# 3.7 Logical operators

## Logical AND, OR, and NOT (general)

A ***logical operator*** treats operands as being true or false, and evaluates to true or false. Logical operators include AND, OR, and NOT. Programming languages typically use various symbols for those operators, but below the words AND, OR, and NOT are used for introductory purposes.

**PARTICIPATION ACTIVITY** 3.7.1: Logical operators: AND, OR, and NOT.

1 2 3 4 2x speed

| a | b | a AND b |
|---|---|---|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

| a | b | a OR b |
|---|---|---|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

| a | NOT a |
|---|---|
| false | true |
| true | false |

*Let x = 7, y = 9*

| | | |
|---|---|---|
| (x > 0) AND (y < 10) true / true — true | (x < 0) OR (y > 10) false / false — false | NOT (x < 0) false — true |
| (x > 0) AND (y < 5) true / false — false | (x < 0) OR (y > 5) false / true — true | NOT (x > 0) true — false |

Each operand is commonly an expression itself. If x = 7, y = 9, then (x > 0) AND (y < 10) is true AND true, so evaluates to true (both operands are true).

Feedback?

Table 3.7.1: Logical operators.

| Logical operator | Description |
|---|---|
| a AND b | ***Logical AND***: true when both of its operands are true |
| a OR b | ***Logical OR***: true when at least one of its two operands are true |
| NOT a | ***Logical NOT***: true when its one operand is false, and vice-versa. |

Feedback?

**PARTICIPATION ACTIVITY** | 3.7.2: Evaluating expressions with logical operators.

Indicate whether the expression evaluates to true or false.
x is 7, y is 9.

1) x > 5

[✓ true] false

**Correct**

Relational operators evaluate to true or false. Because 7 > 5, the expression evaluates to true.

2) (x > 5) AND (y < 20)

[✓ true] false

**Correct**

x > 5 is true. y < 20 is also true. If both operands of AND are true, AND evaluates to true.

3) (x > 10) AND (y < 20)

true [✓ false]

**Correct**

x > 10 is false. Both operands of AND must be true for AND to evaluate to true. Since one is false, AND evaluates to false.

4) (x > 10) OR (y < 20)

[✓ true] false

**Correct**

x > 10 is false, but y < 20 is true. If any operand of OR is true, OR evaluates to true.

5) (x > 10) OR (y > 20)

true [✓ false]

**Correct**

x > 10 is false. y > 20 is also false. Neither operand of OR is true, so OR evaluates to false.

6) NOT (x > 10)

[✓ true] false

**Correct**

x > 10 is false. NOT evaluates to the opposite of the operand. So NOT(false) is true.

7) NOT ( (x > 5) AND (y < 20) )

true [✓ false]

**Correct**

(x > 5) AND (y < 20) evaluates to true. NOT(true) evaluates to false.

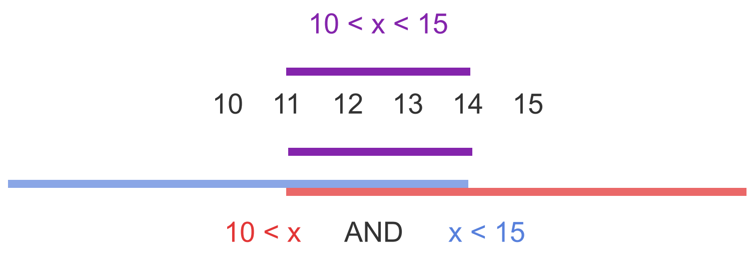
Feedback?

## Detecting ranges with logical operators (general)

A common use of logical operators is to detect if a value is within a range.

PARTICIPATION ACTIVITY | 3.7.3: Using AND to detect if a value is within a range.

1 2 3 ✓ 2x speed



x < 15 defines the range 14 and lower. ANDing yields the overlapping range. Only when x is 11, 12, 13, or 14 will both expressions be true.

Feedback?

PARTICIPATION ACTIVITY | 3.7.4: Using AND to detect if a value is within a range.

1) Which approach uses a logical operator to detect if x is in the range 1 to 99.

- 0 < x < 100
- [✓ (0 < x) AND (x < 100)]
- (0 < x) AND (x > 100)

**Correct**

The first expression checks if x is 1 or greater. The second checks if x is 99 or less. ANDing those checks if x is 1 or greater AND is 99 or less, so checks if x is 1 to 99.

2) Which detects if x is in the range -4 to +4?

- (x < -5) AND (x < 5)
- (x > -5) OR (x < 5)
- [✓ (x > -5) AND (x < 5)]

**Correct**

The first expression checks that x is -4 or greater. The second checks that x is 4 or less. ANDing yields -4 to 4.

3) Which detects if x is either less than -5, or greater than 10?

- ○ (x < -5) AND (x > 10)
- [✓ (x < -5) OR (x > 10)]

**Correct**

Notice that this question asks to detect if x is in one of two ranges, one range being less than -5 (to negative infinity), and the other greater than 10 (to positive infinity). Because two ranges are being checked for, OR is used.

Feedback?

## Logical operators

Special symbols are used to represent the AND, OR, and NOT logical operators.

Table 3.7.2: Logical operators.

| Logical operator | Description |
|---|---|
| a && b | ***Logical AND*** (&&): true when both of its operands are true |
| a \|\| b | ***Logical OR*** (\|\|): true when at least one of its two operands are true |
| !a | ***Logical NOT*** (!): true when its one operand is false, and vice-versa. |

Feedback?

**PARTICIPATION ACTIVITY** 3.7.5: Logical operators.

Match the symbol with the logical operator.

**&&**

AND

Two ampersand symbols mean logical AND. Note: A single ampersand, &, is a valid operator but is different than logical AND. Using a

**Correct**

single ampersand is a common beginner error

| | | |
|---|---|---|
| \|\| | **OR**<br>Two vertical pipe symbols mean logical OR. Note: A single vertical pipe, \|, is a valid operator but is different than logical OR. Using a single vertical pipe is a common beginner error. | Correct |
| ! | **NOT**<br>An exclamation point means NOT. The exclamation point comes before the operand as in !a, rather than after as in a! | Correct |
| !! | **No such operator**<br>Two exclamation points do not represent a valid operator (but instead represent two NOT operators in sequence, which cancel each other out). | Correct |

Reset

Feedback?

**PARTICIPATION ACTIVITY** 3.7.6: Evaluating expressions with logical operators. ✓

Given numPeople = 10, userKey = 'q'. Indicate whether the expression evaluates to true or false.

1)
```
(numPeople >= 10)
&& (userKey ==  'x')
```
○ true
● false

**Correct**
AND requires both operands to be true; though the first is true, the second is false. ✓

2)
```
(numPeople >= 20)
|| (userKey == 'q')
```
● true
○ false

**Correct**
OR only requires that at least one operand is true; here, the second operand is true. ✓

3)
```
!(userKey == 'a')
```
●

**Correct** ✓

[✓ true]

false

'q' == 'a' is false. Then !(false) yields true. Note that `(userKey != 'a')` would yield the same value.

4) `!( (numPeople == 5) || (numPeople == 6) )`

[✓ true]

false

**Correct**

The OR evaluates to false, since numPeople is neither 5 nor 6. Then, !(false) yields true.

Feedback?

Logical operators are commonly used in expressions found in if-else statements.

**PARTICIPATION ACTIVITY** 3.7.7: Logical operators: Complete the expressions to detect the desired range.

1) daysLogged is greater than 30 and less than 90

```
if ( (daysLogged > 30)
          (daysLogged <
90) ) {
   ...
}
```

**Answer**

&&

The expression evaluates to true if daysLogged is between 30 and 90, such as 41.

Check Show answer

2) 0 < maxCars < 100

```
if ( (maxCars > 0)
          (maxCars < 100)
) {
   ...
}
```

**Answer**

&&

Note that the mathematical inequality must be rewritten as two sub-expressions, each involving two operands.

Check Show answer

3) numStores is between 10 and 20, inclusive.

**Answer**

&&

Expressions that detect ranges using logical AND are common.

```
if ( (numStores >= 10)
            (numStores <=
20) ) {
   ...
}
```

Check Show answer

4) notValid is either less than 15, or greater than 79.

```
if ( (notValid < 15)
            (notValid > 79)
) {
   ...
}
```

Check Show answer

**Answer**

||

Expressions that detect two ranges use logical OR.

Feedback?

PARTICIPATION ACTIVITY 3.7.8: Creating expressions with logical operators.

1) numDogs is 3 or more and numCats is 3 or more.

```
if ( (numDogs >= 3)
                          ) {
   ...
}
```

Check Show answer

**Answers**

&& (numCats >= 3) or && (numCats > 2) or && numCats >= 3 or && numCats > 2

Such expressions require attention to details like parentheses and correct operator selection. The parentheses are optional but are good practice.

2) Either wage is greater than 10 or age is less than 18. Use ||. Use > and < (not >= and <=). Use parentheses around sub-expressions.

**Answer**

(wage > 10) || (age < 18)

If either sub-expression is true, the expression evaluates to true.

```
if (
                    )
{
   ...
}
```

Check Show answer

3) num is a 3-digit positive integer. Ex: 100, 989, and 523, are 3-digit positive integers, but 55, 1000, and -4 are not.

For most direct readability, your expression should compare directly with the smallest and largest 3-digit number.

```
if ( (num >= 100)
                          ) {
   ...
}
```

Check Show answer

**Answers**

&& (num <= 999) or

&& (num < 1000) or

&& num <= 999 or

&& num < 1000

Comparing with 100 and 999 is preferred because they themselves are the limits of 3-digit numbers.

Feedback?

PARTICIPATION ACTIVITY 3.7.9: Logical expression simulator.

Try typing different expressions involving x, y and observe whether the expression evaluates to true.

```
int x = 7 ;
int y = 5 ;
if (                    ) {
   ...
}
```

Run code

**Output is:**

```
Awaiting your input...
```

Feedback?

Feedback?

## Example: TV channels

A cable TV provider may have regular channels numbered 2-499, and high-definition channels numbered 1002-1499. A program may set a character variable to 's' for standard, 'h' for high-definition, and 'e' for error.

Figure 3.7.1: Detecting ranges: Cable TV channels.

```
if ( (userChannel >= 2) && (userChannel <= 499) ) {
   channelType = 's';
}
else if ((userChannel >= 1002) && (userChannel <= 1499) ) {
   channelType = 'h';
}
else {
   channelType = 'e';
}
```

Feedback?

zyDE 3.7.1: Detecting ranges: Cable TV channels.

Run the program and observe the output. Change the input box value from 3 to ar number, and run again.

Load default template...

```
#include <iostream>
using namespace std;

int main() {
   int userChannel;
   char channelType;

   cin  >> userChannel;

   if ( (userChannel >= 2) && (userChannel
      channelType = 's';
   }
   else if ((userChannel >= 1002) && (user(
      channelType = 'h';
   }
   else {
      channelType = 'e';
   }

```

```
3
```

Run

Feedback?

PARTICIPATION ACTIVITY

3.7.10: TV channel example: Detecting ranges.

Consider the above example.

1) If userChannel is 300, to what does the if statement's expression, `(userChannel >= 2) && (userChannel <= 499)`, evaluate?

- true
- false

2) If userChannel is 300, does the else if's expression `(userChannel >= 1002) && (userChannel <= 1499)` get checked?

- Yes
- No

3) Did the expressions use logical AND or logical OR?

- AND
- OR

4) Channels 500-599 are pay channels. Does this expression detect that range? `(userChannel >= 500) || (userChannel <= 599)`

- Yes
- No

Feedback?

## Detecting ranges implicitly vs. explicitly

If a program should detect increasing ranges without gaps, a multi-branch if-else statement can be used without logical operators; the low-end of the range is implicitly known upon reaching an expression. Likewise, a decreasing range without gaps has implicitly-known high-ends. In contrast, when gaps exist, the range's low and high ends must both be explicitly detected, using a logical operator.

**PARTICIPATION ACTIVITY** 3.7.11: Detecting ranges implicitly vs. explicitly.

1 2 3 2x speed

```
if (x < 0) {
  // Negative
}
else if ( (x >= 0) && (x <= 10) ) {
  // 0..10
}
else if ( (x >= 11) && (x <= 20) {
  // 11..20
}
else {
  // 21+
}
```

```
if (x < 0) {
  // Negative
}
else if (x <= 10) {
  // 0..10
}
else if (x <= 20) {
  // 11..20
}
else {
  // 21+
}
```

```
if (x < 0) {
  // Negative
}
else if ( (x >= 16) && (x <= 25)
) {
  // 16..25
}
else if ( (x >= 50) && (x <= 62)
) {
  // 50..62
}
else {
  // 0..15, 26..49, 63+
```

If the range has gaps, the range's ends must be explicitly indicated, using AND. So if the if expressi
0, and then next range is 16..25, an explicit range is needed.

Feedback?

**PARTICIPATION ACTIVITY** 3.7.12: Detecting ranges implicitly vs. explicitly.

Using a multi-branch if-else statement, indicate whether one end of each range can be detected implicitly, or whether both ends must be detected explicitly.

1) <= 10
11..20
21..50
51+

[✓ Implicit]
Explicit

**Correct**

No gaps exist in this increasing series of ranges, so the low-end of each range is implicit.

```
if (x <= 10)...
else if (x <= 20)... // x must be 11 or more
else if (x <= 50)... // x must be 21 or more
else... // x must be 51 or more
```

Detecting each range is NOT wrong, but just unnecessary.

2) <=10
50..100
150..200

○ Implicit
● Explicit

**Correct**

While the ranges are increasing, gaps exist. Ex: 11..49 is not present. So the ranges must be detected explicitly.

```
if (x <= 10)...
else if ( (x >= 50) && (x <= 100) )...
else if ( (x >= 150) && (x <= 200) )...
```

3) >= 100
90..99
80..89

● Implicit
○ Explicit

**Correct**

No gaps exist in this decreasing series of ranges, so the upper-end of each range is implicit.

```
if (x >= 100)...
else if (x >= 90)...
else if (x >= 80)...
```

Feedback?

**CHALLENGE ACTIVITY** 3.7.1: Detect specific values.

Write an expression that prints "Special number" if specialNum is 0, -99, or 44.

```
#include <iostream>
using namespace std;

int main() {
   int specialNum;

   cin >> specialNum;

   if (specialNum==0 || specialNum ==-99 || specialNum == 44/* Your solution goes here  */)
      cout << "Special number" << endl;
   }
   else {
      cout << "Not special number" << endl;
   }

   return 0;
}
```

Run ✓ All tests passed

✓ Testing with specialNum = 17

Your output
```
Not special number
```

✓ Testing with specialNum = -99

Your output
```
Special number
```

✓ Testing with specialNum = 0

Your output
```
Special number
```

✓ Testing with specialNum = 44

Your output
```
Special number
```

✓ Testing with specialNum = -412

Your output
```
Not special number
```

Feedback?

CHALLENGE ACTIVITY

3.7.2: Detect number range.

Write an expression that prints "Eligible" if userAge is between 18 and 25 inclusive.
Ex: 17 prints "Ineligible", 18 prints "Eligible".

```
#include <iostream>
using namespace std;

int main() {
   int userAge;

   cin >> userAge;

   if (userAge>=18 && userAge<=25/* Your solution goes here  */) {
      cout << "Eligible" << endl;
   }
   else {
      cout << "Ineligible" << endl;
   }

   return 0;
}
```

Run

✓ All tests passed

✓ Testing with userAge = 17

Your output
```
Ineligible
```

✓ Testing with userAge = 18

Your output
```
Eligible
```

✓ Testing with userAge = 19

Your output
```
Eligible
```

✓ Testing with userAge = 25

Your output
```
Eligible
```

✓ Testing with userAge = 26

Your output
```
Ineligible
```

Feedback?