C Programming Lecture 5

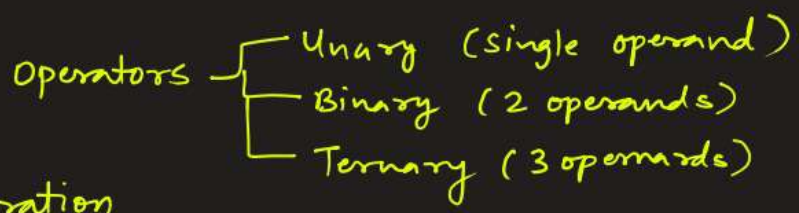
Thursday, 13 June 2024 8:19 PM

Operators in C

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators



Arithmetic Operators

Arithmetic operators carry out ***fundamental mathematical*** operations.

S. No.	Symbol	Operator	Description	Syntax
(B) 1	+	**Plus**	Adds two numeric values.	**a + b**
(B) 2	–	**Minus**	Subtracts right operand from left operand.	**a – b**
(B) 3	*	**Multiply**	Multiply two numeric values.	**a * b**
(B) 4	/	**Divide**	Divide two numeric values.	**a / b**
(B) 5	%	**Modulus**	Returns the remainder after diving the left operand with the right operand.	**a % b**
(U) 6	+	**Unary Plus**	Used to specify the positive values.	**+a**
(U) 7	–	**Unary Minus**	Flips the sign of the value.	**-a**
(U) 8	++	**Increment**	Increases the value of the operand by 1.	a++
(U) 9	—	**Decrement**	Decreases the value of the operand by 1.	a–-

a = -3; a = +3;
↑ unary ↑ unary

pre and post increment / decrement

pre ++a --a
post a++ a--

post inc/dec :- The value is used first and then the op^n is performed

pre inc/dec:- First the op^n is performed then the value is used.

int a = 3; ✓

a++; ++a;

printf("%d", a);

a++ ⟺ a = a+1;

(1)

int a = 3; ✓ 3

int b = (a++);

printf("%d, %d", a, b);

(2)

int a = 3;

int b = (++a);

b = (++a)

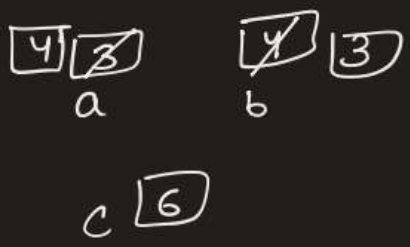
b = (4)

3 4 4
a b

eg. int a=3, b=4;

int c = (a++) + (--b);

printf ("a=%d, b=%d, c=%d", a, b, c);



c = (a++) + (--b);

c = (3) + (--b);

c = (3) + (3);

c = 6;

o/p:- a=4, b=3, c=6

```c
// C program to illustrate the arithmatic operators
#include<stdio.h>
int main() {
    int a = 25, b = 6;

    // using operators and printing results
    printf("a + b = %d\n", a + b);
    printf("a - b = %d\n", a - b);
    printf("a * b = %d\n", a * b);
    printf("a / b = %d\n", a / b);
    printf("a %% b = %d\n", a % b);
    printf("+a = %d\n", +a);
    printf("-a = %d\n", -a);
    printf("a++ = %d\n", a++);
    printf("a-- = %d\n", a--);

    return 0;
}
```

25 → 25

26 → 26

Post

a+b = 31

a-b = 19

a*b = 150

a/b = 4

a%b = 1

+a = 25

-a = -25

a++ = 25

a-- = 26

25/6

↓ int ↓ int

—x—

Relational Operators

Relational operators assess the relationship between values by comparing them. They return either ***true (1)*** or ***false (0)***.

S. No.	Symbol	Operator	Description	Syntax
1	<	Less than	Returns true if the left operand is less than the right operand. Else false	a < b
2	>	Greater than	Returns true if the left operand is greater than the right operand. Else false	a > b
3	<=	Less than or equal to	Returns true if the left operand is less than or equal to the right operand. Else false	a <= b
4	>=	Greater than or equal to	Returns true if the left operand is greater than or equal to right operand. Else false	a >= b
5	==	Equal to	Returns true if both the operands are equal.	a == b
6	!=	Not equal to	Returns true if both the operands are NOT equal.	a != b

```c
// C program to illustrate the relational operators
#include<stdio.h>
int main() {
    int a = 25, b = 6;

    // using operators and printing results
    printf("a < b  : %d\n", a < b);
    printf("a > b  : %d\n", a > b);
    printf("a <= b: %d\n", a <= b);
    printf("a >= b: %d\n", a >= b);
    printf("a == b: %d\n", a == b);
    printf("a != b : %d\n", a != b);
    return 0;
}
```

a<b : false

0
1
0
1
0
1

Logical Operators

Logical operators perform logical operations on ***boolean values*** and return either ***true (1)*** or ***false (0)***.

S. No.	Symbol	Operator	Description	Syntax
1	&&	Logical AND	Returns true if both the operands are true.	a && b
2	||	Logical OR	Returns true if both or any of the operand is true.	a || b
3	!	Logical NOT	Returns true if the operand is false.	!a

Operands → boolean
true false
0
anything other than 0

```c
// C program to illustrate the logical operators
#include<stdio.h>
int main() {
    int a = -1, b = 0;

    // using operators and printing results
    printf("a && b : %d\n", a && b);
    printf("a || b : %d\n", a || b);
    printf("!a: %d\n", !a);
    return 0;
}
```

a	b	a&&b
T	T	T
T	F	F
F	T	F
F	F	F

a	b	a||b
T	T	T
T	F	T
F	T	T
F	F	F

a	!a
T	F
F	T

-1 (a) 0 (b)

a: True
b: False

a&&b = 0
a||b = 1
!a = 0

Example:

```c
#include<stdio.h>
int main(){
    int a = 50, b = 51, c = 5;
    printf("%d", (a==(--b)) && (b/c == 10));
    return 0;
}
```

50 → a, 51 → b, 5 → c

b = 50

10

(a==50) && (b/c==10)

1 && 1

true && true

true → 1

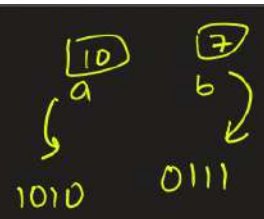
Bitwise Operators

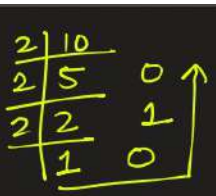
Bitwise operators perform operations on individual bits of the operands. The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.

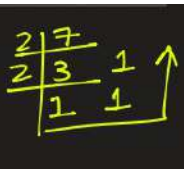
S. No.	Symbol	Operator	Description	Syntax
(B) 1	&	**Bitwise AND**	Performs bit-by-bit AND operation and returns the result.	**a & b**
(B) 2	|	**Bitwise OR**	Performs bit-by-bit OR operation and returns the result.	**a | b**
(B) 3	^	**Bitwise XOR**	Performs bit-by-bit XOR operation and returns the result.	**a ^ b**
(U) 4	~	**Bitwise First Complement**	Flips all the set and unset bits on the number.	**~a**
(B) 5	<<	**Bitwise Left shift**	Shifts the number in binary form by one place in the operation and returns the result.	**a << b**
(B) 6	>>	**Bitwise Right shift**	Shifts the number in binary form by one place in the operation and returns the result.	**a >> b**

int a = 10, b = 7;

a & b = 2 a | b = 15







a 1010
b 0111
& 0010
$2^3 2^2 2^1 2^0$
(2)

a 1010
b 0111
a|b 1111
(15)

a	b	a&b
1	1	1
1	0	0
0	1	0
0	0	0

a	b	a|b
1	1	1
1	0	1
0	1	1
0	0	0

a	~a
0	1
1	0

a	b	a^b
1	1	0
1	0	1
0	1	1
0	0	0

a ^ b = 13

a 1010
b 0111
a^b 1101
(13)

8+4+1

13 >> 1 = 6
Dividing by 2

13 << 1
(26)
Multiplying by 2

Right shift,
leftmost bit gets copied in the leftmost place

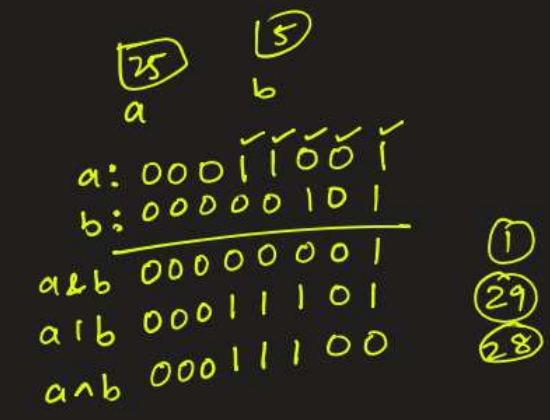
left shift
rightmost bit always 0

a = a * 2; costly.
a = a << 1; ✓ cheaper

Example:

```c
// C program to illustrate the bitwise operators
#include<stdio.h>
int main() {
    int a = 25, b = 5;

    // using operators and printing results
    printf("a & b: %d\n", a & b);
    printf("a | b: %d\n", a | b);
    printf("a ^ b: %d\n", a ^ b);
    printf("~a: %d\n", ~a);
    printf("a >> b: %d\n", a >> b);
    printf("a << b: %d\n", a << b);
    return 0;
}
```

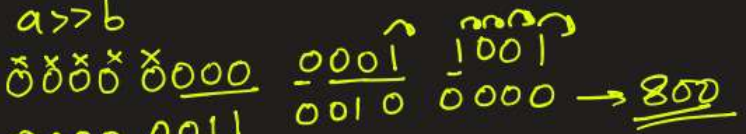
1
29
28
-26
0
800



25>>5 = 0

25<<5 = 800

25×2×2×2×2×2
= 25×32 = 800



Assignment Operator

a = 3

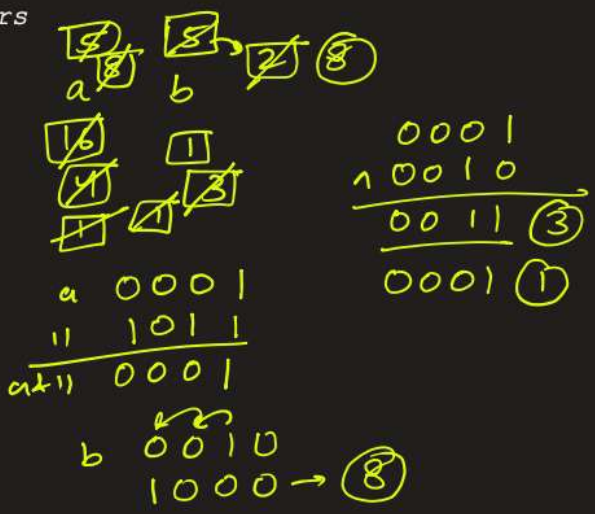
Assignment operators are used to assign values to variables. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value.

The assignment operators can be combined with some other operators in C to provide multiple operations using single operator. These operators are called compound operators.

S. No.	Symbol	Operator	Description	Syntax	
1	=	**Simple Assignment**	Assign the value of the right operand to the left operand.	a = b	
2	+=	**Plus and assign**	Add the right operand and left operand and assign this value to the left operand.	a += b	a = a+b
3	-=	**Minus and assign**	Subtract the right operand and left operand and assign this value to the left operand.	a -= b	a = a-b
4	*=	**Multiply and assign**	Multiply the right operand and left operand and assign this value to the left operand.	a *= b	a = a*b
5	/=	**Divide and assign**	Divide the left operand with the right operand and assign this value to the left operand.	a /= b	a = a/b
6	%=	**Modulus and assign**	Assign the remainder in the division of left operand with the right operand to the left operand.	a %= b	a = a%b
7	&=	**AND and assign**	Performs bitwise AND and assigns this value to the left operand.	a &= b	a = a&b
8	|=	**OR and assign**	Performs bitwise OR and assigns this value to the left operand.	a |= b	a = a|b
9	^=	**XOR and assign**	Performs bitwise XOR and assigns this value to the left operand.	a ^= b	a = a^b
10	>>=	**Rightshift and assign**	Performs bitwise Rightshift and assign this value to the left operand.	a >>= b	a = a>>b
11	<<=	**Leftshift and assign**	Performs bitwise Leftshift and assign this value to the left operand.	a <<= b	a = a<<b

```c
// C program to illustrate the assignment operators
#include<stdio.h>
int main(){
    // using operators and printing results
    int a, b;
    a = 5;
    b = a;
    a += 3;
    b -= 3;
    a *= b;
    a /= 4;
    a %= 3;
    a &= 11;
    a ^= b;
    a >>= 1;
    b <<= 2;
    printf("a = %d, b = %d", a, b);
    return 0;
}
```

a=1, b=8



Miscellaneous Operators

Apart from the above operators, there are some other operators available in C used to perform some specific tasks.

sizeof Operator

- sizeof is much used in the C programming language.
- It is a compile-time unary operator which can be used to compute the size of its operand.
- The result of sizeof is of the unsigned integral type which is usually denoted by size_t.
- Basically, the sizeof the operator is used to compute the size of the variable or datatype.

sizeof(char);
↳ unsigned long

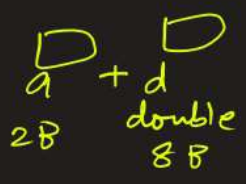
sizeof (operand)

```c
// C Program To demonstrate sizeof operator
#include<stdio.h>
int main(){
    printf("sizeof char = %lu\n", sizeof(char));
    printf("sizeof int = %lu\n", sizeof(int));
    printf("sizeof float = %lu\n", sizeof(float));
    printf("sizeof double = %lu\n", sizeof(double));
    int a = 0;
    double d = 10.21;
    printf("sizeof int + double = %lu", sizeof(a + d));
    return 0;
}
```

1 → 2/4 (32B 64B)

sizeof float → 4

sizeof double → 8



sizeof int + double → 8

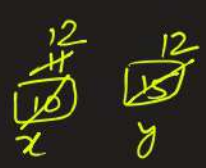
Comma Operator (,)

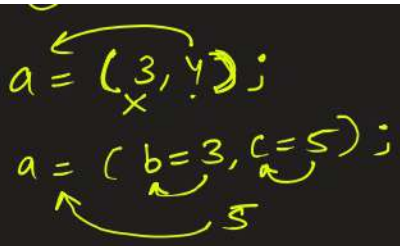
- The comma operator (represented by the token) is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns this value (and type).
- The comma operator has the lowest precedence of any C operator.
- Comma acts as both operator and separator.

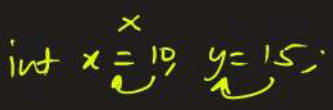
`operand1 , operand2`

a=3, b=4, c=5;

```c
// C Program to Demonstrate comma operator
#include<stdio.h>
int main() {
    // comma as a separator
    int x = 10, y = 15;

    // using comma as an operator
    printf("%d \n", (x, y));

    y = (x++, ++x);
    // using comma as an operator
    printf("%d \n", (x, y));

    // using comma as terminator
    printf("Hello \n"),
    printf("World!");

    return 0;
}
```







Conditional Operator (?:)

- The conditional operator is the only ternary operator in C++.
- Here, Expression1 is the condition to be evaluated. If the condition(Expression1) is True then we will execute and return the result of Expression2 otherwise if the condition(Expression1) is false then we will execute and return the result of Expression3.
- We may replace the use of if..else statements with conditional operators.

```c
// C Program to Demonstrate ternary operator
#include<stdio.h>
int main(){
    int a = 5;
    int b = 3;
    printf("%d", (a > b) ? a : b);
    return 0;
}
```