OPERATORS

OPERATORS

An operator specifies the operation to be applied to its operands.

The values/variables upon which the operation is performed are called operands.

Combination of operator and operands is called an expression.

a+b

a and b are operands and ‘+’ is a operator

OPERATORS

There are 3 types of operators:-

a. Unary operator

b. Binary Operator

c. Ternary Operator

Further there are more categories of operators.

CLASSIFICATION OF OPERATORS

Arithmetic operators (Binary)

Relational Operators (Binary)

Logical Operators (Binary)

Increments and Decrement Operators (unary)

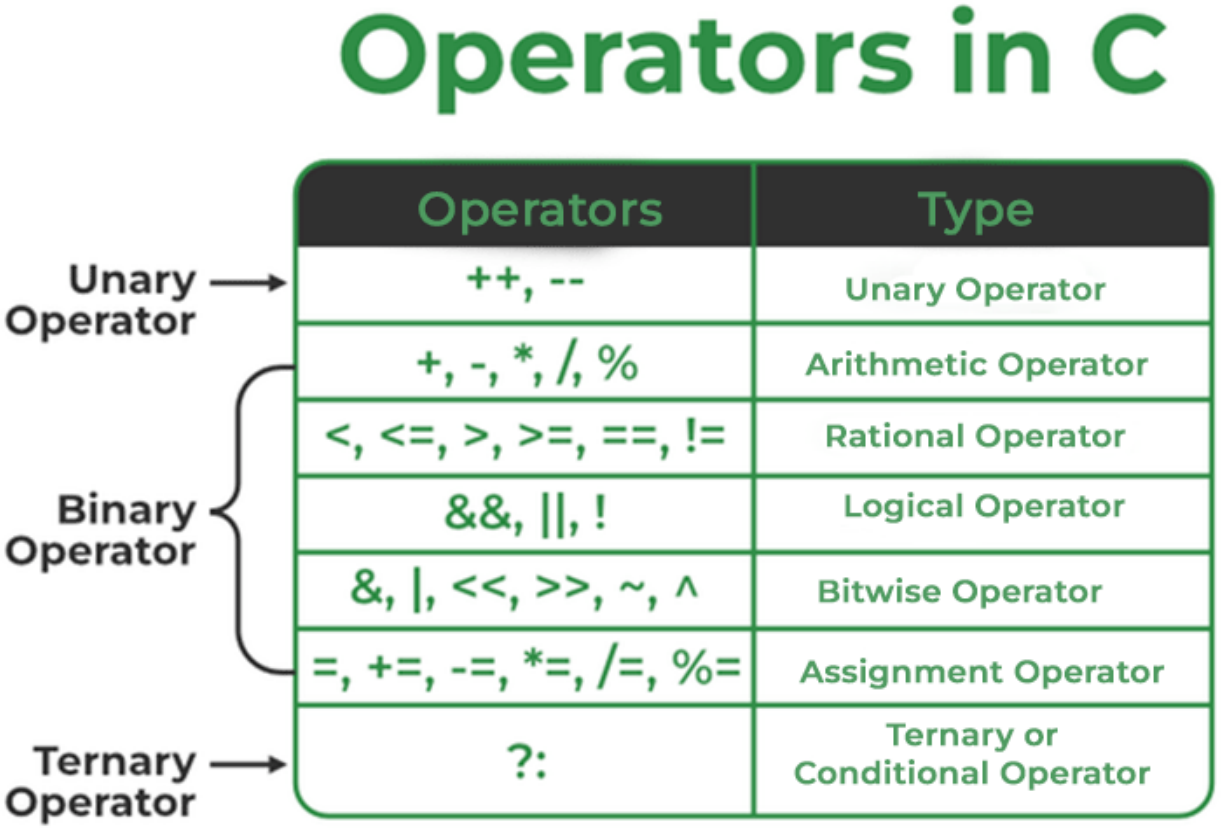
Conditional Operators (ternary)

Bitwise Operators (Binary)

Special Operators.

Assignment Operators

CLASSIFICATION OF OPERATORS



ARITHMETIC OPERATORS

+,-, /, * AND %

% Operator is used to calculate remainder.

- **In case of modulo division,** if the numerator is smaller than the denominator, then answer will be the number itself.

eg, 5%10=5

- In case of modulo division, the sign of the output will always be equal **to sign of numerator.**

A=-5%10, A=-5

ARITHMETIC OPERATORS

- In case of division, if the number is smaller than the denominator, then answer will always be truncated towards zero.

A=5/10

A=0

ARITHMETIC OPERATORS

//program on arithmetic operations

```
#include <stdio.h>
int main()
{     int a = 9,b = 4, c;
      c = a+b;
     printf("a+b = %d \n",c);
     c = a-b;
   printf("a-b = %d \n",c);
     c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);
          return 0; }
```

RELATIONAL OPERATORS

Relational operators are used to compare 2 quantities together.

Relational operator always give output in the form of true or false (i.e. 1 or 0)

Relational Operators are:

> , < , >= , <=, ==, !=

RELATIONAL OPERATORS

Relational Operators in C

Operator Symbol	Operator Name
==	Equal To
!=	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than Equal To
>=	Greater Than Equal To

RELATIONAL OPERATORS

Assume variable A holds 10 and variable B holds 20 then:

- **==** Checks if the value of two operands is equal or not, if yes then condition becomes true.

(A == B) is not true.

- != Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.

(A != B) is true.

RELATIONAL OPERATORS

- 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.

RELATIONAL OPERATORS

- <= 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

RELATIONAL OPERATORS

//example

```
#include<stdio.h>
int main ()
{
  int a=10,b=20;
printf("a>b: %d",a>b);
printf("a<b : %d",a<b);
printf("a>=b : %d",a>=b);
printf("a<b : %d",a<=b);
printf("a==b: %d",a==b);
printf("a!=b: %d",a!=b);
 return 0;
}
```

LOGICAL OPERATORS

Logical Operators: &&, ||, !

Logical AND and OR is binary operator , ! Is unary operator

Assume variable A holds 10 and variable B holds 20 then:

- **&&** Called Logical AND operator. If both the operands are non zero then then condition becomes true.

(A && B) is true.

- || Called Logical OR Operator. If any of the two operands is non zero then then condition becomes true. (A || B) is true.

LOGICAL OPERATORS

- ! 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 false.

LOGICAL OPERATORS

```
//logical operator
#include<stdio.h>
void main()
{
int a=10,b=20;
printf("%d",a&&b);
printf("%d",a||b);
printf("%d",!(a&&b));  }
```

ASSIGNMENT OPERATORS

- **=** Simple assignment operator, **Assigns values from Assigns values from right side operands to left side operand.**
- int a=15

(15 is assigned to a)

- **Note: there can be single variable at LHS of assignment operator.**

a=a+b; **c=d+e;**

- **A+b=c+d //not allowed**

ASSIGNMENT OPERATORS

- Shorthand assignment operator
- 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

ASSIGNMENT OPERATORS

- *= 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

ASSIGNMENT OPERATORS

```
//  Assignment operators
#include <stdio.h>
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);
    return 0;
}
```

CONDITIONAL OPERATOR/ TERNARY OPERATOR

- It is a ternary operator that works on 3 operands. It is an alternate of if-else.
- Syntax:

condition ? True: false

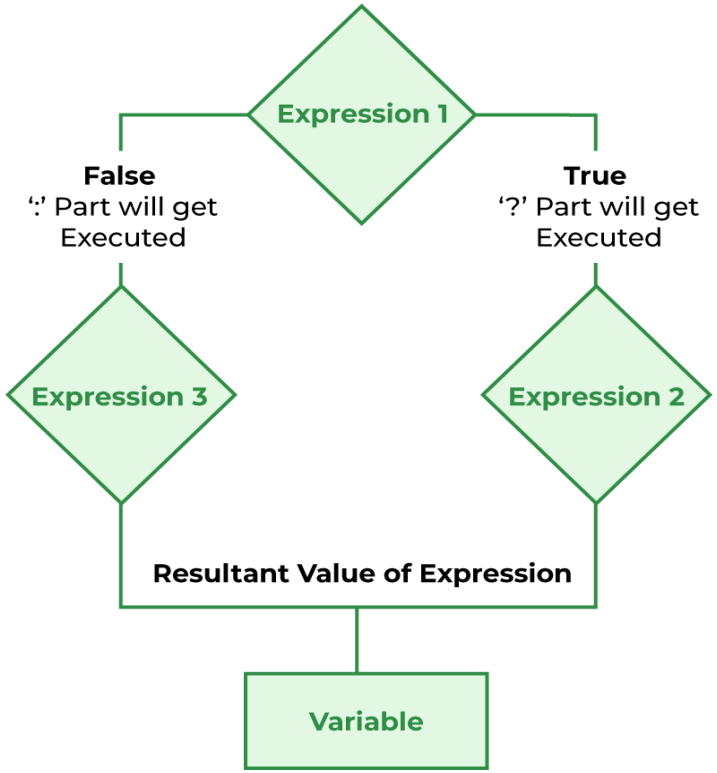
Exp1? Exp2: Exp3

If the Exp 1 is true then Exp 2 will be evaluated. If the Exp1 is false then Exp 3 will be evaluated.

variable= Exp1? Exp2: Exp3

CONDITIONAL OPERATOR/ TERNARY OPERATOR

- variable= Exp1? Exp2: Exp3



CONDITIONAL OPERATOR

// write a program to check whether 2 numbers are equal or not using conditional operator

```
#include<stdio.h>

void main()  {

int a,b;

printf("enter 2 numbers");

scanf("%d%d",a,&b);

a==b ? printf("equal") : printf("not equal");   }
```

BITWISE OPERATOR

& (bitwise AND): The result of AND is 1 only if both bits are 1.

& (bitwise OR): The result of AND is 1 only if any of the two bits is 1.

^ (bitwise XOR): XOR is 1 if the odd number of 1's are present or if the two bits are different.

<< (left shift) : Takes two numbers, the left shifts the bits of the first operand, and the second operand decides the number of places to shift.

BITWISE OPERATOR

>> (right shift): Takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift.

~ (bitwise NOT): Takes one number and inverts all bits of it.

BITWISE OPERATOR

Truth table of Bitwise operator

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

EXAMPLE OF BITWISE OPERATOR

```
#include <stdio.h>

int main()   {   int a = 5, b = 9;

    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 = ~a);

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

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

    return 0;  }
```

Output:

```
a&b = 1
a|b = 13
a^b = 12
~a = -6
b<<1 = 18
b>>1 = 4
```

UNARY OPERATOR

Increment (++)

Decrement (—)

UNARY OPERATOR

Increment (++)

The increment operator (++) is used to increment the value of the variable by 1. The increment can be done in two ways:

Preincrement:

The operator precedes the operand (e.g., ++a). The value of the operand will be changed before execution of the statement or before it is used.

UNARY OPERATOR

Postincrement:

The operator follows the operand (e.g., a++). The value operand will be altered after execution of statement or after it is used.

UNARY OPERATOR

//program on pre-increment

```
#include <stdio.h>

 int main()  {

    int a = 5;

    int b = ++a;

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

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

    return 0;     }
```

Output:

a=6

b=6

UNARY OPERATOR

//program on post-increment

```
#include <stdio.h>

 int main()  {

    int a = 5;

    int b = a++;

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

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

    return 0;     }
```

Output:

a=6

b=5

UNARY OPERATOR

Decrement (--)

The decrement operator (--) is used to decrement the value of the variable by 1. The decrement can be done in two ways:

Predecrement:

The operator precedes the operand (e.g., --a). The value of the operand will be changed before execution of the statement or before it is used.

UNARY OPERATOR

Postdecrement:

The operator follows the operand (e.g., a--). The value operand will be altered after execution of statement or after it is used.

UNARY OPERATOR

//program on pre-decrement

```
#include <stdio.h>

 int main()  {

    int a = 5;

    int b = --a;

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

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

    return 0;     }
```

Output:

a=4

b=4

UNARY OPERATOR

//program on post-decrement

```
#include <stdio.h>

 int main()  {

    int a = 5;

    int b = a--;

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

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

    return 0;     }
```

Output:

a=4

b=5

SIZEOF OPERATOR

It is a compile-time unary operator which can be used to compute the **size of its operand.**

Syntax:

sizeof(Expression);

where ‘Expression‘ can be a data type or a variable of any type.

Return: It returns the size of the given expression.

SIZEOF OPERATOR

```
// sizeof operator

#include <stdio.h>

int main()    {

    printf("%d\n", sizeof(char));

    printf("%d\n", sizeof(int));

    printf("%d\n", sizeof(float));

     return 0;   }
```

Output:
1
2
4

Note: *sizeof() may give different output according to machine, program on a 16-bit compiler.*

ADDRESS OF OPERATOR

This operator returns an integer value which is the address of its operand in the memory.

Syntax:

&operand

ADDRESS OF OPERATOR

```
// C program to illustrate the use of address operator
#include <stdio.h>
  int main()
{
    int x = 100;
    printf("The address of x is %u", &x);
    return 0;    }
```

Output:

The address of x is 6422044

PRECEDENCE OF OPERATORS

Each operator in C has a precedence associated with it.

In a compound expression, if the operators involved are of different precedence, the operator of higher precedence is evaluated first.

Compound expression:

a+b+c*d

PRECEDENCE OF OPERATORS

Each operator in C has a precedence associated with it.

In a compound expression, if the operators involved are of different precedence, the operator of higher precedence is evaluated first.

ASSOCIATIVITY OF OPERATORS

KEY POINTS:

An operator can be either left-to-right associative or right-to-left associative.

The operators with same precedence always have the same associativity.

If operators are left-to-right associative, they are applied in left-to-right order i.e. the operator which appears towards left will be evaluated first.

ASSOCIATIVITY OF OPERATORS

KEY POINTS:

If they are right-to-left associative, they will be applied in the right-to-left order.

The multiplication and the division operators are left-to-right associative. Hence, in expression 2*3/5, the multiplication operator is evaluated prior to the division operator as it appears before the division operator in left-to-right order.

PRECEDENCE AND ASSOCIATIVITY TABLE

Precedence	Operator	Description	Associativity
1	()	Parentheses (function call)	Left-to-Right
	[]	Array Subscript (Square Brackets)	
	.	Dot Operator	
	->	Structure Pointer Operator	
	++ , —	Postfix increment, decrement	
	++ / —	Prefix increment, decrement	
	+ / –	Unary plus, minus	
	! , ~	Logical NOT, Bitwise complement	

PRECEDENCE AND ASSOCIATIVITY TABLE

2	(type)	Cast Operator	Right-to-Left
	*	Dereference Operator	
	&	Addressof Operator	
	sizeof	Determine size in bytes	
3	*,/,%	Multiplication, division, modulus	Left-to-Right
4	+/-	Addition, subtraction	Left-to-Right
5	<< , >>	Bitwise shift left, Bitwise shift right	Left-to-Right

PRECEDENCE AND ASSOCIATIVITY TABLE

6	< , <=	Relational less than, less than or equal to	Left-to-Right
	> , >=	Relational greater than, greater than or equal to	
7	== , !=	Relational is equal to, is not equal to	Left-to-Right
8	&	Bitwise AND	Left-to-Right
9	^	Bitwise exclusive OR	Left-to-Right
10	|	Bitwise inclusive OR	Left-to-Right
11	&&	Logical AND	Left-to-Right

PRECEDENCE AND ASSOCIATIVITY TABLE

12	||	Logical OR	Left-to-Right
13	?:	Ternary conditional	Right-to-Left
14	=	Assignment	Right-to-Left
	+= , -=	Addition, subtraction assignment	
	*= , /=	Multiplication, division assignment	
	%= , &=	Modulus, bitwise AND assignment	
	^= , |=	Bitwise exclusive, inclusive OR assignment	
	<<=, >>=	Bitwise shift left, right assignment	

PRECEDENCE AND ASSOCIATIVITY TABLE

15	,	comma (expression separator)	Left-to-Right