# Operators

Chris Wild

May 25, 2013

## Contents

# 1 Description

- Relational, equality and logical operators are used to form logic expressions.
- Logic expressions are those expressions whose value is either `true` or `false`
  - (as opposed to arithmetic expressions whose value is a number)
- Relational and equality operators usually compare two numbers and return a value of `true` or `false` (forming a logic expression).
- Logical operators combine logical values of `true` or `false` into a logical expression.
- Relational operators and logical operators are often used together in logic expressions.
  - Logical operators often combine logical expressions formed from relational operators.
- Because of this, it is important to understand the precedence relationship between relational and logical operators
  - Precedence is used by the compiler to decide in order in which the expression is evaluated.
- Logical expressions are important in the conditional expression in ***if, while, for, and do-while*** statements

# 2 Examples

## 2.1 Relational and Equality Operators

The *relational operators* are `<, <=, >, >=`

The *Equality operators* are `= =, !=`

Given two numbers, `a` and `b`, the following table lists examples of *logic expressions* formed using the relational and equality operators.



| Example | Meaning |
|---|---|
| a = = b | a is equal to b |
| a < b | a is less than b |
| a > b | a is greater than b |
| a <= b | a is less than or equal to b |
| a >= b | a is greater than or equal to b |
| a != b | a is not equal to b |

## 2.2 Logical Operators

The *logical operators* include && and | |

Given numbers **a, b, c,** and **d**. The following table lists examples combining relational and logical operators.

| Example | Meaning |
|---|---|
| (a = = b) && (c < d) | true if a is equal to b AND c is less than d |
| a = = b && c < d | same as the above (see precedence rules) |
| (a > b) \| \| ( c != d) | true if a is greater than b OR if c is not equal to d |
| a > b \| \| c != d | same as the above (see precedence rules) |

# 3 Precedence Rules

**Motivation:** Consider the following expression:

```
a > b && b == c || c < d
```

There are two interpretations which give two different results

```
(a > b && b > c) || c < d //do first two relational operators first, then the last, value is true
a > b && (b >c || c < d)  // do last two relational operators first, then the first, value is false
```

Precedence rules let the compiler decide which interpretation to take. The precedence rules are:

- do relational operators first,
- then equality,



- then the logical AND (&&)
- then the logical OR (||).

By these rules, the first interpretation is used.

*However*, if in doubt, use parenthesis to make your meaning clear to yourself and other programmers.

# 4 Tips

- *Do not confuse the logical operator "==" ( double equal signs) with the assignment operator "=" (single equals sign).* It can lead to logical errors that the compiler will not catch. (click for experiment about this)
- Use pararenthesis even if not necessary to make your logical expressions easier to read.
- When trying to decide whether you need a simple comparision ('<' and '>') or a compound one ( "<=" and ">=") mentally run the program for a simple case (e.g. like executing the loop 0 or 1 time). *Off by one* errors are common in logical expressions. *Always mentally check the boundary conditions.*

# 5 Experiment

**Difference between the assignment and equality**

```
#include <iostream>

using namespace std;

int main()
{
    int i = 4;
    int j = 5;

    if ( i = j) {
        cout << "equal" << endl;
```



```
    } else {
        cout << "NOT equal" << endl;
    }
    return 0;
}
```

**Hypothesis:** What will this program print out?

- equal?
- NOT equal?

If you aren't sure, compile and run the program to see.



OK, its a trick question - but why?



- The = operator is used where the == was intended.
- Instead of comparing `i` and `j`, `j` is assigned to `i`.
- An assignment operator "returns" or "evaluates to" whatever value was actually copied. In this case, the condition evaluates to 5.
  - By ancient C++ convention, any non-zero integer value is considered to be "true", while zero is considered to be "false".

What is the value of "i" after the "if" statement executes? (you can modify the experiment to output the value)



*5*