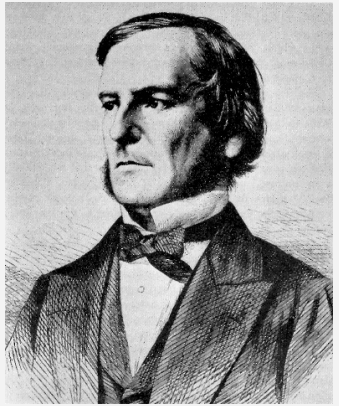
BOOLEAN OPERATIONS

Fabio Stroppa

Boolean operations

A **bit** is the minimum amount of information that we can imagine, since it only stores either value **`1`** or **`0`**, which represents either YES or NO, activated or deactivated, true or false, etc... that is: two possible states each one opposite to the other, without possibility of any shades.

Boolean operations can be performed on bits, named after the mathematicians George Boole (1815-1864).



In C++, these operators can be used with variables of any integer data type.

AND **&**: true when both true

This operation is performed between two bits, which we will call **a** and **b**. The result of applying this **AND** operation is **1** if both **a** and **b** are equal to **1**, and **0** in all other cases (i.e., if one or both of the variables is **0**).

If **a** is **true** and **b** is **true**,
then the result is **true**

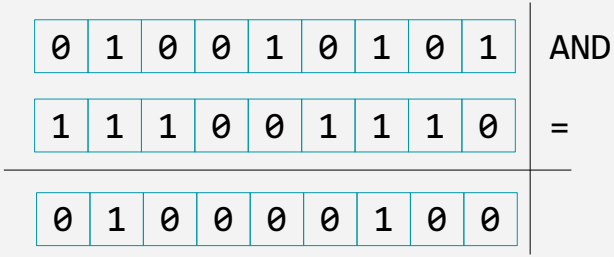
AND &: true when both true

This operation is performed between two bits, which we will call **a** and **b**. The result of applying this **AND** operation is **1** if both **a** and **b** are equal to **1**, and **0** in all other cases (i.e., if one or both of the variables is **0**).

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a&b
false	false	false
false	true	false
true	false	false
true	true	true

Truth Table



Example

OR `|`: true when one is true

This operation is performed between two bits, which we will call **`a`** and **`b`**. The result of applying this **`OR`** operation is **`1`** if either **`a`** or **`b`** or both are equal to **`1`**. If none is equal to **`1`** the result is **`0`**.

If **`a`** is **`true`** or **`b`** is **`true`**,
then the result is **`true`**

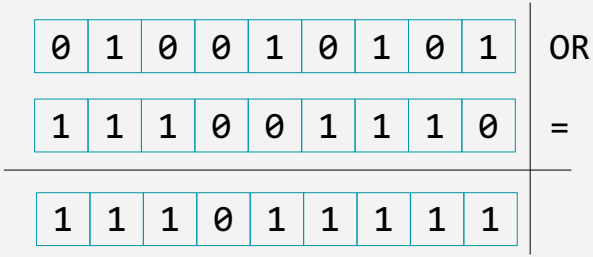
OR |: true when one is true

This operation is performed between two bits, which we will call **a** and **b**. The result of applying this **OR** operation is **1** if either **a** or **b** or both are equal to **1**. If none is equal to **1** the result is **0**.

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a|b
false	false	false
false	true	true
true	false	true
true	true	true

Truth Table



Example

XOR ^: true when different

This operation is performed between two bits, which we will call **a** and **b**. The result of applying this **XOR** operation is **1** if either one between **a** or **b** is equal to **1**, but not in the case that both are (i.e., if neither or both of them are equal to **1** the result is **0**).

If **a** is exclusively **true** or **b** is exclusively **true**, then the result is **true**

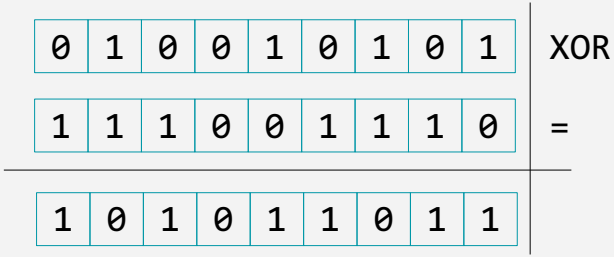
XOR ^: true when different

This operation is performed between two bits, which we will call **a** and **b**. The result of applying this **XOR** operation is **1** if either one between **a** or **b** is equal to **1**, but not in the case that both are (i.e., if neither or both of them are equal to **1** the result is **0**).

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

a	b	a^b
false	false	false
false	true	true
true	false	true
true	true	false

Truth Table



Example

NOT ~: inversion

This operation is performed on a single bit, which we will call **a**. Its result is the inversion of the actual value **a**: if it was set to **1** it becomes **0**, and if it was **0** it becomes **1**.

If **a** is **true**,
then **a** is not **false**

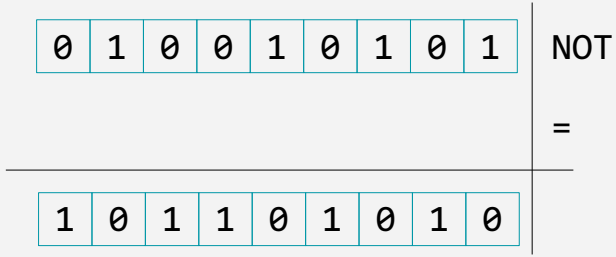
NOT ~: inversion

This operation is performed on a single bit, which we will call **a**. Its result is the inversion of the actual value **a**: if it was set to **1** it becomes **0**, and if it was **0** it becomes **1**.

a	~a
0	1
1	0

a	~a
false	true
true	false

Truth Table



Example

Theorems

- a OR 0 = a
- a OR 1 = 1
- a OR a = a
- a OR NOT(a) = 1
- a OR b = b OR a
- (a OR b) OR c = a OR (b OR c)

- a AND 0 = 0
- a AND 1 = a
- a AND a = a
- a AND NOT(a) = 0
- a AND b = b AND a
- (a AND b) AND c = a AND (b AND c)

Theorems

- `a OR (b AND c) = (a AND b) OR (a AND c)`
- `NOT(NOT(a)) = a`
- `NOT(a OR b) = NOT(a) AND NOT(b)`
- `NOT(a AND b) = NOT(a) OR NOT(b)`

De Morgan's Theorem

- `a AND b OR a = a`