Operators and Expressions

Dr. Nachiket Tapas

Review

```
main()
{
        int a, b, c, d;
        a = 15;
        b = 10;
        c = ++a - b;
        printf("a = %d b = %d c = %d\n",a, b, c);
        d = b++ +a;
        printf("a = %d b = %d d = %d\n",a, b, d);
        printf("a/b = %d\n", a/b);
        printf("a%%b = %d\n", a%b);
        printf("a *= b = %d\n", a*=b);
        printf("%d\n", (c>d) ? 1 : 0);
        printf("%d\n", (c<d) ? 1 : 0);
}
```

Output

```
a = 16 b = 10 c = 6
a = 16 b = 11 d = 26
a/b = 1
a%b = 5
a *=b = 176
0
1
```

Review

```
main()
{
        float a, b, c, x, y, z;
        a = 9;
        b = 12;
        c = 3;
        x = a – b / 3 + c * 2 - 1;
        y = a – b / (3 + c) * (2 - 1);
        z = a – (b / (3 + c) * 2) - 1;
        printf(“x = %f\n”, x);
        printf(“y = %f\n”, y);
        printf(“z = %f\n”, z);
}
```

Output

```
x = 10.000000
y = 7.000000
z = 4.000000
```

Interesting Question

```
#include<stdio.h>
void main() {
        float no1 = 10.2;
        float no2 = 20.4;
        float sum = 0;
        sum = no1 + no2;
        printf("Sum: %f", sum);
        }
```

What is the output in this case?

Special Operators

- sizeof()
  - printf(“%d”, sizeof(int));

Did you try the following? What did you get?

```
printf(“%d”, sizeof(5.2));
```

Did you get 4 or 8?

Default decimal number

- By default, C programming considers decimal values as double.
- That is why you get output as 8 and not 4.
- Remember the value 5.2 used by us is literal constant. If you declare a float variable and find its size, you will get 4 as output.
- Another type of constant is called symbolic constant

```
#define PI 3.14
```

- Constant variable

```
const int I = 10;
```

Special Operators

- comma operator
  - value = (x=5, y=7, x+y);
  - printf(“%d”, value);

```
#include<stdio.h>
void main(){
        int value, x, y;
        value = (x=5, y=7, x+y);
        printf("%d", value);
}
```

What is the result?

- [7 + 8 { 4 × (6 + 4 × 3) × 4}]

[7 + 8 { 4 x (6 + 12) x 4 }]

[7 + 8 { 4 x 18 x 4 }]

[7 + 8 { 72 x 4 }]

[7 + 8 x 288]

[7 + 2304]

O/P: 2311

What about this?

- 45 + 3 { 34 – 18 – 14 } ÷ 3 [17 + 3 × 4 – (2 × 7)]

45 + 3 { 16 – 14 } / 3 [17 + 3 x 4 – (2 x 7)]

45 + 3 x 2 / 3 [17 + 3 x 4 – (2 x 7)]

45 + 3 x 2 / 3 [17 + 3 x 4 – 14]

45 + 3 x 2 / 3 [17 + 12 – 14]

45 + 3 x 2 / 3 [29 – 14]

45 + 3 x 2 / 3 x 15

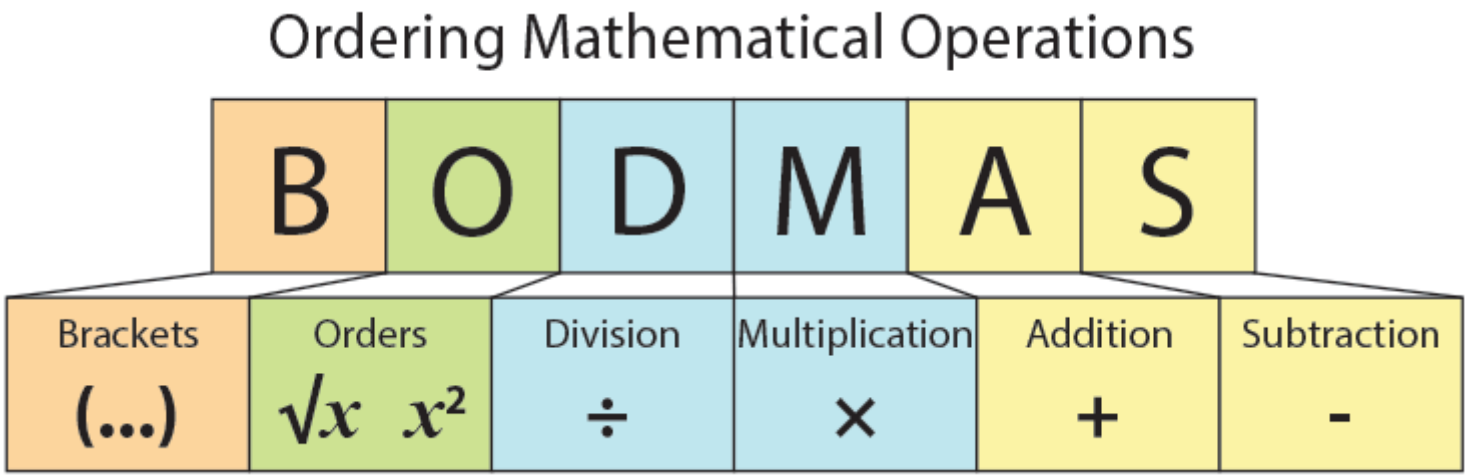
45 + 6 / 3 x 15

45 + 2 x 15

45 + 30

O/P: 75

Operator Precedence



Operator Precedence in C

( )	Parentheses (function call) (see Note 1)	left-to-right
[ ]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ –	Postfix increment/decrement (see Note 2)	
++ –	Prefix increment/decrement	right-to-left
+ –	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(*type*)	Cast (convert value to temporary value of *type*)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right

Contd.

<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

When to use associativity?

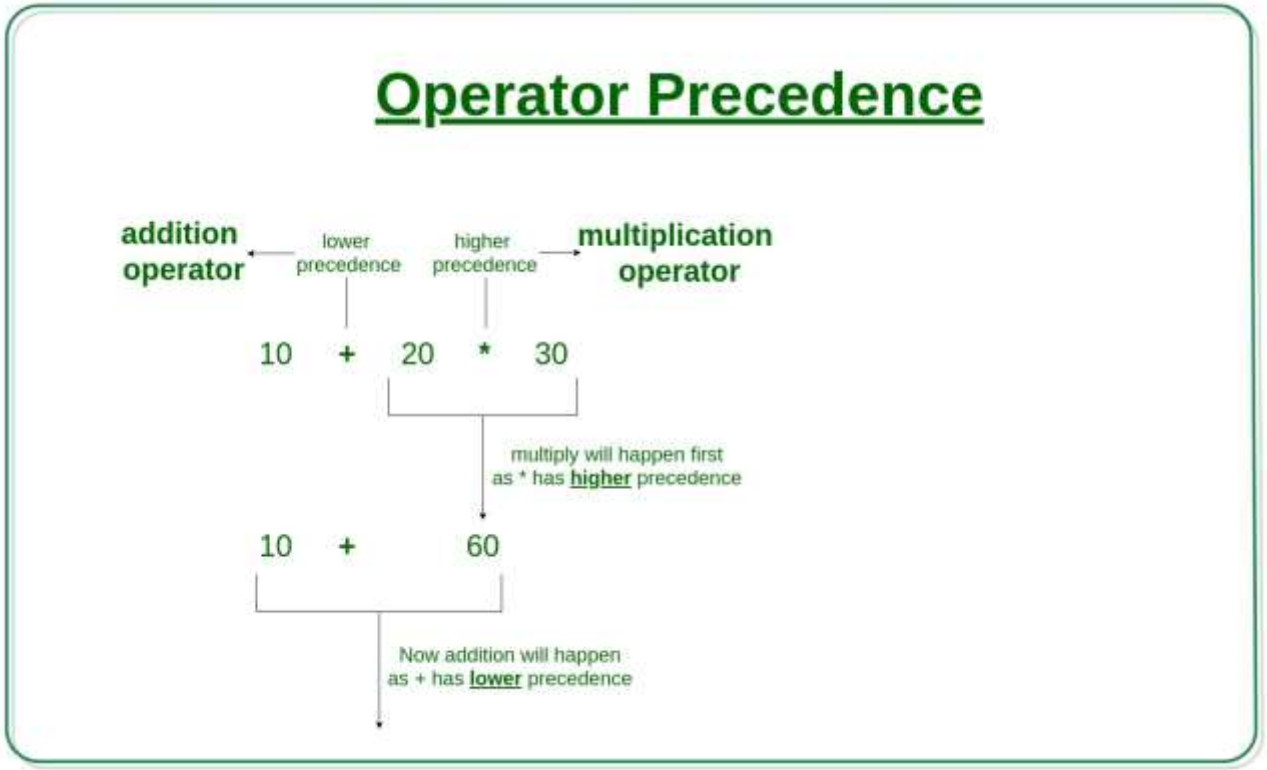
- We only use associativity when we have two or more operators that have the same precedence in an expression.
- The point to note here is that associativity is not applicable when we are defining the order of evaluation of operands with different levels of precedence.
- **All operators with the same precedence have same associativity**
  This is necessary, otherwise, there won't be any way for the compiler to decide evaluation order of expressions which have two operators of same precedence and different associativity. For example + and – have the same associativity.

Truth Table

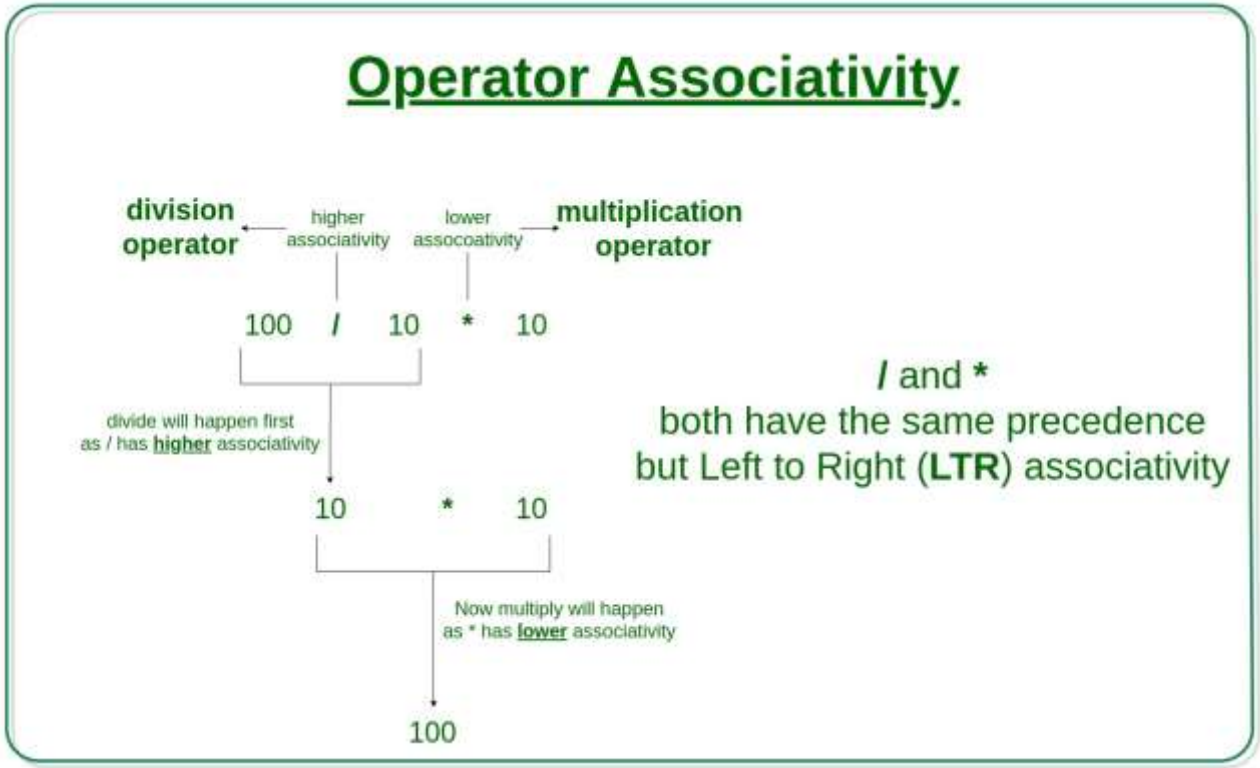
Precedence	Associativity
Different	Same
Same	Different

Precedence	Associativity
Different	Same/Different
Same	Same

Example


Operator Precedence
addition operator
lower precedence
higher precedence
multiplication operator
10 + 20 * 30
multiply will happen first as * has higher precedence
10 + 60
Now addition will happen as + has lower precedence

Example


Operator Associativity
division operator
higher associativity
lower assocoativity
multiplication operator
100 / 10 * 10
divide will happen first as / has higher associativity
10 * 10
Now multiply will happen as * has lower associativity
100
/ and *
both have the same precedence
but Left to Right (LTR) associativity

Example

- 12 + 3 – 4 / 2 < 3 + 1

12 + 3 – 2 < 3 + 1

15 – 2 < 3 + 1

13 < 3 + 1

13 < 4

O/P: 0

Example

X = -5 * 4 / 2 * 3 + -1 * 2;

Caution

- Operator precedence if not used correctly, can give absurd results.
- Always use parenthesis to define precedence.

Code

```
#include <stdio.h>
main() {
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;
   e = (a + b) * c / d; // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n", e );
   e = ((a + b) * c) / d; // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is : %d\n" , e );
   e = (a + b) * (c / d); // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is : %d\n", e );
   e = a + (b * c) / d; // 20 + (150/5)
   printf("Value of a + (b * c) / d is : %d\n" , e );
   return 0;
}
```

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50
```

Is the following true?

- 5 + 5 = = 10 || 1 + 3 = = 5
- !(5 + 5 >=10)
- 5 > 10 || 10 < 20 && 3 < 5
- 10 ! = 15 && !(10<20) || 15 > 30