a,b) : %.2lf\n\n", pow(a,b));
```

```
    ///log(n)
```

```
    printf("log(n) : %lf\n", log(number));
```

```
    printf("log10(n) : %lf\n\n", log10(number));
```

```
}
```

```
#include<stdio.h>
```

```
int main(){
```

```
    ///sin(), cos(), tan()
```

```
    double n=90;
```

```
    printf("sin(n) : %lf\n", sin(n));
```

```
    printf("cos(n) : %lf\n", cos(n));
```

```
    printf("tan(n) : %lf\n\n", tan(n));
```

```
    ///round(), trunc(), ceil(), floor()
```

```
    printf("round(n) : %.2lf\n", round(11.2));
```

```
    printf("round(n) : %.2lf\n", round(11.5));
```

```
    printf("trunc(n) : %.2lf\n\n", trunc(7.5));
```

```
    /// ceil(), floor()
```

```
    printf("ceil(n) : %.2lf\n", ceil(2.7));
```

```
    printf("floor(n) : %.2lf\n\n", floor(2.7));
```

```
    printf("ceil(n) : %.2lf\n", ceil(4.3));
```

```
    printf("floor(n) : %.2lf\n", floor(4.3));
```

```
}
```