Chapter 03: Controlling Program Flow

Decisions constructs; Loop constructs; break and continue

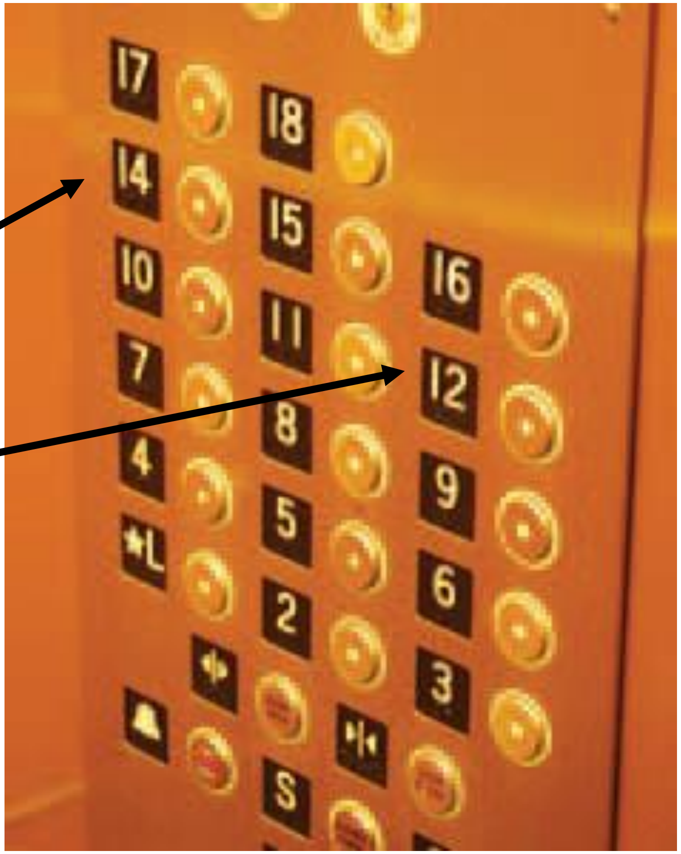
ILO: Describe various program building blocks

Acknowledged: C++ for Everyone *by C Horstmann by John Wiley & Sons.*

The if Statement

The thirteenth floor!

It's missing!



***OH NO*!!!**

The if Statement

We must write the code to control the elevator.

How can we skip the 13th floor?

The if Statement

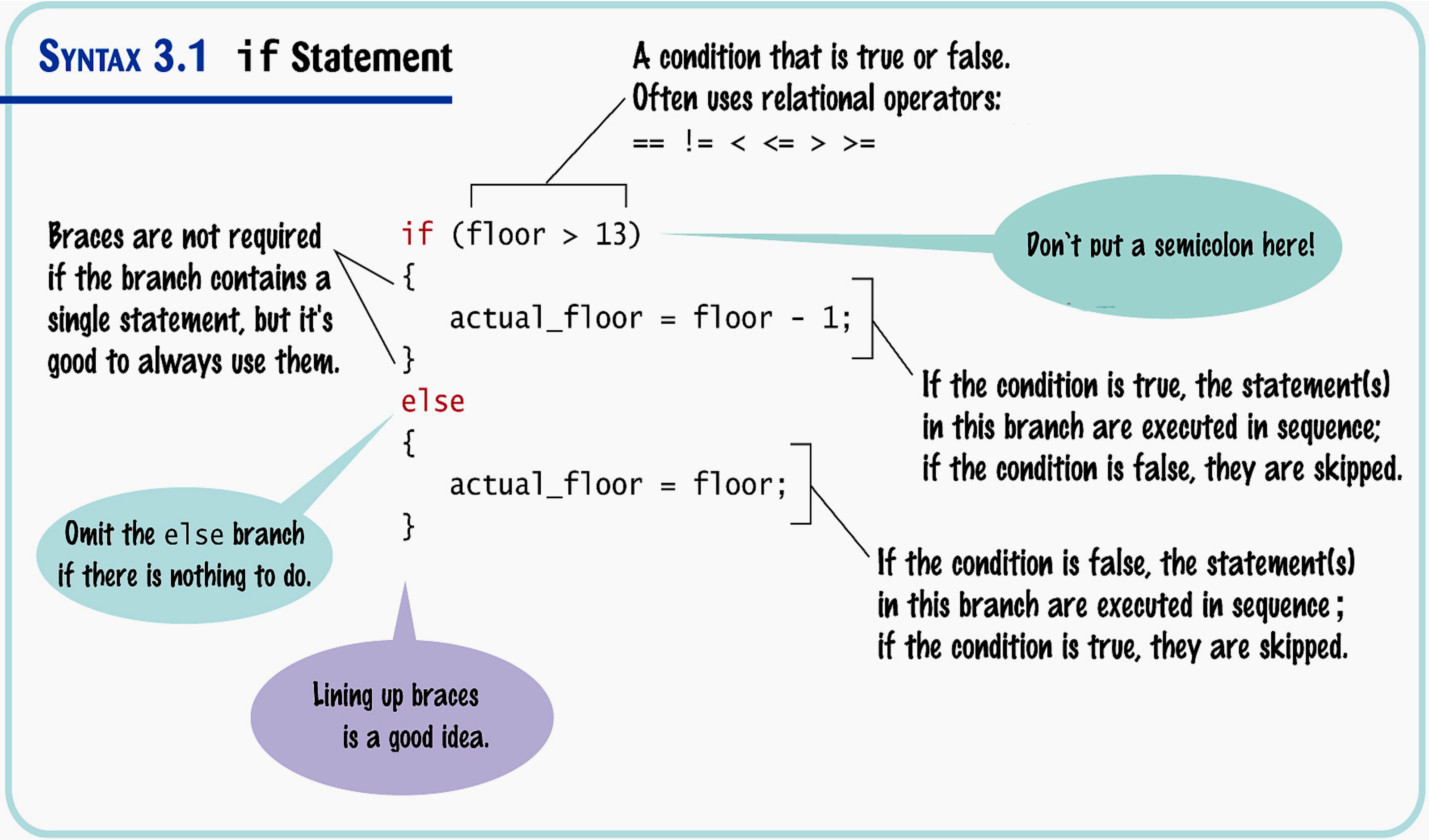
We will model a person choosing a floor by getting input from the user:

```
int floor;
cout << "Floor: ";
cin >> floor;
```

The if Statement

```
int actual_floor;
if (floor > 13)
{
   actual_floor = floor - 1;
}
else
{
   actual_floor = floor;
}
```

The if Statement



SYNTAX 3.1 if Statement

```
if (floor > 13)
{
   actual_floor = floor - 1;
}
else
{
   actual_floor = floor;
}
```

A condition that is true or false. Often uses relational operators: == != < <= > >=

Don't put a semicolon here!

Braces are not required if the branch contains a single statement, but it's good to always use them.

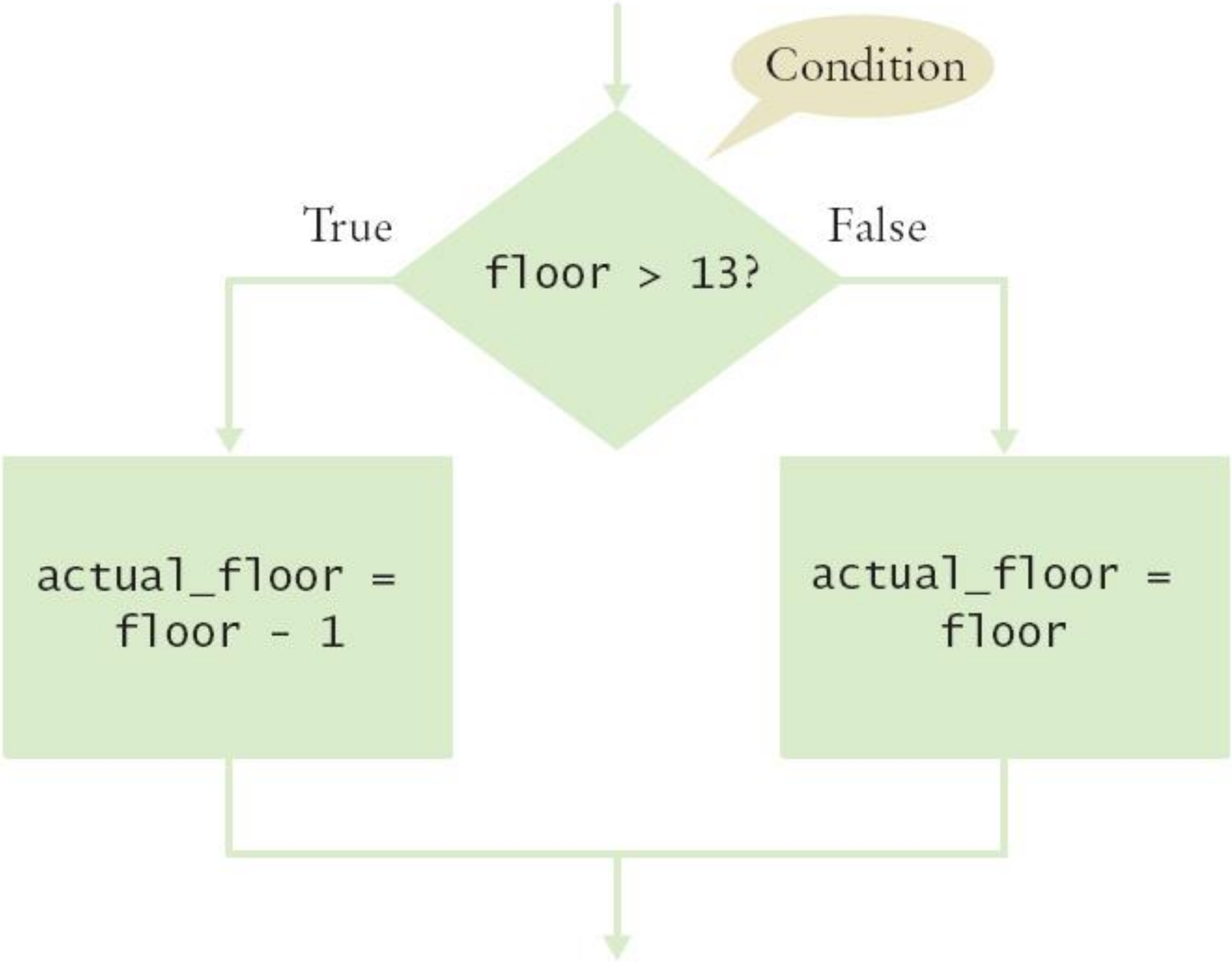
If the condition is true, the statement(s) in this branch are executed in sequence; if the condition is false, they are skipped.

Omit the else branch if there is nothing to do.

If the condition is false, the statement(s) in this branch are executed in sequence ; if the condition is true, they are skipped.

Lining up braces is a good idea.

The if Statement – The Flowchart



The if Statement

Here is another way to write this code:

We only need to decrement when the floor is greater than 13.

We can set *actual_floor* ***before testing:***

```
int actual_floor = floor;
if (floor > 13)
{
   actual_floor--;
} // No else needed
```

(And you’ll notice we used the decrement operator this time.)

The if Statement – A Complete Elevator Program

ch03/elevator1.cpp

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

int main()
{
   int floor;
   cout << "Floor: ";
   cin >> floor;
   int actual_floor;
   if (floor > 13)
   {
      actual_floor = floor - 1;
   }
   else
   {
      actual_floor = floor;
   }

   cout << "The elevator will travel to the actual floor "
      << actual_floor << endl;

   return 0;
}
```

The if Statement – Common Error – The Do-nothing Statement

```
if (floor > 13) ; // ERROR ?
{
    floor--;
}
```

This is **not** ***a compiler error.***
The compiler does not complain.
It interprets this `if` ***statement as follows:***

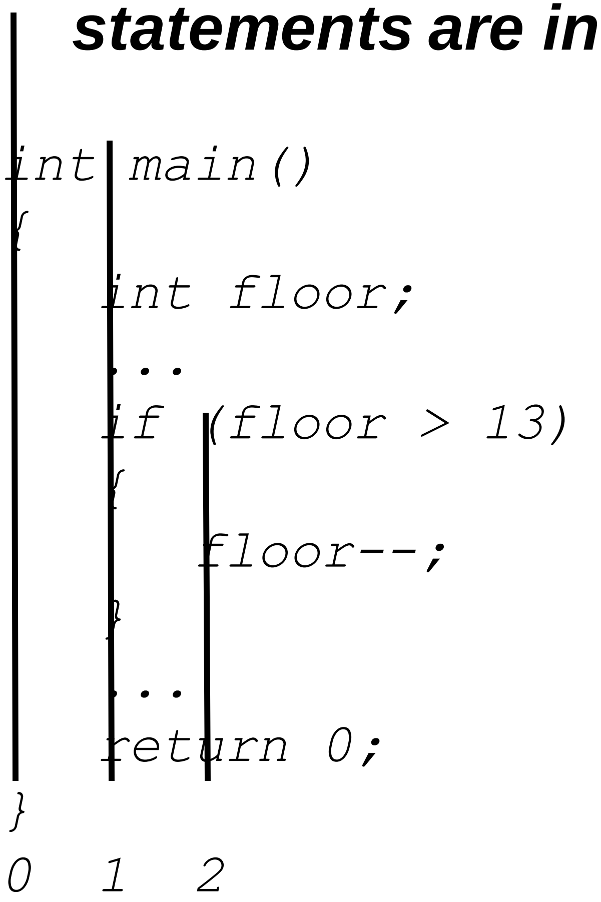
If floor is greater than 13, execute the **do-nothing statement***.*
(semicolon by itself is the do nothing statement)

Then **after that** ***execute the code enclosed in the braces.***

Any statements enclosed in the braces are no longer a part of the `if` ***statement.***

The if Statement – Indent when Nesting

Block-structured code has the property that **nested** ***statements are indented by one or more levels.***



```
int main()
{
   int floor;
   ...
   if (floor > 13)
   {
      floor--;
   }
   ...
   return 0;
}
```

Indentation level

The Conditional Operator

C++ has the conditional operator of the form

```
condition ? value1 : value2
```

The value of that expression is either `value1` ***if the test passes or*** `value2` ***if it fails.***

The Conditional Operator

For example, we can compute the actual floor number as

```
actual_floor = floor > 13 ? floor - 1 : floor;
```

which is equivalent to

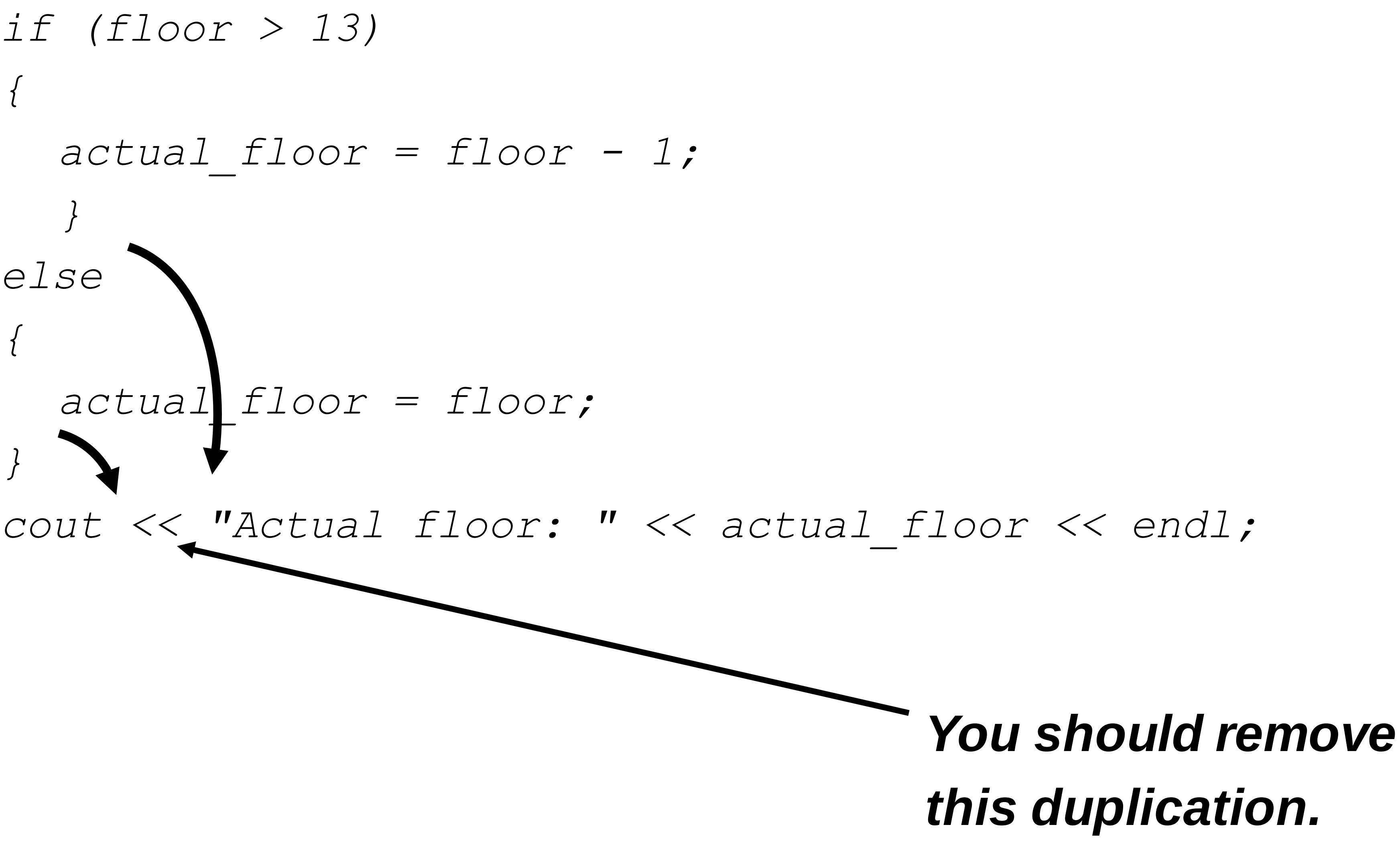
```
if (floor > 13)
{
   actual_floor = floor - 1;
}
else
{
   actual_floor = floor;
}
```

The if Statement – Removing Duplication

```
if (floor > 13)
{
   actual_floor = floor - 1;
   cout << "Actual floor: " << actual_floor << endl;
}
else
{
   actual_floor = floor;
   cout << "Actual floor: " << actual_floor << endl;
}
```

Hmmm…

The if Statement – Removing Duplication



```
if (floor > 13)
{
   actual_floor = floor - 1;
   }
else
{
   actual_floor = floor;
}
cout << "Actual floor: " << actual_floor << endl;
```

You should remove this duplication.

Relational Operators

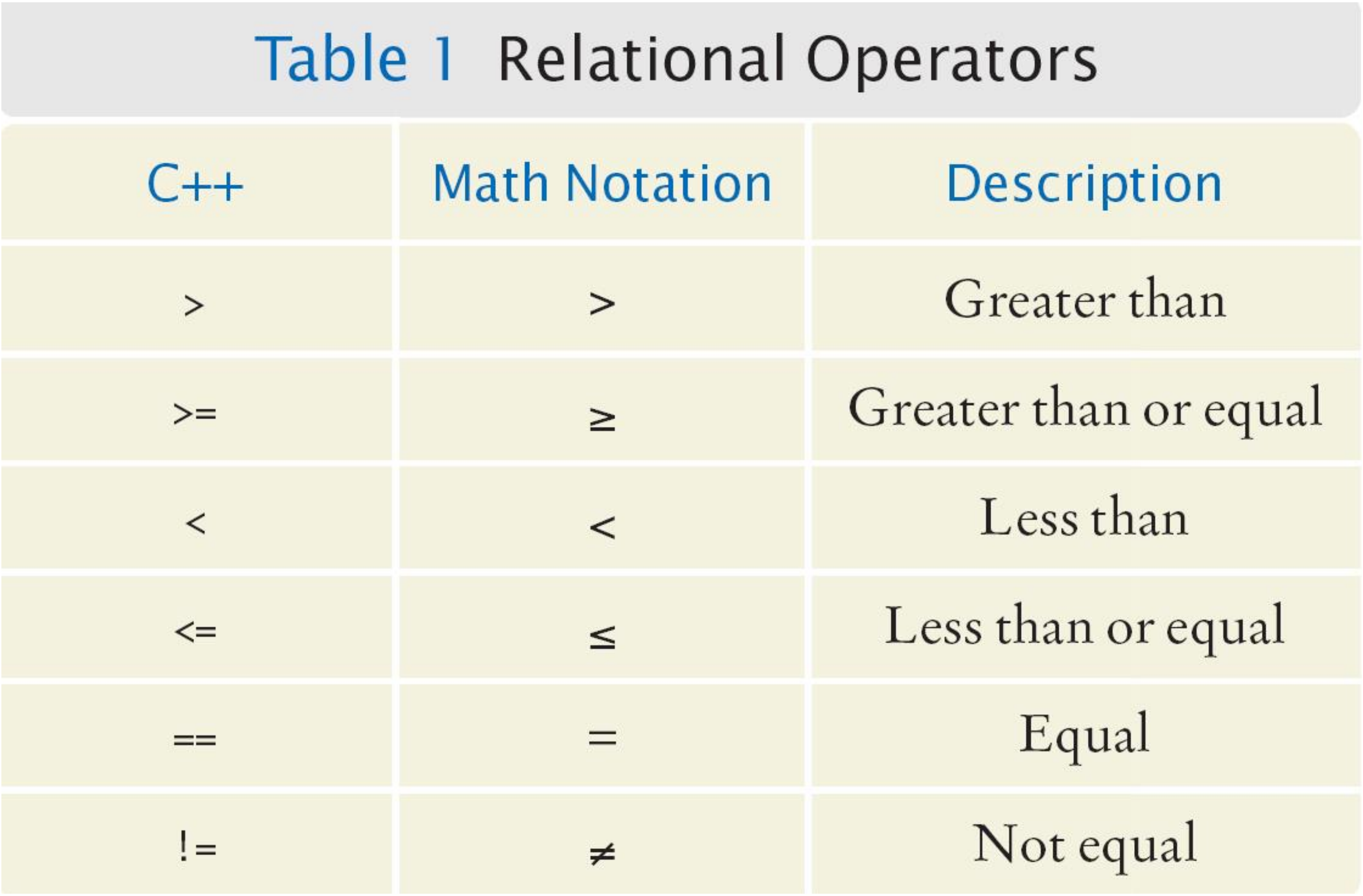


Table 1 Relational Operators

C++	Math Notation	Description
>	>	Greater than
>=	≥	Greater than or equal
<	<	Less than
<=	≤	Less than or equal
==	=	Equal
!=	≠	Not equal

Relational Operators

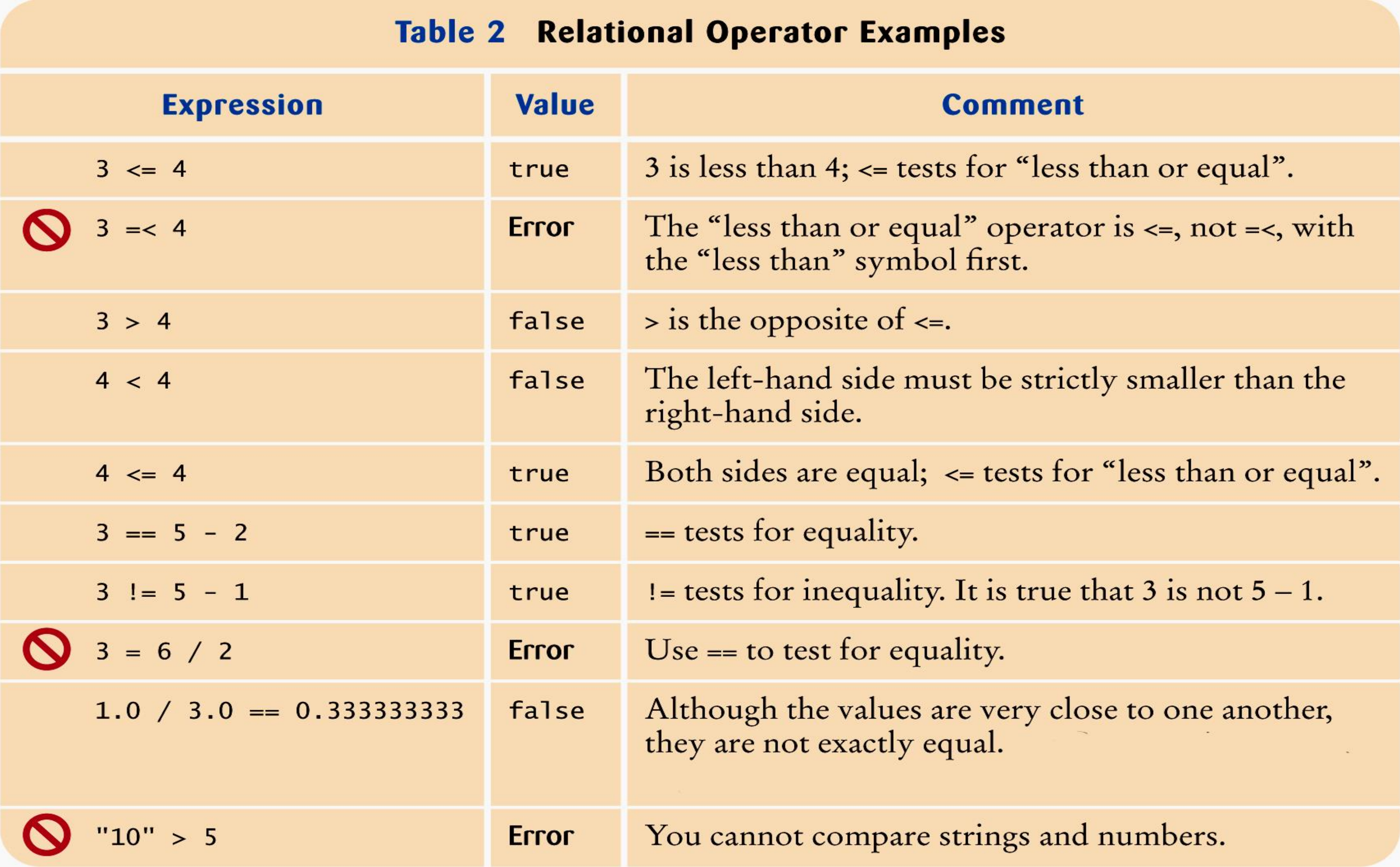


Table 2 Relational Operator Examples

Expression	Value	Comment
`3 <= 4`	`true`	3 is less than 4; <= tests for "less than or equal".
🚫 `3 =< 4`	Error	The "less than or equal" operator is <=, not =<, with the "less than" symbol first.
`3 > 4`	`false`	> is the opposite of <=.
`4 < 4`	`false`	The left-hand side must be strictly smaller than the right-hand side.
`4 <= 4`	`true`	Both sides are equal; <= tests for "less than or equal".
`3 == 5 - 2`	`true`	== tests for equality.
`3 != 5 - 1`	`true`	!= tests for inequality. It is true that 3 is not 5 – 1.
🚫 `3 = 6 / 2`	Error	Use == to test for equality.
`1.0 / 3.0 == 0.333333333`	`false`	Although the values are very close to one another, they are not exactly equal.
🚫 `"10" > 5`	Error	You cannot compare strings and numbers.

Multiple Alternatives

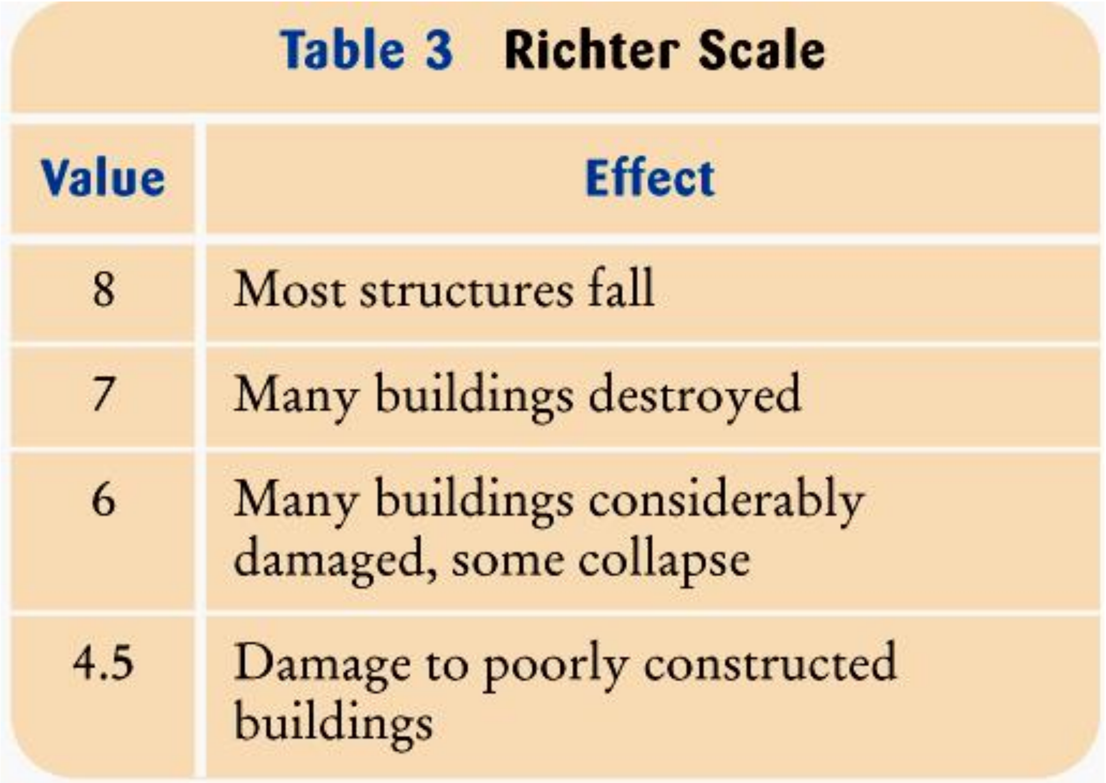
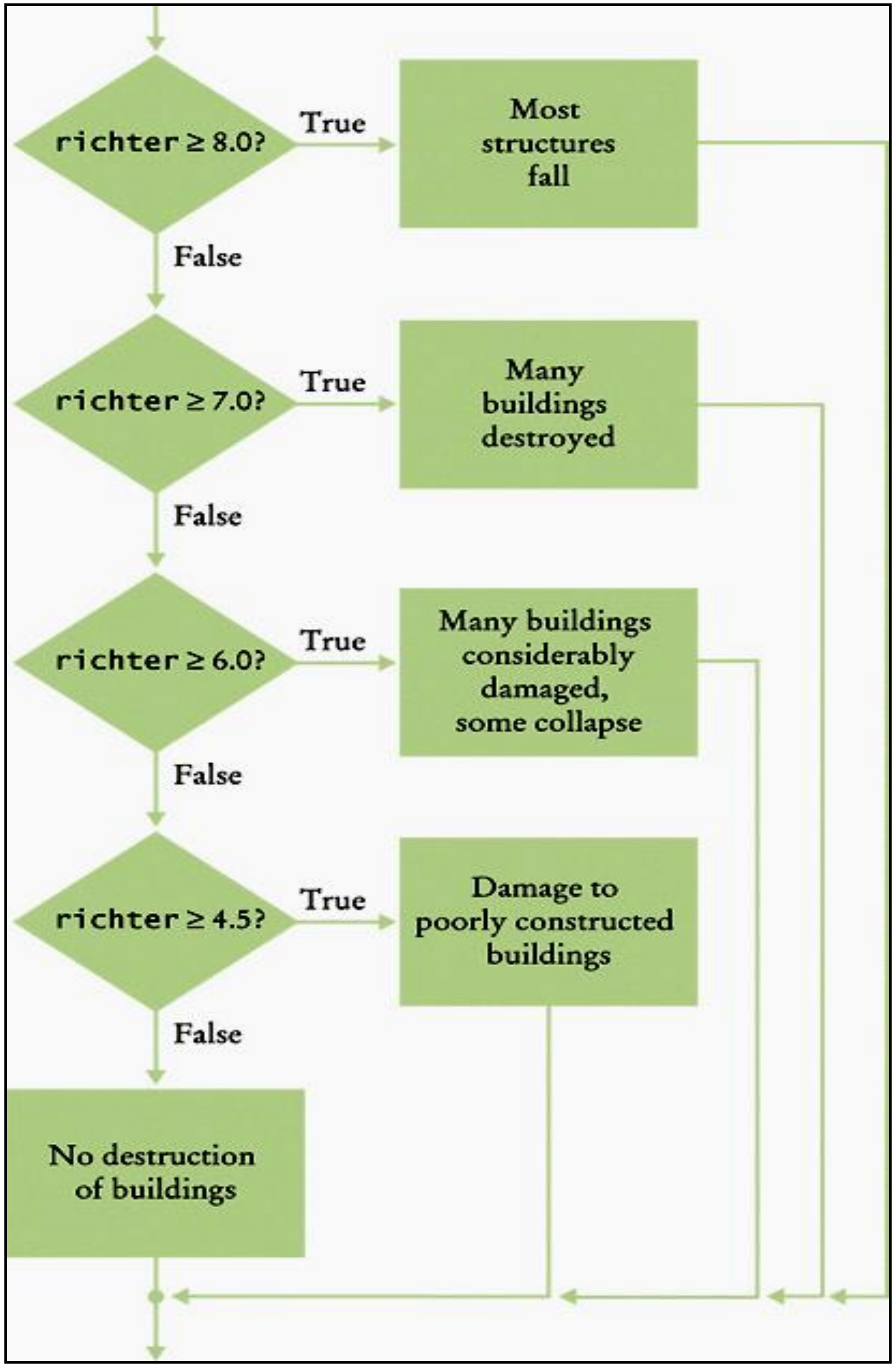


Table 3 Richter Scale

Value	Effect
8	Most structures fall
7	Many buildings destroyed
6	Many buildings considerably damaged, some collapse
4.5	Damage to poorly constructed buildings

Richter flowchart



Multiple Alternatives

```
    if (richter >= 8.0)
    {
       cout << "Most structures fall";
    }
    else if (richter >= 7.0)
    {
       cout << "Many buildings destroyed";
    }
    else if (richter >= 6.0)
    {
       cout << "Many buildings considerably damaged, some collapse";
    }
    else if (richter >= 4.5)
    {
       cout << "Damage to poorly constructed buildings";
    }
    else
    {
       cout << "No destruction of buildings";
    }
    . . .
```

The `switch` Statement

This is a bit of a mess to read.

```
int digit;
...
if (digit == 1) { digit_name = "one"; }
else if (digit == 2) { digit_name = "two"; }
else if (digit == 3) { digit_name = "three"; }
else if (digit == 4) { digit_name = "four"; }
else if (digit == 5) { digit_name = "five"; }
else if (digit == 6) { digit_name = "six"; }
else if (digit == 7) { digit_name = "seven"; }
else if (digit == 8) { digit_name = "eight"; }
else if (digit == 9) { digit_name = "nine"; }
else { digit_name = ""; }
```

The `switch` Statement

```
int digit;
.
switch (digit)
{
   case 1: digit_name = "one"; break;
   case 2: digit_name = "two"; break;
   case 3: digit_name = "three"; break;
   case 4: digit_name = "four"; break;
   case 5: digit_name = "five"; break;
   case 6: digit_name = "six"; break;
   case 7: digit_name = "seven"; break;
   case 8: digit_name = "eight"; break;
   case 9: digit_name = "nine"; break;
   default: digit_name = ""; break;
}
```

Nested Branches

It is possible to have multiple case clauses for a branch:

```
case 1: case 3: case 5: case 7: case 9: odd = true; break;
```

The `default:` ***branch is chosen if none of the case clauses match.***

Boolean Variables and Operators

Two values, eh?

like true and false

like on and off

– like electricity!

In essence he invented the computer!

Boolean Variables

Here is a definition of a Boolean variable, initialized to `false`**:**

```
bool failed = false;
```

It can be set by an intervening statement so that you can use the value **later** ***in your program to make a decision:***

```
// Only executed if failed has
// been set to true
if (failed)
{
   ...
}
```

The Boolean Operator `&&` (and)

In C++, the `&&` ***operator (called*** **and*****) yields*** `true` ***only when*** **both** ***conditions are*** `true`**.**

```
if (temp > 0 && temp < 100)
{
    cout << "Liquid";
}
```

If `temp` ***is within the range, then both the left-hand side*** **and** ***the right-hand side are*** `true`**,** ***making the whole expression’s value*** `true`**.**

In all other cases, the whole expression’s value is `false`**.**

The Boolean Operator `||` (or)

The `||` ***operator (called*** **or*****) yields the result*** `true` ***if at least one of the conditions is*** `true`**.**

- ***This is written as two adjacent vertical bar symbols.***

```
if (temp <= 0 || temp >= 100)
{
   cout << "Not liquid";
}
```

If **either** ***of the expressions is*** `true`**,**
the whole expression is `true`**.**

The only way "Not liquid" won't appear is if **both** ***of the expressions are*** `false`**.**

The Boolean Operator `!` (not)

Sometimes you need to invert a condition with the logical* not *operator.

The `!` operator takes a single condition and evaluates to `true` if that condition is `false` and to `false` if the condition is `true`.

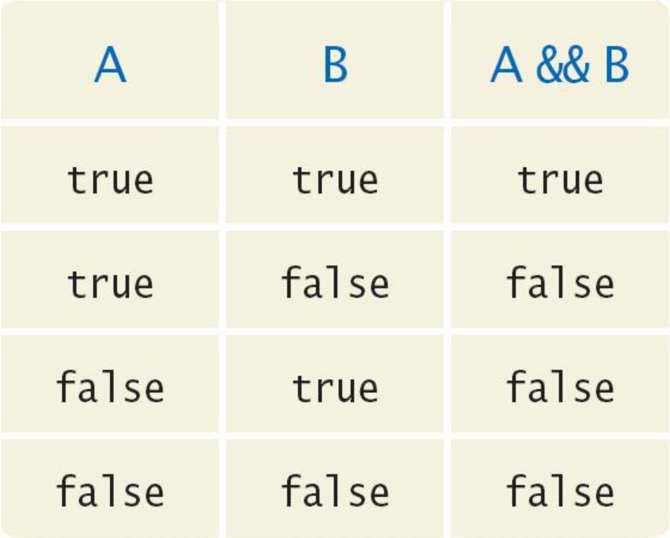
```
if (!frozen) { cout << "Not frozen"; }
```

"Not frozen" will be written only when frozen contains the value `false`.

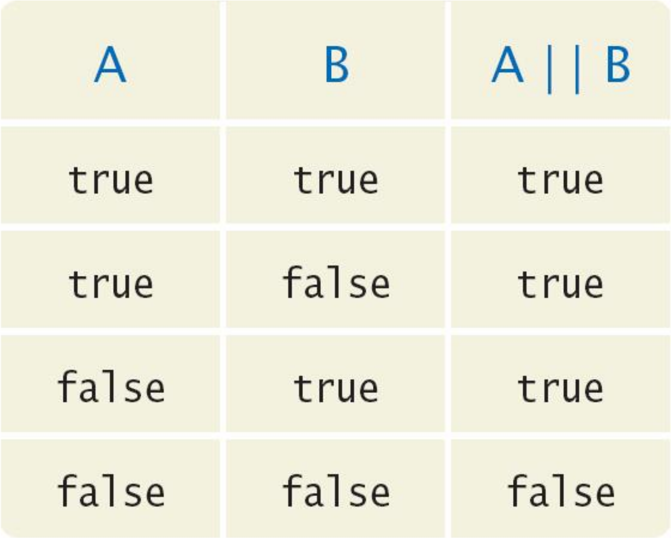
`!false` **is** `true`**.**

Boolean Operators

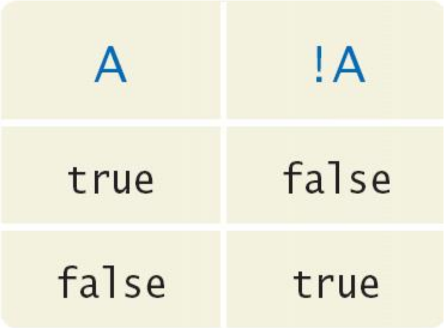
This information is traditionally collected into a table called a* truth table*:



A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false



A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false



A	!A
true	false
false	true

where A and B denote `bool` ***variables or Boolean expressions.***

Common Error – Combining Multiple Relational Operators

Consider the expression

```
if (0 <= temp <= 100)…
```

This looks just like the mathematical test:

$$0 \leq temp \leq 100$$

Unfortunately, it is not.

Input Validation with `if` Statements

Let's return to the elevator program and consider input validation.

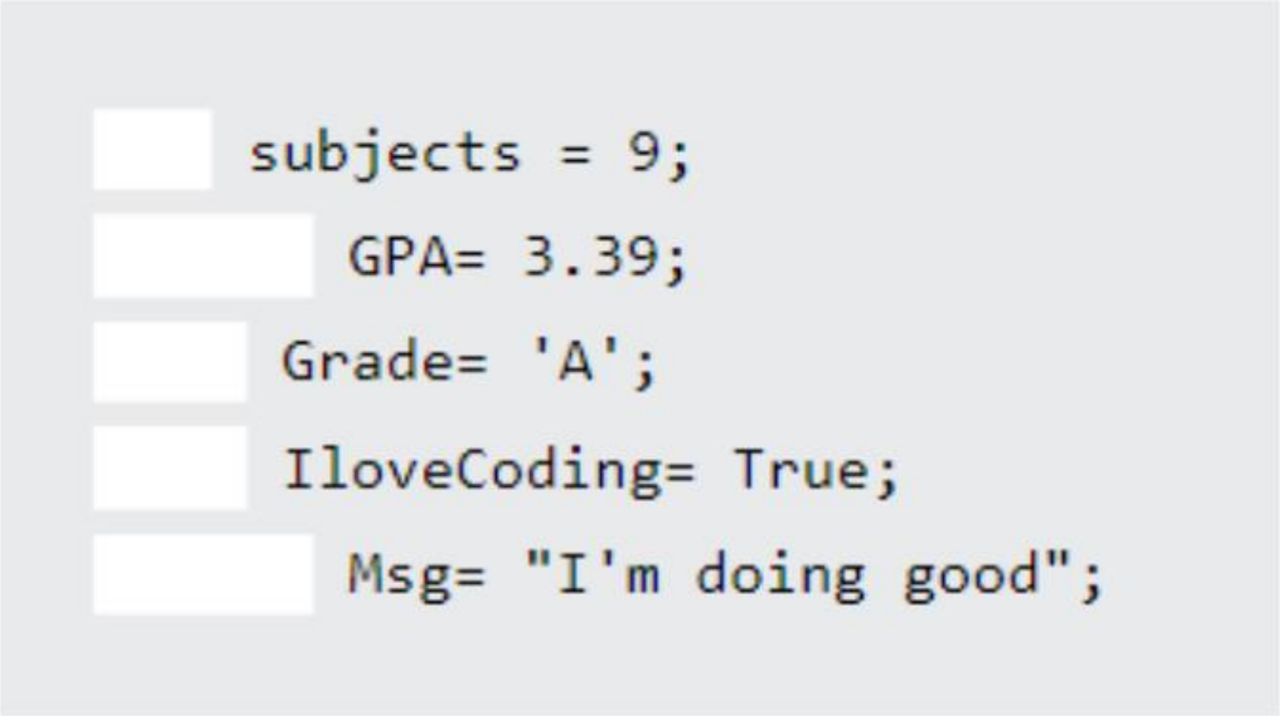
Input Validation with `if` Statements – Elevator Program

ch03/elevator2.cpp

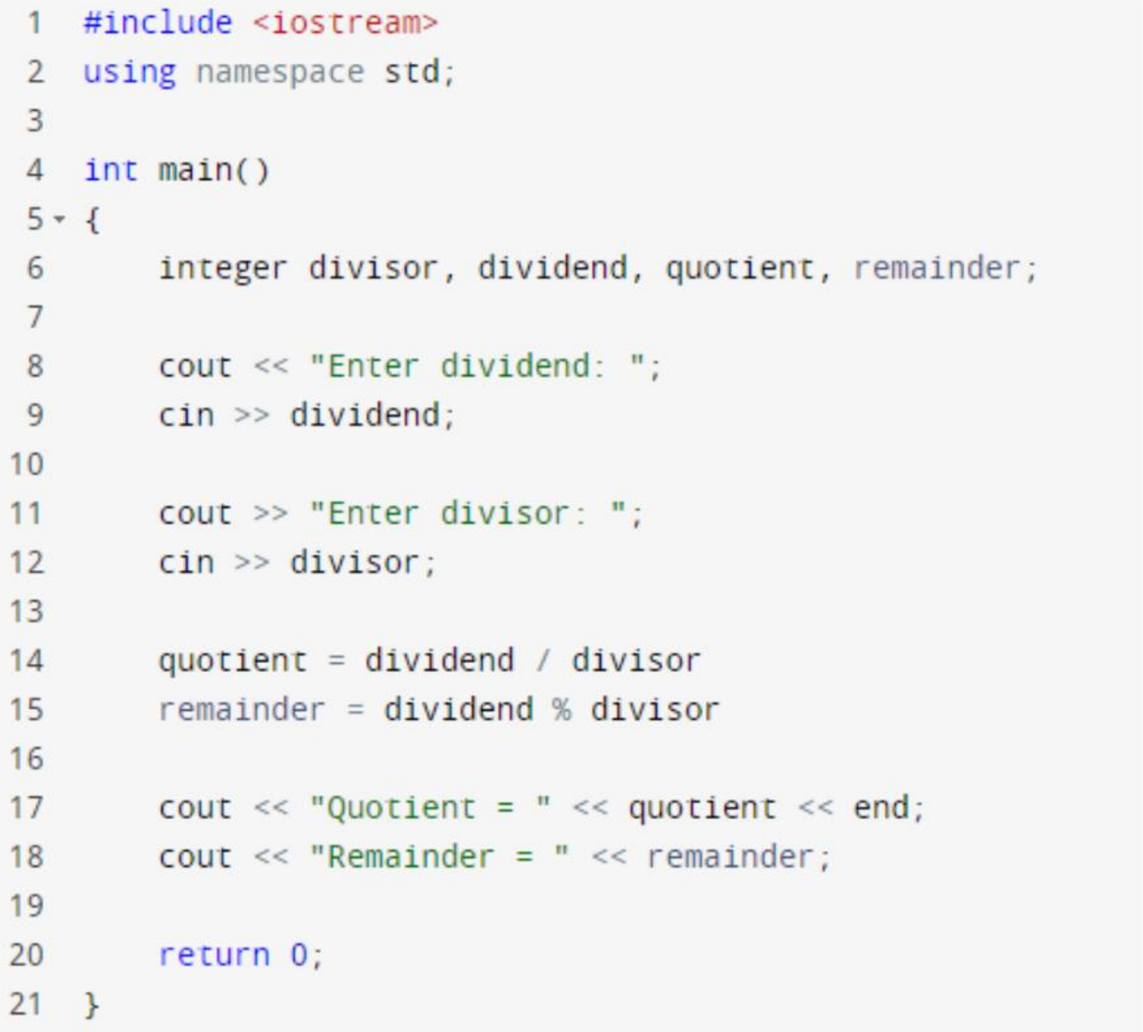
```
#include <iostream>
using namespace std;

int main()
{
   int floor;
   cout << "Floor: ";
   cin >> floor;

   // The following statements check various input errors
   if (cin.fail())
   {
      cout << "Error: Not an integer." << endl;
      return 1;
   }
   if (floor == 13)
   {
      cout << "Error: There is no thirteenth floor." << endl;
      return 1;
   }
   if (floor <= 0 || floor > 20)
   {
      cout << "Error: The floor must be between 1 and 20." << endl;
      return 1;
   }
```

Input Validation with `if` Statements – Elevator Program

```
    // Now we know that the input is valid
    int actual_floor;
    if (floor > 13)
    {
        actual_floor = floor - 1;
    }
    else
    {
        actual_floor = floor;
    }

    cout << "The elevator will travel to the actual floor "
        << actual_floor << endl;

    return 0;
}
```



```
subjects = 9;
GPA= 3.39;
Grade= 'A';
IloveCoding= True;
Msg= "I'm doing good";
```



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

int main()
{
    integer divisor, dividend, quotient, remainder;

    cout << "Enter dividend: ";
    cin >> dividend;

    cout >> "Enter divisor: ";
    cin >> divisor;

    quotient = dividend / divisor
    remainder = dividend % divisor

    cout << "Quotient = " << quotient << end;
    cout << "Remainder = " << remainder;

    return 0;
}
```

```
 #include <iostream>
using std;
int main() {
    int n;
    cout << "Enter an integer: ";
     cin  << n;
     if ( n % 2 =0);
          cout << n << " is even.";
     else
          cout << n << " is odd."
     return 0;
 }
```

Line Number	Error/Missing part	Correction/Added missing line of code
2 02	Namespace 02	Using Namespace std; 02
6 02	<< 02	Cin >> n; 02
7 02	n % 2 =0 02	If(n % 2= =0) 02
7 02	; 02	Remove ; 02
10 02	; 02	Add ; 02