operators

Types of operators

1) Arithmetic
2) Increment and Decrement
3) Relational
4) Logical
5) Assignment
6) conditional
7) Comma
8) sizeof
9) Bitwise
10) others

1) Arithmetic operators

Arithmetic operators are used for numeric calculations.

They are of two types

A) Unary B) Binary

A) Unary

Unary operators require only one operand for Example

+x, -y

Here '-' changes the sign of operand y.

B) Binary

It requires two operands.

operator	Purpose
+	Addition
−	Substraction
*	Multiplication
/	Division
%	Modulas (gives the remainder of integer divsion)

⇒ % operator cannot be applied with floating point operands.

2) Assignment operator :-

A value can be stored in a variable using Assignment operator

for example

⇒ x = 10;

Here the value 10 is stored in variable x.

⇒ x = y = 20.

Here x and y both have value 20.

⇒ when the variable on left hand side also occurs on Right hand side we can avoid writing the variable twice by using compound assignment

operator

for example

$x = x + 5$ can also be written as

$x += 5$

$\Rightarrow x -= 5$ is equivalent to $x = x - 5$.

$\Rightarrow x *= 5$ " " $x = x * 5$

$\Rightarrow x /= 5$ $x = x / 5$

3) Increment and Decrement operators

The increment and decrement operators are unary operators because they operate on single operand.

$\Rightarrow$ (++) $\Rightarrow$ The increment operator increases the value by 1. while (--) it decreases the value by 1.

for example

$++x \Rightarrow x = x + 1$

$--x \Rightarrow x = x - 1$

$\Rightarrow$ * These operators should be used only with variables; they can't be used with constants or expressions

for example

$++5 \Longrightarrow$ invalid

$++(x+y+z) \Longrightarrow$ _____

These operators are of two types

1) Prefix (Increment/decrement operators)
2) Postfix (Increment/decrement operators)

1) Prefix Increment / decrement (eg ++x or --x)

Here first the value is incremented/decremented then the new value is used in the operation

for example.

```
x = 3
⇒ y = ++x  // means first increment the value of x by 1, then asign the value of x to y.
```

⇒ This statement is ~~equa~~ equivalent to the following statements

```
x = 3
x = x + 1
y = x
```

Now the value of x = 4 and y = 4.

```
x = 3
⇒ y = --x  // means first decrement the value of x by 1, then assign the value of x to y.
```

```
⇒ x = 3
  x = x - 1
  y = x
```

* Now the value of x = 2 and y = 2.

2) Postfix Increment / Decrement (for example x++, x--)

Here the value of variable is used in the operation and then increment/decrement is performed.

for Example

```
x = 3
y = x++; // means first the value of
           x is assigned to y and
           then value of x is increased
```

⇒
```
x = 3
y = x
x = x + 1
```

Now the value of x = 4 and y = 3

```
x = 3
y = x--; // means first the value of
           x is ~~decreased~~ assigned to y
           and than the value is
           decreased.
```

⇒
```
x = 3
y = x
x = x - 1
```

Now the value of x = 2 and y = 3.

4) Relational Operators

Relational operators are used to compare values of two expressions. An expression that contains relation operators is called relational expression. If the relation is true then the value of relational Expression is 1 and if the relation is false the the value of expression is 0

operators	Meaning
<	less than
<=	less than equal to
!=	Not equal to
>	greater than
>=	greater than equal to

for example:- Let x = 9, y = 5.

Expression	Relation	Value.
a < b	false	0
a <= b	False	0
a != b	True	1
a > b	True	1
a >= b	True	1

Logical and boolean operators

An expression that combines two or more expressions is termed as a logical expression. These operators returns 0 for false and 1 for True

C has three logical operators

Operator	Name
&&	And
||	OR
!	Not

AND (&&) operator :-

This operator returns the net result true if both the conditions are true otherwise the result is false.

Truth Table

Condition 1	Condition 2	Result
False	False	False
False	True	False
True	False	False
True	True	True

For Example :-

Let a=10, b=5, c=0.

and (a==10) && (b<=a)

Here both conditions a==10 and b<=a are true so the result will be true. Since the logic operators returns 1 for true.

(a==10) && (b>a) → True && False → False

(b>=a) && (b==3) → False && False → False

⇒ (a&&b) → true && true → true

⇒ (a&&c) → true && false → false.

In the last two expressions we have taken only variables. ~~Sou~~ since nonzero values are regarded as true and zero value is regarded as false.

OR (||) operator

This operator gives the net result false if both the conditions have the value false. otherwise the result is true.

Truth Table

condition 1	condition 2	Result.
False	False	False
False	True	True
True	False	True
True	True	True.

Let a=10, b=5, c=0.

then (a>=b) || (b>15)

Result is True as one condition is True

a||b → true || true → true.
a||c → true || false → true
(a<9) || (b>10) → false || false → false

Not (!) operator

This is unary operator and it negates the value of the condition. If the value of condition is false that it gives the result true (and vice versa)

Truth table

Condition	Result
false	True
True	False

Let a=10, b=5, c=0.

then !(a==10)

as the value of condition a==10 is true. Not operator negates the value of the condition, thus the result is false.

!a → !true → false

!(b>c) → !true → false.

!(a && b) → !false → True.

Conditional operator:

Also known as Ternary operator (? and :) which requires three expressions as operands

Syntax

Test Expression ? Expression1 : Expression2.

Firstly the Test Expression is evaluated

(i) If Test Expression is true, then Expression 1 is evaluated and it becomes the value of the overall conditional Expression.

(ii) If Test expression is false, then Expression 2 is Evaluated.

for Example $a=5, b=8$

max = a>b ? a : b.

First the Test Expression (a>b) is evaluated It is false so the value of b will become the value of conditional Expression. and it is assigned to max.

Comma operator: The comma operator (,) is used to permit different Expressions to appear in situations where only one expression would be used. The Expression are seperated by the comma operator.

for example

```
int x, y;
```

Sizeof operators

sizeof is an unary operator that gives the size of its operand in terms of bytes. the operand can be a variable, constant or any data type.

for example:-

```
int var;
printf("size of int = %u\n", sizeof(int));
printf("size of int = %u\n", sizeof(var));
```

Bitwise operators

C has the abelity to support manipulation of data at the bit level. Bitwise operator operates on integer only and they are used for operation on individual bits.

Bitwise operator	Name
&	Bitwise AND
|	Bitwise OR
~	One's Complement
<<	left shift
>>	Right shift
^	bitwise XOR.

Bitwise AND

Truth Table

A	B	Result
0	0	0
0	1	0
1	0	0
1	1	1.

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the resulting of corresponding bit is evaluated to 0.

Let a = 12
b = 25

Binary No of 12 and 15 are.

a = 12 ⇒ 00001100 (In Binary)
b = 25 ⇒ 00011001 (In Binary)

00001000 = 8 (In decimal)

Example (C code)

```
#include<stdio.h>
void main()
{
  int a=12, b=25;
  printf("Output = %d", a&b);
}
```

Output

Output = 8

Bitwise OR

Truth Table

A	B	Result
0	0	0
0	1	1
1	0	1
1	1	1.

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C programming bitwise OR operator is denoted by (|).

Let a = 12 ⇒ 00001100 (In Binary)
b = 25 ⇒ 00011001 (In Binary)

00011101 = 29 (In decimal)

Example (C prog)

```
#include<stdio.h>
Void main()
  {
   int a=12, b=25;
   printf("%d", a|b);
  }.
```

Output

29.

Bitwise XOR (exclusive OR) operator (^)

Truth Table

A	B	Result
0	0	0
0	1	1
1	0	1
1	1	0

The Result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite.

Let a = 12 ⟹ 00001100 (In Binary)
b = 25 ⟹ 00011001 (In Binary)
00010101 = 21 (In decimal)

Example (C prog)

```
#include <stdio.h>
Void main()
{
 int a=12, b=25;
 printf("%d", a^b);
}
```

Output

21.

Complements

Complements are used in the digital computers in order to simplify the subtraction operations and for the logical operations.

Binary complement

it is a unary operator. It changes 1 to 0 and 0 to 1.

for example

1 0 1 0 1. → Number

0 1 0 1 0 → 1's complement

2's complement

The 2's complement of binary Number is obtained by adding 1 to Least significant bit (LSB) of 1's complement of the number

2's complement = 1's complement + 1.

Example

1 0 1 0 1 → Number

0 1 0 1 0 → 1st complement

Add 1 + → In 1's complement (LSB)

```
  01010
+     1
-------
  01011   → 2's complement.
```

Binary Addition

It is a key for binary Substraction, multiplication, division.

Truth table

A	+	B	Sum	Carry
0	+	0	0	0
0	+	1	1	0
1	+	0	1	0
1	+	1	0	1

Example

$$\begin{array}{r} {\scriptstyle 1\ 1} \\ 0\,0\,1\,1\,0\,1\,0 \\ +\ 0\,0\,0\,1\,1\,0\,0 \\ \hline 0\,1\,0\,0\,1\,1\,0 \end{array} \quad \begin{array}{l} \rightarrow 26 \\ \rightarrow 12 \\ \\ \rightarrow 38 \end{array}$$

Substraction

Truth Table

A	−	B	Substract	Borrow.
0	−	0	0	0
1	−	0	1	0
1	−	1	0	0
0	−	1	1	1

Example

$$\begin{array}{r} {\scriptstyle 1\ 1} \\ 0\,0\,1\,1\,0\,1\,0 \\ 0\,0\,0\,1\,1\,0\,0 \\ \hline 0\,0\,0\,1\,1\,1\,0 \end{array} \quad \begin{array}{l} \rightarrow 26 \\ \rightarrow 12 \\ \\ \rightarrow 14 \end{array}$$

Binary Multiplication

A	*	B	Result
0	x	0	0
0	x	1	0
1	x	0	0
1	x	1	1

Example

```
    0 0 1 1 0 1 0
  x 0 0 0 1 1 0 0
  -----------------
    0 0 0 0 0 0 0
  0 0 0 0 0 0 0
0 0 1 1 0 1 0
0 0 1 1 0 1 0
-----------------
0 1 0 0 1 1 0 0 0
-----------------
```

Bitwise shift operators

Two type of bitwise shift operators exists in C prog. The bitwise shift operators will shift the bits either on the left-side or right side.

1) Left shift operator
2) Right shift operator

1) Left shift operator

It is an operator that shifts the no of bits to the left side.

Syntax

operand << n.

→ operand is an integer expression on which we apply the left shift operation.

→ n is the no of bits to be shifted

In case of Left shift operator, 'n' bits will be shifted on the left side. ~~with~~ The 'n' bits on the left side will be popped out and 'n' bits on right-side are filled with 0.

for Example a = 5; (In decimal)

0101 → (In Binary)

Let a<<2

→ 0101 <<2

→ 00000101 → 00010100 →

Right shift :-

It is an operator that shifts the no of bits to the right side.

Syntax

operand >> n;

where

→ operand is an integer Expression on which we apply the right shift operations

→ n is the number of bits to be shifted.

for Example

a = 7; (In decimal)

a = 0111 (In binary)

Let

a >> 2;

0000 0111 >> 2 = 00000001

Ans → 1 (In decimal)

$\frac{7}{2^2} = \frac{7}{4} = 1.$

Means

Left shift = (number * 2^n)

n → no of bits to be shifted

Right shift = (number / 2^n)

n → no of bits to be shifted

Type conversion

① Implicit type conversion

② Explicit Type conversion.

① Implicit

If the type of the two operands in an assignment expression are different, then the type of the Right hand side operand is converted to the type of left hand operand. Here if the right hand operand is of lower rank then it will be promoted to the rank of left hand operand, and if it is higher rank then it will demoted to the rank of left hand operand.

for example

```
void main()
{
  char c1, c2;
  int i1, i2;
  float f1, f2;
  c1 = 'H';
  i1 = 80.25; /* converted to int, 80 will assigned to i1 */
  f1 = 12.6.
```

```
c2 = i1; /* int converted to char */
i2 = f1; /* float converted to int */
f2 = i1; /* int converted to float */
i2 = c1; /* char converted to int */
```

You can print the values to find out Results.

② Explicit Type conversion

There may be certain situations where implicit conversions may not solve our purpose for Example

```
float z;
int x = 20, y = 3;
z = x + y;
```

The value of z will be 6.0 instead of 6.66.

In these types of cases we can specify our own conversions known as type casting or coercion. This is done with the help of cast operator.

Syntax of cast operator

(data type) Expression

Hence

```
z = (float) x / y;
```

z = 6.66 Ans.

As cast operator temporarily converted int variable x into float type that why fractional part will not lost.

Some other Examples

(int) 20.3 ⟶ 20

(float) 20/3 ⟶ 6.66

(float)(20/3) ⟶ 6.00

double (x+y+z) → result will converted to double.

(double) x+y-z → first x is converted to double and then used in expression.

Precedence and Associativity of operators

for evaluation of expressions having more than one operators, there are certain precedence and associativity rules defined in C language.

for Example

2+3*5

Here there are two operators + and *
If addition is performed first then Result will be 25 and if multiplication is performed first then result will be 17.

Let $a=2, b=6, c=9$.

1) $x = a+b<c$

Here + operator has higher precedence than < and =, so a+b will be evaluated first and at last whole value will be assigned to x.

$x = 2+6<9$

$x = 8<9$ (cond is True)

$\therefore \boxed{x=1}$

2) $x*=a+b$ where $x=5, a=2, b=6$

$\Rightarrow x = x*(a+b)$

$x = 5*(2+6)$

$\boxed{x=40}$

3) $x=5+16/2*4$

Since /, * have higher precedence than + operator. Here / and * are of same group hence Associativity is to be used.

Therefore

$x = 5+8*4$

$x = 5+32$

$x = 37$

Operator	Description	Precedence level	Associativity
( ) [ ] → .	Function call Array subscript Arrow operator Dot operator	1	Left to Right
+ - ++ - - ! ~ * & (datatype) sizeof	Unary plus Unary minus Increment Decrement Logical NOT One's complement Indirection Address Type cast Size in bytes	2	Right to Left
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
= = !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
? :	Conditional operator	13	Right to Left
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right

Operator	Description	Precedence level	Associativity
( ) [ ] → .	Function call Array subscript Arrow operator Dot operator	1	Left to Right
+ - ++ - - ! ~ * & (datatype) sizeof	Unary plus Unary minus Increment Decrement Logical NOT One's complement Indirection Address Type cast Size in bytes	2	Right to Left
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
= = !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
? :	Conditional operator	13	Right to Left
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right