Rational (Boolean) Expressions

Relational operators

Logical operators

Order of precedence

Rational (Boolean) Expressions

Expressions

- any combination of variables and constants that can be evaluated to yield a result
- typically involve *operators*
- *Examples:*

```
        5

        x

      x + y

      num++

      a = 3 + j
```

Relational Expressions

- compare operands

- used in decision making

- evaluate to *1* (true) or *0* (false)

Operand	Relational Operator	Operand
price	<	34.98

Relational Operators

Relational Operators

less than	**<**
greater than	**>**
less than or equal to	**<=**
greater than or equal to	**>=**
Equal to	**==**
Not Equal to	**!=**

a - b < 0 is equivalent to (a - b) < 0

- Expressions such as 4 < 6 is an example of **a logical (Boolean) expression**.
- When C++ evaluates a logical expression, it returns an integer value of **`1`** if the logical expression evaluates to `true`; it returns an integer value of **`0`** otherwise.

```
int x;
x= 4<6;
cout<<x ;        1
```

```
cout<<(4<6) ;      1
```

Must be between brackets

- In C++, any nonzero value is treated as `true,` and a zero value is treated as `false`.

Relational Operators

Can use any relational operator

Expression	Example
int < int	3 < 4
float > float	6.2 > 4.2
char > char	‘a’ < ‘A’
int < float	2 < 3.1
int < char	66 > ’A’
variable < arithmetic operation	x > 3+y
arithmetic operation < arithmetic operation	x+7 < y--

Relational Operators and Simple Data Types

Expression	Meaning	Value
`8 < 15`	8 is less than 15	`true`
`6 != 6`	6 is not equal to 6	`false`
`2.5 > 5.8`	2.5 is greater than 5.8	`false`
`5.9 <= 7.5`	5.9 is less than or equal to 7.5	`true`

Relational Operators and equality Operations Examples

Valid

```
a < 3
a > b
-1.1 >= (2.2 * x + 3.3)
k != -2
y == 2 * z - 5
```

Not Valid

```
a =< b                  // out of order
a <  = b                // space not allowed
a =  = b                // space not allowed
a = b                   // assignment statement
a =  = b - 1            // space not allowed
y =! z                  // this is equivalent to y = (!z)
```

Equality Operators Examples

```
void main()
{
int x=7;
int y=5;
y=! x;
cout<<y;
getch();
}
```

0

```
void main()
{
int x=7;
int y=5;
y!= x;
cout<<y;
getch();
}
```

5

```
void main()
{
int x=7;
int y=5;
cout<< (y!= x);
getch();
}
```

1

```
void main()
{
int x=7;
int y;
cout<<(y=! x);
getch();
}
```

0

Logical Operators
and
logical expressions

Logical Operators and logical expressions

- **Logical** (**Boolean**) **operators** enable you to combine logical expressions

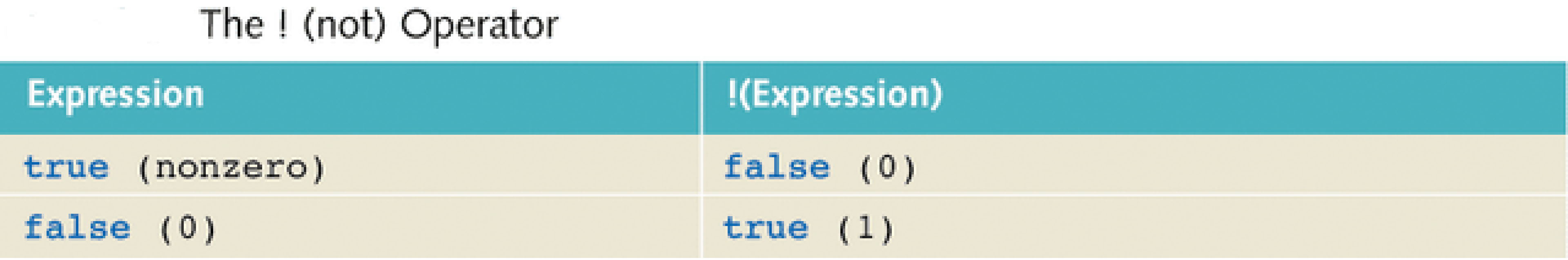
- In C++, there are three **logical** (**Boolean**) **operators**:

<table>
<tr><td>Negation</td><td>!</td></tr>
<tr><td>Logical and</td><td>&&</td></tr>
<tr><td>Logical or</td><td>||</td></tr>
</table>

- The operator ! is unary, so it has only one operand.
- The operators && and || are binary operators.

Logical Operators and logical expressions

! Variable	! x
! (Logical exp)	! (4>2)
! value	! 6
variable && variable	x && y
variable && exp	x && (5==3)
exp && exp	(3<5) && (6>33)
The same in ||	

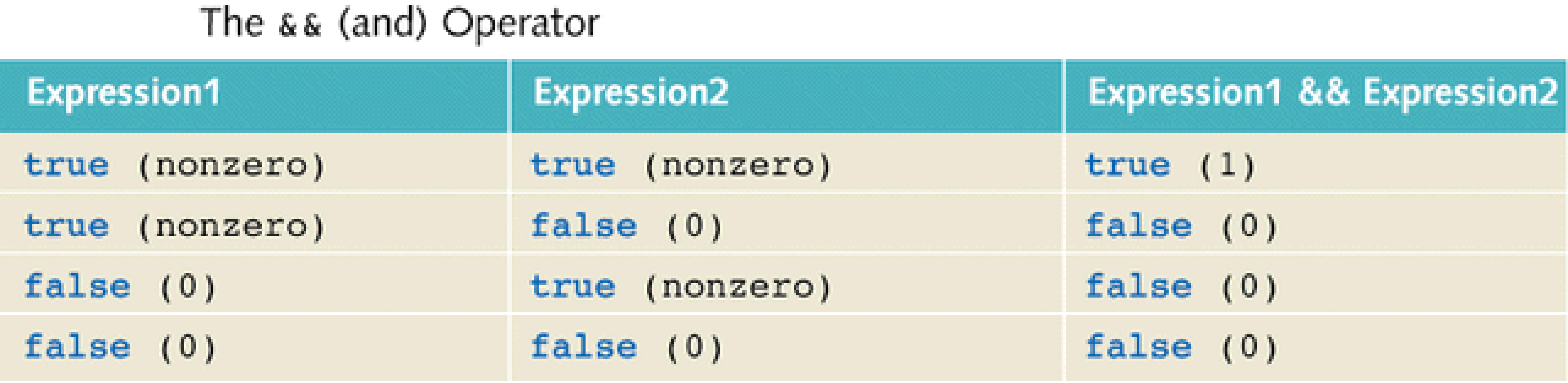


The ! (not) Operator

Expression	!(Expression)
true (nonzero)	false (0)
false (0)	true (1)

Example

Expression	Value	Explanation
`!(6 <= 7)`	`false`	Because `6 <= 7` is `true`, `!(6 <= 7)` is `false`.

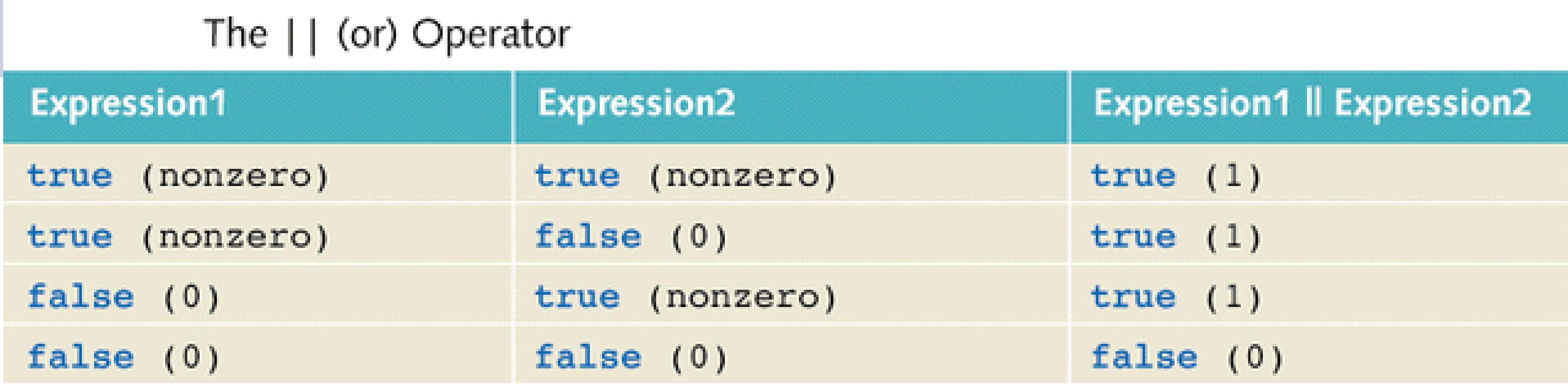


The && (and) Operator

Expression1	Expression2	Expression1 && Expression2
true (nonzero)	true (nonzero)	true (1)
true (nonzero)	false (0)	false (0)
false (0)	true (nonzero)	false (0)
false (0)	false (0)	false (0)

Example

Expression	Value	Explanation
(14 >= 5) && (-1 < 2)	true	Because (14 >= 5) is true, (-1 < 2) is true, and true && true is true, the expression evaluates to true.
(24 >= 35) && (-1 < 2)	false	Because (24 >= 35) is false, (-1 < 2) is true, and false && true is false, the expression evaluates to false.



The || (or) Operator

Expression1	Expression2	Expression1 || Expression2
true (nonzero)	true (nonzero)	true (1)
true (nonzero)	false (0)	true (1)
false (0)	true (nonzero)	true (1)
false (0)	false (0)	false (0)

Example

Expression	Value	Explanation
(14 >= 5) || (7 > 13)	true	Because (14 >= 5) is true, (7 > 13) is false, and true || false is true, the expression evaluates to true.
(24 >= 35) || (7 > 13)	false	Because (24 >= 35) is false, ((7 > 13) is false, and false || false is false, the expression evaluates to false.

Examples

Valid

a && b
a || b && c
!(a < b) && c
3 && (-2 * a + 7)

Not Valid

```
a &&                  // one operand missing
a |□| b               // extra space not allowed
```

Order of Precedence

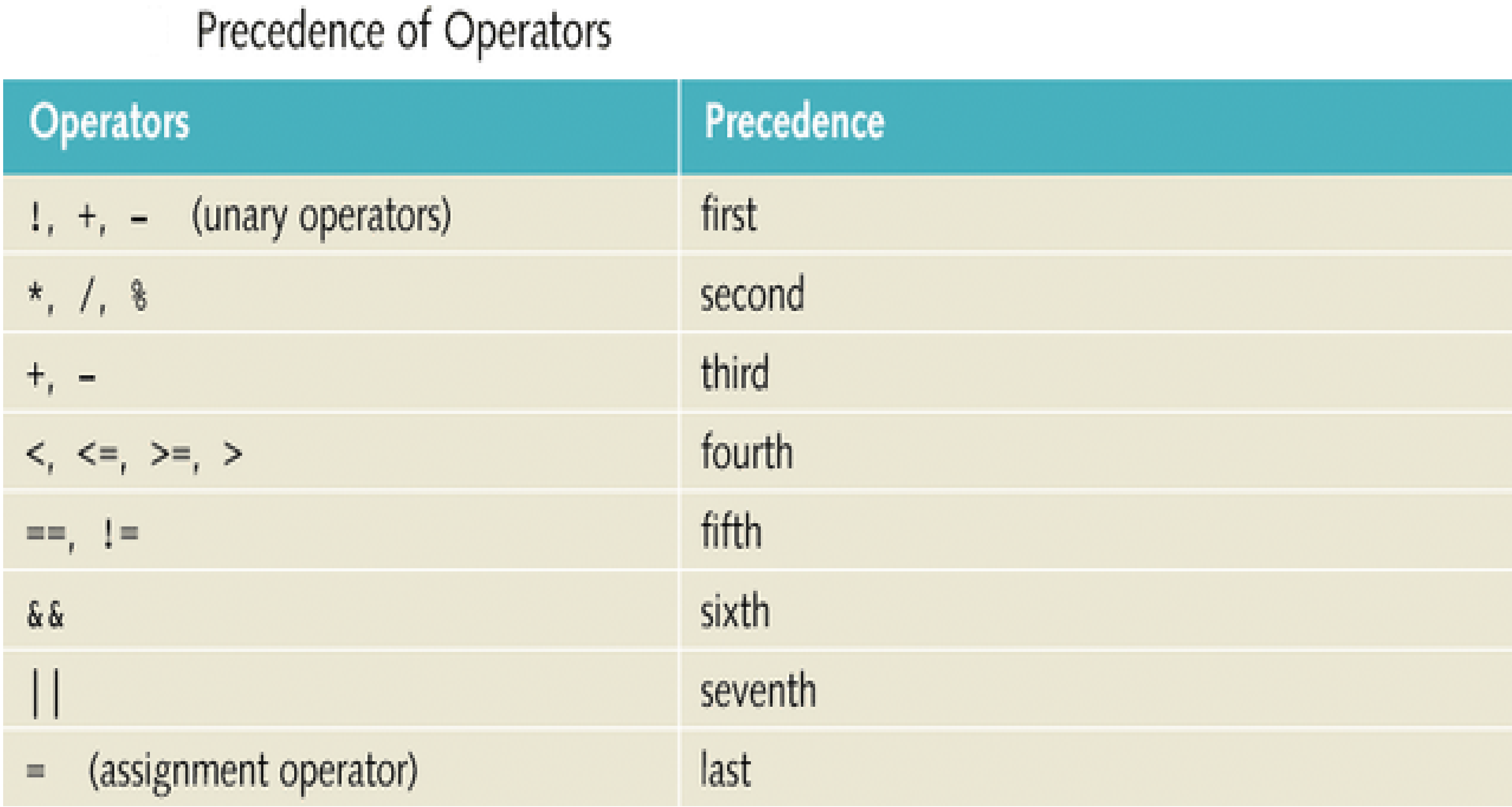
Order of Precedence

Consider the logical expression:

```
11 > 5 || 6 < 15 && 7 >= 8
```

- This logical expression will yield different results if `||` is evaluated first or && is evaluated first.
- If `||` is evaluated first, this logical expression evaluates to `0` (`false`).
- If && is evaluated first, this logical expression evaluates to `1`(`true`).

Precedence of Operators



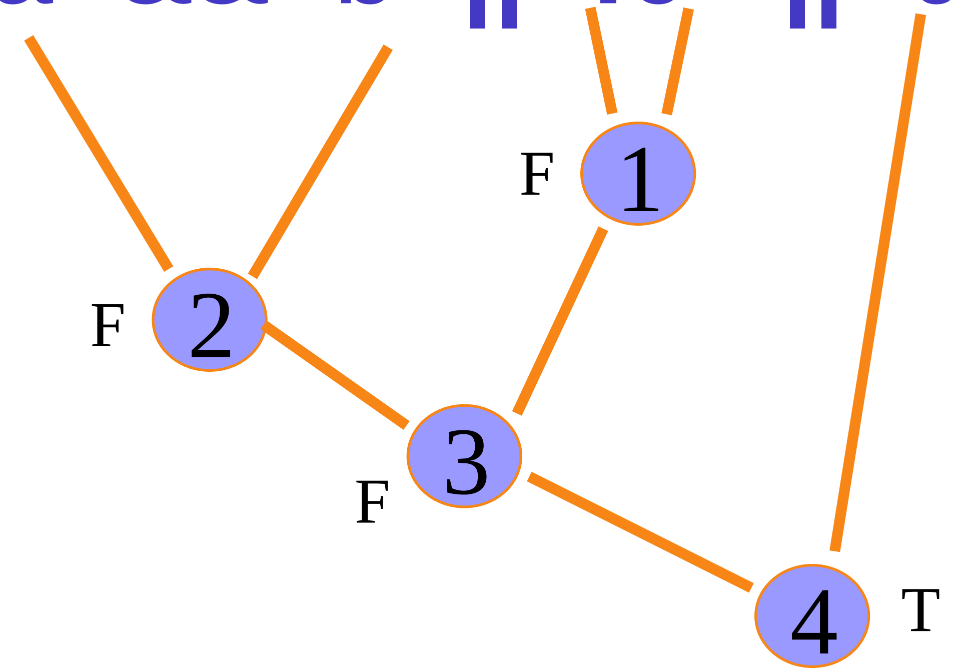
Precedence of Operators

Operators	Precedence
!, +, - (unary operators)	first
*, /, %	second
+, -	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

Logical Operators: Example

```
int a = 0, b = 3, c = 1, d =4;
```

```
a && b || !c || d
```



Logical Operators

Expression	Expression Equivalent
!(a == b)	a != b
!(a == b || a == c)	a != b && a != c
!(a == b && c > d)	a != b || c <= d

True or False

- Remember False evaluates to Zero
- True is any non zero (but when talking about relation expressions and logical expressions True evaluates specifically to one.
- (7 && 8==8) → (true && true) → 1

Example

```
void main()
{
int num = 1 ,   a = 5 , b = 8  , n = 20;
float x = 5.2, y = 3.4;

cout<<(x > 4.0)<<endl;

cout<<!num<<endl;

cout<<(x + y <= 20.5)<<endl;

cout<<((n >= 0) && (n <= 100))<<endl;

getch();

}
```

Output:

```
1
0
1
1
```

- You can insert parentheses into an expression to clarify its meaning.

The expression

```
11 > 5 || 6 < 15 && 7 >= 8
```

is equivalent to

```
11 > 5 || (6 < 15 && 7 >= 8)
```

This logical expression evaluates to `1` (`true`).

Example

Evaluate the following expression:

```
     (17 < 4*3+5) || (8*2 == 4*4) && !(3+3 == 6)

  =  (17 < 12+5) || (16 == 16) && !(6 == 6)
  =  (17 < 17) || true && !(true)
  =  false || true && false
  =  false || false
  =  false
```

When its printed on the screen as an output it displays

```
0
```

Example

Consider the following expressions:

1. `(5 >= 3) || ( x == 5)`
2. `(2 == 3) && (x >= 7)`

- In statement 1, because `(5 >= 3)` is `true` and the logical operator used in the expression is `||`, the expression evaluates to `true`. The computer does not evaluate `(x == 5)`.
- In statement 2, because `(2 == 3)` is `false` and the logical operator used in the expression is &&, the expression evaluates to `false`. The computer does not evaluate `(x >= 7)`.

Logical (Boolean) Assignments

The `int` Data Type and Logical (Boolean) Expressions

- Since logical expressions are evaluated to either `1` or `0`, the value of a logical expression can be stored in a variable of the type `int`. That is, logical (Boolean) expressions were manipulated with the help of `int` data type.

- Example:

```
int legalAge,age;
 Cin>>age;

legalAge = (age >= 21);
```

assigns the value `1` to `legalAge` if the value of `age` is greater than or equal to `21`. The statement assigns the value `0` if the value of `age` is less than `21`.

The Boolean Data type

A Boolean type is an integral type whose variable can have only two values : **false** and **true**

These value are stored as integers **0** and **1**

bool identifier; **OR** bool identifier= value;

bool identifier=false; **OR** bool identifier= true;

bool x;
x= !(5 != 2);
cout<<x;
	bool x=true;
	cout<<x;
		bool x=0;
		cout<<x;
			bool x=9;
			cout<<x;

0	1	0	1

Notes:

- The expression

```
0 <= num <= 10        Syntax error
```

- The correct way to write this expression in C++ is

```
0 <= num && num <= 10
```

Review Question

- Assume a=5, b=2, c=4, d=6, and e=3. Determine the value of each of the following expressions:
  - □ a > b
  - □ a != b
  - □ d % b == c % b
  - □ a * c != d * b
  - □ a % b * c

Common Programming Errors

- not declaring all variables
- storing data of one type in a variable of a different type.
- using a variable before assigning it a value
- in integer division 4/5 = 0