Unit-4
Operators and Expression

Operators

- An operator is a symbol that operates on single or multiple data items.
- Used in program to perform certain mathematical or logical manipulations.
  E.g. In a simple expression **2+3**, the symbol "+" is called an operator which operates on two data items 2 and 3.
  - The data items that operator act upon are called **operands**.

Expression

- An expression is a combination of variables, constants and operators written according to syntax of the language.
  E.g. 7+8, x+y*z, a>b

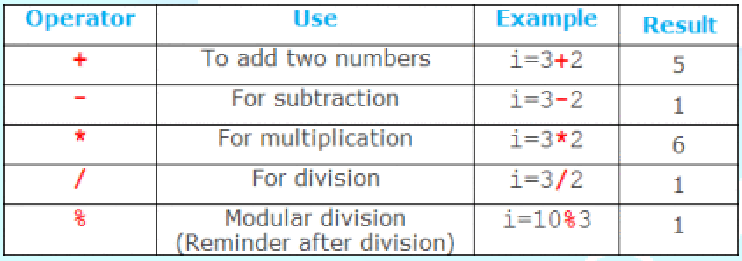
Types of operator

C operators can be classified into following types:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and Decrement Operators
- Conditional Operators
- Bitwise Operators
- Special Operators

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations. There are five arithmetic operators:



Operator	Use	Example	Result
+	To add two numbers	i=3+2	5
-	For subtraction	i=3-2	1
*	For multiplication	i=3*2	6
/	For division	i=3/2	1
%	Modular division (Reminder after division)	i=10%3	1

Division Rule:

- int/int = int
- float/float = float
- int/float = float
- float/int = float

Note: For modulo operator, the sign of the result is always the sign of the first operand.
E.g. 10%3=1, -10%3=-1, -10%-3=-1, 10%-3=1

```
/* Program to Perform Arithmetic Operations in C */
#include<stdio.h>
int main()
{
 int a = 12, b = 3;
 int add, sub, mul, div, mod;
 add = a + b;
 sub = a - b;
 mul = a * b;
 div = a / b;
 mod = a % b;
 printf("Addition of two numbers a, b is : %d\n", add);
 printf("Subtraction of two numbers a, b is : %d\n", sub);
 printf("Multiplication of two numbers a, b is : %d\n", mul);
 printf("Division of two numbers a, b is : %d\n", div);
 printf("Modulus of two numbers a, b is : %d\n", mod);
}
```

Relational Operators

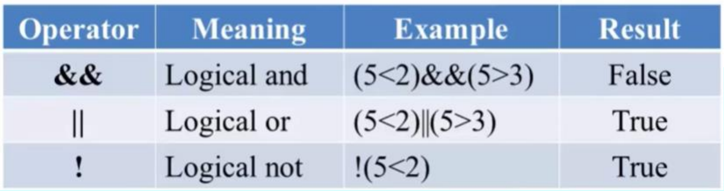
- Relational operators are used to compare two operands and taking decisions based on their relation.
- Result of relational expression is either True(1) or False(0).
- Relational operators are used in decision making and loops.
- Relational operators are:

OPERATOR	MEANING	EXAMPLE	RESULT
<	Less than	1<2	True
>	Greater than	1>2	False
<=	Less than or equal to	1<=2	True
>=	Greater than or equal to	1>=2	False
==	Equal to	1==2	False
!=	Not equal to	1!=2	True

```
/* Program to compare two numbers whether they are equal or not in C */
#include <stdio.h>
int main()
{
   int m=40, n=20;
   if (m == n)
   {
      printf("m and n are equal");
   }
   else
   {
      printf("m and n are not equal");
   }
 }
```

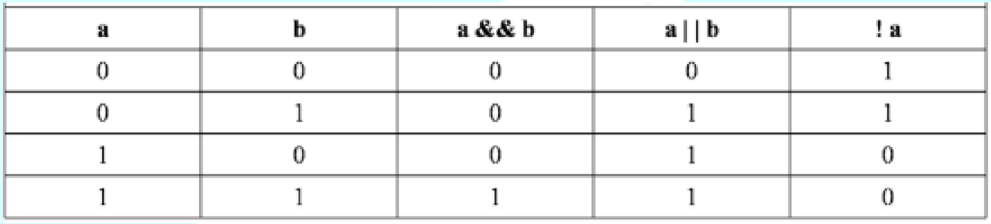
Logical Operators

- Logical operators are used to compare logical and relational expression.
- The operands of logical operators must be either Boolean value (1 or 0) or expression that produces Boolean value.
- The output of these operators is always 0 (flase) or 1 (true).
- The logical operators are:



Operator	Meaning	Example	Result
&&	Logical and	(5<2)&&(5>3)	False
||	Logical or	(5<2)||(5>3)	True
!	Logical not	!(5<2)	True

Truth table for logical operators:



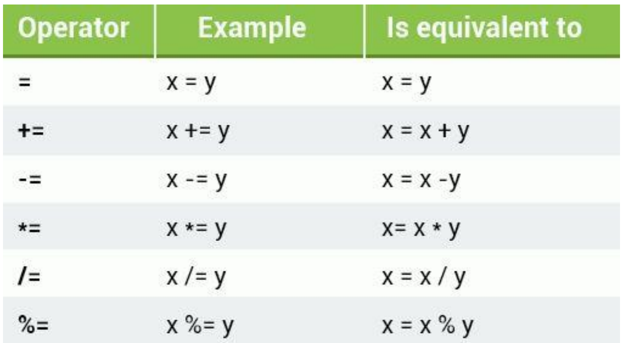
a	b	a && b	a | | b	! a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

/* C program to demonstrate working of logical operators */

```
#include <stdio.h>
int main()
{
   int a = 10, b = 4, c = 10, d = 20;

   // logical AND example
   if (a > b && c == d)
      printf("a is greater than b AND c is equal to d\n");
   else
      printf("AND condition not satisfied\n");

   // logical OR example
   if (a > b || c == d)
      printf("a is greater than b OR c is equal to d\n");
   else
      printf("Neither a is greater than b nor c is equal to d\n");

   // logical NOT example
   if (!a)
      printf("a is zero\n");
   else
      printf("a is not zero");

   return 0;
}
```

Assignment Operator

- Assignment operators are used to assign the result of an expression to a variable.
- The mostly used assignment operator is '='.
- C also supports shorthand assignment operators which simplify operation with assignment.



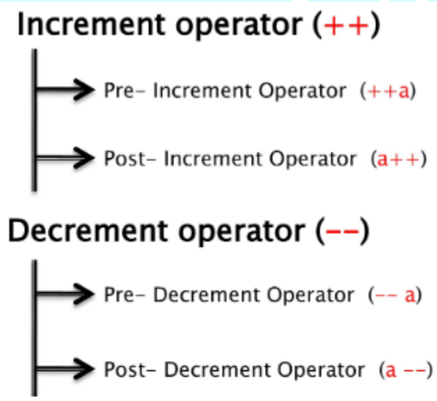
Operator	Example	Is equivalent to
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x -y
*=	x *= y	x= x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

/* program to demonstrate working of Assignment operators */

```
#include <stdio.h>
int main()
{
    int a = 10;
    printf("Value of a is %d\n", a);          //10
    a += 10;
    printf("Value of a is %d\n", a);          //20
    a -= 10;
    printf("Value of a is %d\n", a);           //10
    a *= 10;
    printf("Value of a is %d\n", a);          //100
    a /= 10;
    printf("Value of a is %d\n", a);           //10
    return 0;
}
```

Increment and Decrement Operators

- Increment operator is used to increase the value of an operand by 1.
- Decrement operator is used to decrease the value of an operand by 1.


Increment operator (++)
Pre- Increment Operator (++a)
Post- Increment Operator (a++)
Decrement operator (--)
Pre- Decrement Operator (-- a)
Post- Decrement Operator (a --)

Pre-increment operator (++a): the value is incremented first and then the expression is evaluated.

E.g. a= 10; b=++a; after this statement, a= 11, b = 11.

Post-increment operator (a++): the expression is evaluated first and then the value is incremented.

E.g. a= 10; b=a++; after this statement, a= 11, b = 10.

Pre-decrement operator (- -a): the value is decremented first and then the expression is evaluated.

E.g. a= 10; b=--a; after this statement, a= 9, b = 9.

Post-decrement operator (a- -): the expression is evaluated first and then the value is decremented.

E.g. a= 10; b=a--; after this statement, a= 9, b = 10.

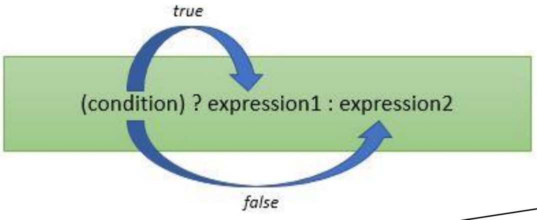
/* program to demonstrate working of increment and decrement operators */

```
#include <stdio.h>
int main()
{
    int a = 5;
    int b = 6;
    printf("a=%d, b=%d",a,b);   //a=5, b=6
    b=++a;
    printf("a=%d, b=%d",a,b);   //a=6,b=6
    b=a++;
    printf("a=%d, b=%d",a,b);   //a=7,b=6
    b=a--;
    printf("a=%d, b=%d",a,b);   //a=6,b=7
    b=--a;
    printf("a=%d, b=%d",a,b);   //a=5, b=5
    return 0;
}
```

Conditional Operator (Ternary Operator)

- It takes three arguments.
- Conditional operators return one value if condition is true and returns another value if condition is false.

Syntax: (condition) ? value_if_true : value_if_false



E.g. (a>b) ? 1 : 0;

```
if a>b
    return 1;
else
    return 0;
```

Q. Write a program to read two numbers from user and determine the larger number using conditional (ternary) operator.

```
#include <stdio.h>
int main()
{
  int n1, n2, larger;
  printf("Enter two numbers:");
  scanf("%d%d",&n1,&n2);
  larger = (n1>n2)?n1:n2;
  printf("The larger number is %d", larger);
  return 0;
}
```

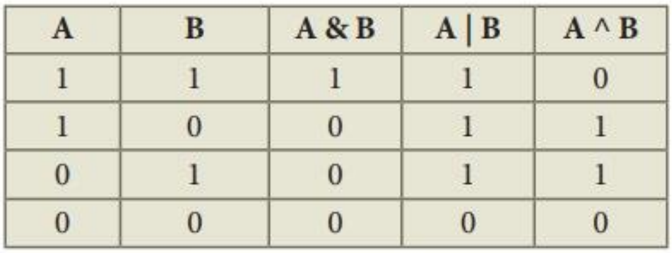
Bitwise Operator

- Bitwise operators are used for manipulating data at bit level.
- These operators are used for testing the bits or shifting them to the left or to the right.
- Can be applied only to integer-type operands and not to float or double.
- Three types of bitwise operators:
  (i) Bitwise logical operators
  (ii) Bitwise shift operators
  (iii) One's compliment operator

Bitwise logical operators:

- Performs logical tests between two integer-type operands.
- These operators work on their operands bit-by-bit starting from the least significant (i.e. rightmost) bit.
- Three logical bitwise operators:
  - ***Bitwise AND (&):*** The result of ANDing operation is 1 if both the bits have a value 1; otherwise it is 0.
  - ***Bitwise OR (|):*** The result of ORing operation is 1 if either of the bits have value of 1; otherwise it is 0.
  - ***Bitwise XOR (^):*** The result of exclusive ORing operations is 1 only if one of the bits have a value of 1; otherwise it is 0.

Truth table for bitwise operators (AND, OR, XOR)

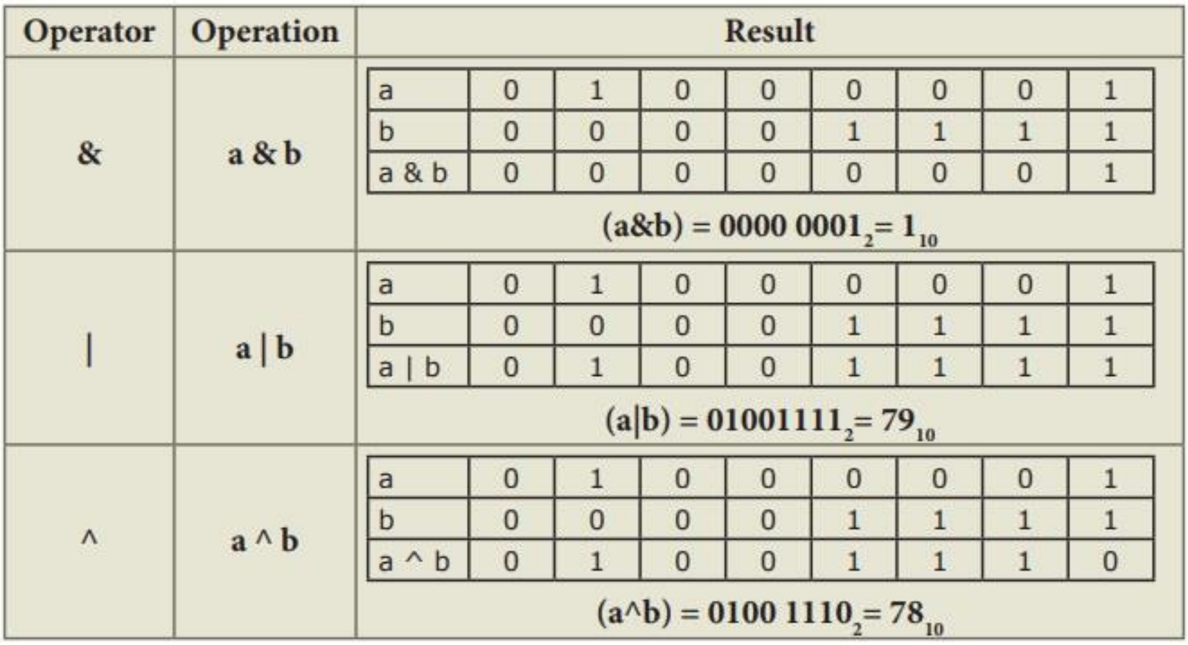


A	B	A & B	A | B	A ^ B
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

E.g.

If a = 65, b=15

Equivalent binary values of 65 = 0100 0001; 15 = 0000 1111



Operator	Operation	Result								
&	a & b	a	0	1	0	0	0	0	0	1
		b	0	0	0	0	1	1	1	1
		a & b	0	0	0	0	0	0	0	1
		$(a\&b) = 0000\ 0001_2 = 1_{10}$								
|	a | b	a	0	1	0	0	0	0	0	1
		b	0	0	0	0	1	1	1	1
		a | b	0	1	0	0	1	1	1	1
		$(a\|b) = 01001111_2 = 79_{10}$								
^	a ^ b	a	0	1	0	0	0	0	0	1
		b	0	0	0	0	1	1	1	1
		a ^ b	0	1	0	0	1	1	1	0
		$(a\hat{}b) = 0100\ 1110_2 = 78_{10}$								

Bitwise shift operators:

- Are used to move bit patterns either to left or to the right.
- There are two bitwise shift operators:

▪ ***Left shift(<<):*** Causes the operand to be shifted to the left by n positions.

operand<<n

The leftmost n bits in the original bit pattern will be lost and the rightmost n bits empty position will be filled with 0's.

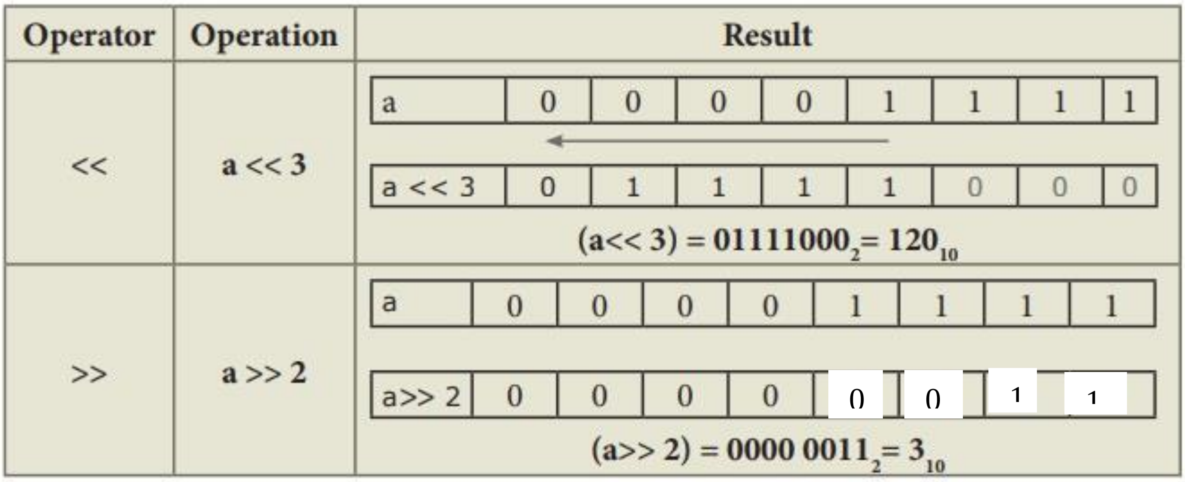
▪ ***Right shift(>>):*** Causes the operand to be shifted to the right by n positions.

operand<<n

The empty leftmost n bits positions will be filled with 0's, if the operand is an unsigned integer.

E.g.

If a =15; Equivalent binary value of a is 0000 1111



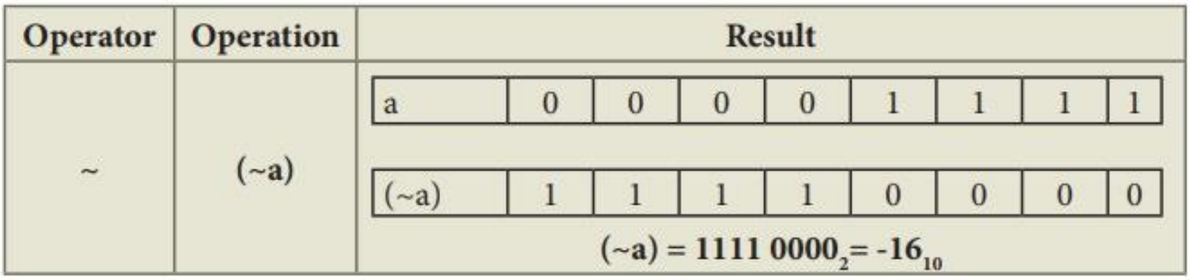
Operator	Operation	Result								
<<	a << 3	a	0	0	0	0	1	1	1	1
		a << 3	0	1	1	1	1	0	0	0
		$(a<< 3) = 01111000_2 = 120_{10}$								
>>	a >> 2	a	0	0	0	0	1	1	1	1
		a>> 2	0	0	0	0	0	0	1	1
		$(a>> 2) = 0000\ 0011_2 = 3_{10}$								

***Bitwise one's complement operator*:**

- It is a unary operator which inverts all the bits represented by its operand. This means that all 0s becomes 1s and 1s becomes 0s.

E.g.

If a =15; Equivalent binary value of a is 0000 1111



Operator	Operation	Result								
~	(~a)	a	0	0	0	0	1	1	1	1
		(~a)	1	1	1	1	0	0	0	0
		$(\sim a) = 1111\ 0000_2 = -16_{10}$								

/* program to demonstrate working of bitwise operator */

```
#include <stdio.h>
void main()
{
   int a=65,b=15,AND, OR, XOR;
   AND = a&b;
   OR = a|b;
   XOR = a^b;
   printf("AND of a and b=%d\n",AND);
   printf("OR of a and b=%d\n",OR);
   printf("XOR of a and b=%d\n",XOR);
}
```

```
#include <stdio.h>
void main()
{
   unsigned int a=15, left, right;
   left = a<<3;
   right = a>>2;
   printf("%d\n", left);
   printf("%d\n",right);
}
```

<u>Special Operators</u>

- ***Comma operator (,)*:**

- The comma operator can be used link related expressions together.
- A comma-linked list of expression are evaluated from left-to-right and the value of the rightmost expression is the value of the combined expressions.

  E.g. X=(a=5, b=10, a+b);

  - The first assign the value 5 to a
  - Assign the value 10 to b
  - Assign sum(a+b) to X

- ***Sizeof operator***

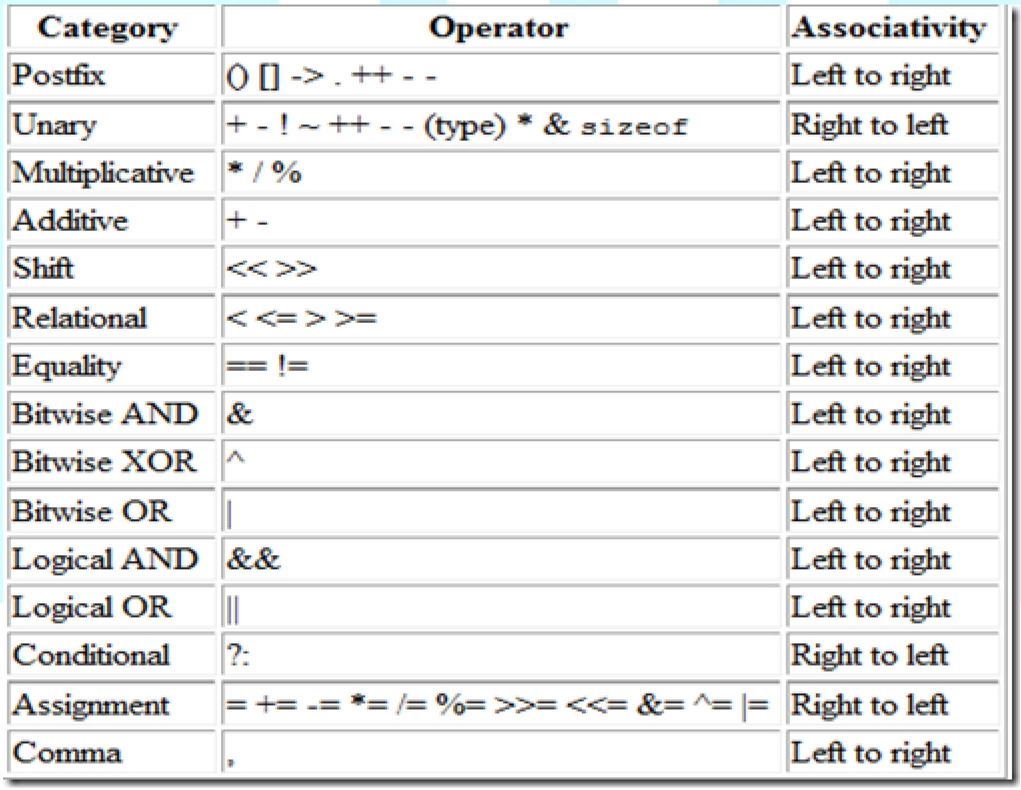
- It is used with an operand to return the number of bytes it occupies.
- The operand may be constant, variable or a data type qualifier.

E.g.

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));
    printf("Size of char=%lu byte\n",sizeof(d));
    return 0;
}
```

Operator precedence and associativity

- The precedence is used to determine how an expression involving more than one operator is evaluated.
- There are distinct level of precedence.
- The operator at the higher level of precedence are evaluated first.
- Operators of same precedence are evaluated either from “left to right” or “right to left” depending on the level also known as associativity.



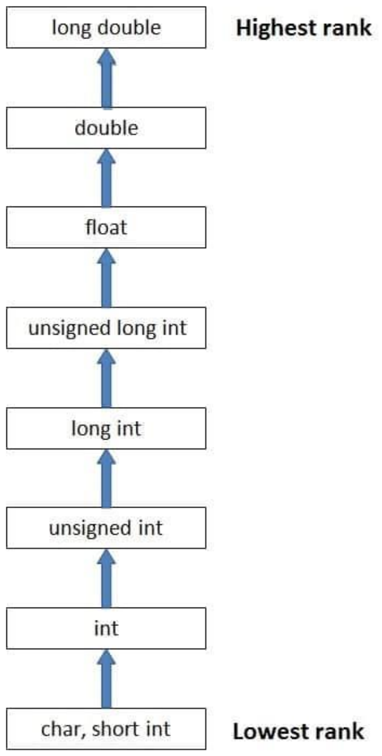
Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Type conversion in expressions

- When variables and constants of different types are combined in an expression then they are converted to same data type.
- The process of converting one predefined type into another is called type conversion.
- Type conversion in C can be classified into the following two types:

1. *Implicit Type Conversion:*

- When the type conversion is performed automatically by the compiler without programmer‘s intervention, such type of conversion is known as **implicit type conversion** or **type promotion.**
- When the expression contains different types of data items, the operand with a lower rank will be converted to the type of higher rank operand.



Implicit type conversion

E.g.

```
#include <stdio.h>
int main()
{
    int  x = 13;           // integer x
    char c = 'a';          // character c
    float sum;
    x = x + c;              // c implicitly converted to int. ASCII ('a'=97)
    sum = x + 1.0;       // x is implicitly converted to float
    printf("x = %d, sum = %f", x, sum);
    return 0;
}
```

2. Explicit Type Conversion:

- The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion.
- The explicit type conversion is also known as **type casting**.
- Type casting in C is done in the following form:

  (data_type)expression;

  where, *data_type* is any valid C data type, and *expression* may be constant, variable or expression.

E.g.

```
#include<stdio.h>
int main()
{
   float a = 1.2;
   int b;
   b = (int)a + 1;                  // a is explicitly converted to int type
   printf("Value of a is %f\n", a);
   printf("Value of b is %d\n",b);
   return 0;
}
```

Some Q &A

Q. Find the value of 'a' in each of the following statements:

int i=2, j=5, k=7

float a=1.5, b=2.5, c=3.5

i) a = c - i/j + c/k
   = 3.5 - 2/5 + 3.5/7
   = 3.5 – 0 + 0.5
   = 4

int/int = int, so 2/5= 0.4 = 0 (int part)

ii) a = (b+4)%(c+2)
   = (2.5+4)%(3.5+2)
   = 6.5%5.5
   = Not valid

iii) a = c + k%2 + b
   = 3.5 + 7%2 + 2.5
   = 3.5 + 1+ 2.5
   = 7

Q. Use the value initially assigned to the variable for each expression. Find the value of following operations.

```
        int a=8, b=5;
        float x=0.005, y=-0.01;
```

i) (x>y)&&(a>0)||(b<5);
= (0.005>-0.01)&&(8>0)||(5<5)
= (1)&&(1)||(0)
= 1 || 0
= 1

ii) (a>b)?a:b;
= (8>5)?8:5;
= 8

For more notes visit:
https://collegenote.pythonanywhere.com/