

OPERATOR PRECEDENCE AND ASSOCIATIVITY

- Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence.
- For example
 - $10 + 20 * 30$ is calculated as $10 + (20 * 30)$ and not as $(10 + 20) * 30$.



CONT..

- Associativity is used when two operators of same precedence appear in an expression. Associativity can be either **Left to Right** or **Right to Left**.
- For example ‘*’ and ‘/’ have same precedence and their associativity is **Left to Right**, so the expression “100 / 10 * 10” is treated as “(100 / 10) * 10”.



- **All operators with same precedence have same associativity**
- **chaining of comparison operators is not allowed in C**
 - In Python, expression like “ $c > b > a$ ” is treated as “ $a > b$ and $b > c$ ”, but this type of chaining doesn't happen in C



OPERATORS (1)

Operator Precedence and Associativity	
Operator	Associativity
() ++(postfix) --(postfix)	left to right
+(unary) -(unary) ++(prefix) --(prefix)	right to left
* / %	left to right
+ -	left to right



OPERATORS EXAMPLE

Declarations and Initializations	
int a = 1, b = 2, c = 3, d = 4;	
Expression	Value
a * b / c	0
a * b % c + 1	3
++ a * b - c --	1
7 - - b * ++ d	17



- $a*b/c=(a*b)/c = 0$
- $a*b\%c+1=((a*b)\%c)+1 = 3$
- $++a*b-c--=((++a)*b)-(c--)=??$
- $7--b*++d=7-((-b)*(++d))=7-(-10)=17$



ASSIGNMENT OPERATORS

- C treats **=** as an operator
- `variable = Right_Hand_Side`
- Other assignment operators
 - `variable op (expression)`
 - `+=`, `-=`, `*=`, `/=`, `%=`, `>>=`, `<<=`, `&=`, `^=` and `|=`



OPERATORS (2)

Operator Precedence and Associativity	
Operator	Associativity
() ++(postfix) --(postfix)	left to right
+(unary) -(unary) ++(prefix) --(prefix)	right to left
* / %	left to right
+ -	left to right
= += -= *= /= etc	right to left



EXAMPLE

- `int x,y,z=5;`
- `x=y=z`
- `int x,y,z=5;`
- `x=z=y`



RELATIONAL OPERATORS AND EXPRESSIONS

- Relational operators are binary.
- Takes two expressions as operands and yields either the int value 1 (TRUE) or 0 (FALSE)
- The relational operators are
 - < (less than), > (greater than)
 - <= (less than or equal to), >= (greater than or equal to)
 - Same precedence, left to right associativity



OPERATORS (3)

Operator Precedence and Associativity

Operator	Associativity
() ++(postfix) --(postfix)	left to right
+(unary) -(unary) ++(prefix) --(prefix)	right to left
* / %	left to right
+ -	left to right
< > <= >=	left to right
= += -= *= /= etc	right to left



RELATIONAL EXPRESSIONS EXAMPLE

- $a < 3, a < b, a < c$
- $a - b < 0$

Values of relational expressions				
$a - b$	$a < b$	$a > b$	$a \leq b$	$a \geq b$
Positive	0	1	0	1
Zero	0	0	1	1
Negative	1	0	1	0



EQUALITY OPERATORS AND EXPRESSIONS

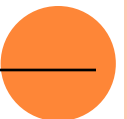
- `==` and `!=`
- Lower precedence than relational operators and left to right associativity
- Binary operators
- Yield either 1 (TRUE) or 0 (FALSE).
- What is the output of this equality expression?
 - $i + j + k == -2 * -k$ where $i = 1, j = 2, k = 3$;



OPERATORS (4)

Operator Precedence and Associativity

Operator	Associativity
() ++(postfix) --(postfix)	left to right
+(unary) -(unary) ++(prefix) --(prefix)	right to left
* / %	left to right
+ -	left to right
< > <= >=	left to right
== !=	left to right
= += -= *= /= etc	right to left



LOGICAL OPERATORS AND EXPRESSIONS

- ! (not) is unary, && (and) and || (or) are binary
- && has higher precedence than ||.
- ! has same precedence as other unary operators.
- Semantics of the ! operator

expr	!expr
Zero	1
Non-zero	0



SEMANTICS OF && AND || OPERATOR

expr1	expr2	expr1 && expr2	expr1 expr2
Zero	Zero	0	0
Zero	Non-zero	0	1
Non-zero	Zero	0	1
Non-zero	Non-zero	1	1

OPERATORS (5)

Operator Precedence and Associativity

Operator	Associativity
() ++(postfix) --(postfix)	left to right
+(unary) -(unary) ++(prefix) --(prefix) !	right to left
* / %	left to right
+ -	left to right
< > <= >=	left to right
== !=	left to right
&&	left to right
	left to right
= += -= *= /= etc	right to left

EXAMPLES OF LOGICAL OPERATORS

- `char c = 'B'; int i = 3, j = 3, k = 3;`
- `double x = 0.0, y = 2.3;`

Expression	Value
<code>i && j && k</code>	1
<code>x i && j - 3</code>	0
<code>i < j x < y</code>	1
<code>c - 1 == 'A' c + 1 == 'Z'</code>	1



COMMA OPERATOR

- Lowest Precedence, Binary operator
- Syntax: `expr1, expr2`

```
j=10;
```

```
for(i = 1; i <= N; i++)
```

```
    j--;
```

can be re-written as

```
for(i = 1, j = 10; i <= N; i++, j--)
```



EXAMPLES

- `int a=1, b=2, c=3, i=0; // comma acts as separator in this line, not as an operator ... a=1, b=2, c=3, i=0`
- `i = (a, b); // stores b into i ... a=1, b=2, c=3, i=2`
- `i = a, b; // stores a into i. Equivalent to (i = a), b; ... a=1, b=2, c=3, i=1`
- `i = (a += 2, a + b); // increases a by 2, then stores a+b = 3+2 into i ... a=3, b=2, c=3, i=5`
- `i = a += 2, a + b; // increases a by 2, then stores a into i.`
 - *Equivalent to (i = (a += 2)), a + b; ... a=5, b=2, c=3, i=5*
- `i = a, b, c; // stores a into i ... a=5, b=2, c=3, i=5`
- `i = (a, b, c); // stores c into i ... a=5, b=2, c=3, i=3`



OPERATORS (6)

Operator Precedence and Associativity

Operator	Associativity
() ++(postfix) --(postfix)	left to right
+(unary) -(unary) ++(prefix) --(prefix) !	right to left
* / %	left to right
+ -	left to right
< > <= >=	left to right
== !=	left to right
&&	left to right
	left to right
= += -= *= /= etc	right to left
, (comma operator)	left to right

PUNCTUATORS

- A symbol that has a semantic significance but does not specify an operation to be performed.
- “{“, “;”, “(“ and “)” are punctuators.
- Both operators and punctuators are collected by the compiler as tokens.

