C Operators & Expressions

Operators

An ***operator*** is a symbol that tells the compiler to perform specific mathematical or logical functions on operands.
The data items on which the operators works are called as ***operands.***

- The operators that act on only one operand (variable) are called ***unary operators***.
- The operators that act on two operand (variables) are called as ***binary operators***.
- The operators that act on three operand (variables) are called as ***ternary operators.***

The C programming language has seven different categories of operators

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operator
5. Increment/ Decrement operators
6. Conditional operators
7. Bitwise operators
8. Special operators

Arithmetic operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Example

```
#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;
}
```

Output

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

Relational operators

- A relational operator checks the relationship between two operands.
- If the relation is true, it returns 1.
- If the relation is false, it returns value 0.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

Example

```
#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10;

   printf("%d == %d is %d \n", a, b, a == b);
   printf("%d == %d is %d \n", a, c, a == c);
   printf("%d > %d is %d \n", a, b, a > b);
   printf("%d > %d is %d \n", a, c, a > c);
   printf("%d < %d is %d \n", a, b, a < b);
   printf("%d < %d is %d \n", a, c, a < c);
   printf("%d != %d is %d \n", a, b, a != b);
   printf("%d != %d is %d \n", a, c, a != c);
   printf("%d >= %d is %d \n", a, b, a >= b);
   printf("%d >= %d is %d \n", a, c, a >= c);
   printf("%d <= %d is %d \n", a, b, a <= b);
   printf("%d <= %d is %d \n", a, c, a <= c);
   return 0;
```

```
}
```

Output

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

Logical operators

- These operators combines two or more relations and compare the relation between them.
- It will return 1. If it is true.
- It will return 0. If it is false.

Operator	Meaning
&&	The result of logical AND is true only if both operands are true
||	The result of logical OR is true only if anyone or both operands are true.
!	The result of logical NOT is reverse (complement) of the operand

Truth Table for logical AND

Operands		**OP1 && OP2**
OP1	**OP2**	
F	F	F
T	T	T
F	T	F
T	F	F

Truth Table for logical OR

Operands		**OP1 || OP2**
OP1	**OP2**	
F	F	F
T	T	T
F	T	T
T	F	T

Truth Table for logical NOT

Operand (OP)	**! (OP)**
F	T
T	F

Example

```
#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) is %d \n", result);

   result = (a == b) && (c < b);
   printf("(a == b) && (c < b) is %d \n", result);

   result = (a == b) || (c < b);
   printf("(a == b) || (c < b) is %d \n", result);

   result = (a != b) || (c < b);
   printf("(a != b) || (c < b) is %d \n", result);

   result = !(a != b);
   printf("!(a != b) is %d \n", result);

   result = !(a == b);
   printf("!(a == b) is %d \n", result);

   return 0;
}
```

Output

```
(a == b) && (c > b) is 1
(a == b) && (c < b) is 0
(a == b) || (c < b) is 1
(a != b) || (c < b) is 0
!(a != b) is 1
!(a == b) is 0
```

Assignment operator

An assignment operator is used for assigning a value to a variable. The most common assignment operator is = .
The variable must be on the left, and the value or expression must be right.

```
A=10;
A=B;
A= 10+12;
```

C allows combining assignment operators with arithmetic operators. Such operators are called as ***shorthand arithmetic operation.***

Operator	Example	Same as
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Example

```
#include <stdio.h>
int main()
{
   int a = 5, c;

   c = a;      // c is 5
   printf("c = %d\n", c);
   c += a;     // c is 10
   printf("c = %d\n", c);
   c -= a;     // c is 5
   printf("c = %d\n", c);
   c *= a;     // c is 25
   printf("c = %d\n", c);
   c /= a;     // c is 5
   printf("c = %d\n", c);
```

```
    c %= a;     // c = 0
    printf("c = %d\n", c);

    return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

Increment / decrement operator

These operators are used to increase or decrease the value of a variable by one unit.
++ is used to increment or increase the value by 1.
-- is used to decrement or decrease the value by 1.

Pre-increment / Pre-decrement operator

When ++ or - - is used before the variable, it is called as pre-increment/ pre-decrement operator.

++i
- -i

Post-increment / Post-decrement operator

When ++ or - - is used after the variable, it is called as post-increment/ post-decrement operator.

i++
i - -

Example

```
#include <stdio.h>
int main()
{
   int a = 10, b = 100;

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

   return 0;
}
```

Output

```
++a = 11
--b = 99
```

Conditional operator

- The conditional operator is also known as a **ternary operator**.
- The conditional statements are the decision-making statements which depends upon the output of the expression.
- It is represented by two symbols, i.e., '?' and ':'.

The syntax of ternary operator is :

```
testCondition ? expression1 : expression 2;
```

The testCondition is a boolean expression that results in either **true** or **false**. If the condition is

- true - **expression1** (before the colon) is executed
- false - **expression2** (after the colon) is executed

The ternary operator takes 3 operands (condition, expression1 and expression2).

Hence, the name **ternary operator**.

Example

```
#include <stdio.h>
int main()
{
int age;

  printf("Enter your age: ");
```

```
  scanf("%d", &age);

  (age >= 18) ? printf("You can vote") : printf("You cannot vote");

  return 0;
}
```

Output

```
Enter your age: 12
You cannot vote
```

Bitwise operators

- The bitwise operators are the operators used to perform the operations on the data at the bit-level.
- It consists of two digits, either 0 or 1.
- It is mainly used in numerical computations to make the calculations faster.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
	Shift left
>>	Shift right

Bitwise AND Operator &

- 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 result of corresponding bit is evaluated to 0.
- In C Programming, the bitwise AND operator is denoted by &.

Bitwise AND operation of two integers 12 and 25.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
```

Bitwise OR Operator |

- 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 |.

Bitwise OR operation of two integers 12 and 25.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

Bitwise XOR (exclusive OR) Operator ^

- The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite
- It is denoted by ^.

Bitwise XOR operation of two integers 12 and 25.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
^ 00011001
```

```
 ________
 00010101  = 21 (In decimal)
```

Bitwise Complement Operator ~

- Bitwise complement operator is a unary operator (works on only one operand).
- It changes 1 to 0 and 0 to 1.
- It is denoted by ~.

```
35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011

 ________
 11011100  = 220 (In decimal)
```

Right Shift Operator

- Right shift operator shifts all bits towards right by certain number of specified bits.
- It is denoted by >>.

```
212 = 11010100 (In binary)
212 >> 2 = 00110101 (In binary) [Right shift by two bits]
212 >> 7 = 00000001 (In binary)
212 >> 8 = 00000000
212 >> 0 = 11010100 (No Shift)
```

Left Shift Operator

- Left shift operator shifts all bits towards left by a certain number of specified bits.
- The bit positions that have been vacated by the left shift operator are filled with 0.
- The symbol of the left shift operator is <<.

```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 = 11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)
```

Example

```
#include <stdio.h>
int main()
{

   int a = 5, b = 9;

   printf("a = %d, 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 = ~a);
   printf("b<<1 = %d\n", b << 1);
   printf("b>>1 = %d\n", b >> 1);
  return 0;

}
```

Output

```
a = 5, b = 9
a & b = 1
a | b = 13
a ^ b = 12
~a = -6
b << 1 = 18
b >> 1 = 4
```

Special operators

Comma Operator

Comma operators are used to link related expressions together. For example:

int a, c = 5, d;

The sizeof operator

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

```
#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;
}
```

Output

```
Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
```

Operator Precedence and Associatively

An arithmetic expression without any parentheses will be evaluated from left to right using rules of precedence of operators.
There are two distinct priority levels of arithmetic operators in C#.

- High priority * / %
- Low priority + -

There basic evaluation procedure includes two left-to-right passes through the expression.

- During first pass, the high priority operators (if any) are applied as they are encountered.
- During second pass, the low priority operators (if any) are applied as they are encountered.
- Whenever the parentheses are used, the expression within parentheses assume highest priority. If two or more sets of parentheses appear one after another, the expression contained in the left-most set is evaluated first and the right-most set at last

First pass:

Step 1: x=9-12/6*(2-1)

Step 2: x=9-12/6*1

Second pass:

Step 3: x= 9-2*1

Step 4: x=9-2

Third pass:

Step 5: x=7

C Type Conversion

In C programming, we can convert the value of one data type (int, float, double, etc.) to another. This process is known as type conversion.

In C, there are two types of type conversion:

- Implicit Conversion
- Explicit Conversion

Implicit Type Conversion In C

In implicit type conversion, the value of one type is automatically converted to the value of another type.

```
#include<stdio.h>
int main()

{

  double value = 4150.12;
  printf("Double Value: %.2lf\n", value);


  int number = value;

  printf("Integer Value: %d", number);

  return 0;
}
```

Output

```
Double Value: 4150.12
Integer Value: 4150
```

Explicit Type Conversion In C

In explicit type conversion, we manually convert values of one data type to another type.

```
#include<stdio.h>

int main()
{

  int number = 35;
  printf("Integer Value: %d\n", number);


  double value = (double) number;

  printf("Double Value: %.2lf", value);

  return 0;
}
```

Output

```
Integer Value: 35
Double Value: 35.00
```