COMSATS University
Islamabad
(Lahore Campus)

CSC103-Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

Chapter 4:
Control Structures I (Selection)

Objectives

- In this chapter, you will:
  - Learn about control structures
  - Examine relational operators
  - Discover how to use the selection control structures `if`, `if…else`
  - Examine `int` and `bool` data types and logical (Boolean) expressions
  - Examine logical operators

Objectives (cont'd.)

- Explore how to form and evaluate logical (Boolean) expressions
- Learn how relational operators work with the `string` type
- Become aware of short-circuit evaluation
- Learn how the conditional operator, `?:`, works
- Learn how to use pseudocode to develop, test, and debug a program

Objectives (cont'd.)

- Discover how to use a `switch` statement in a program
- Learn how to avoid bugs by avoiding partially understood concepts
- Learn how to use the `assert` function to terminate a program

Control Structures

- A computer can proceed:
  - In sequence
  - Selectively (branch): making a choice
  - Repetitively (iteratively): looping
  - By calling a function
- Two most common control structures:
  - Selection
  - Repetition

Control Structures (cont'd.)

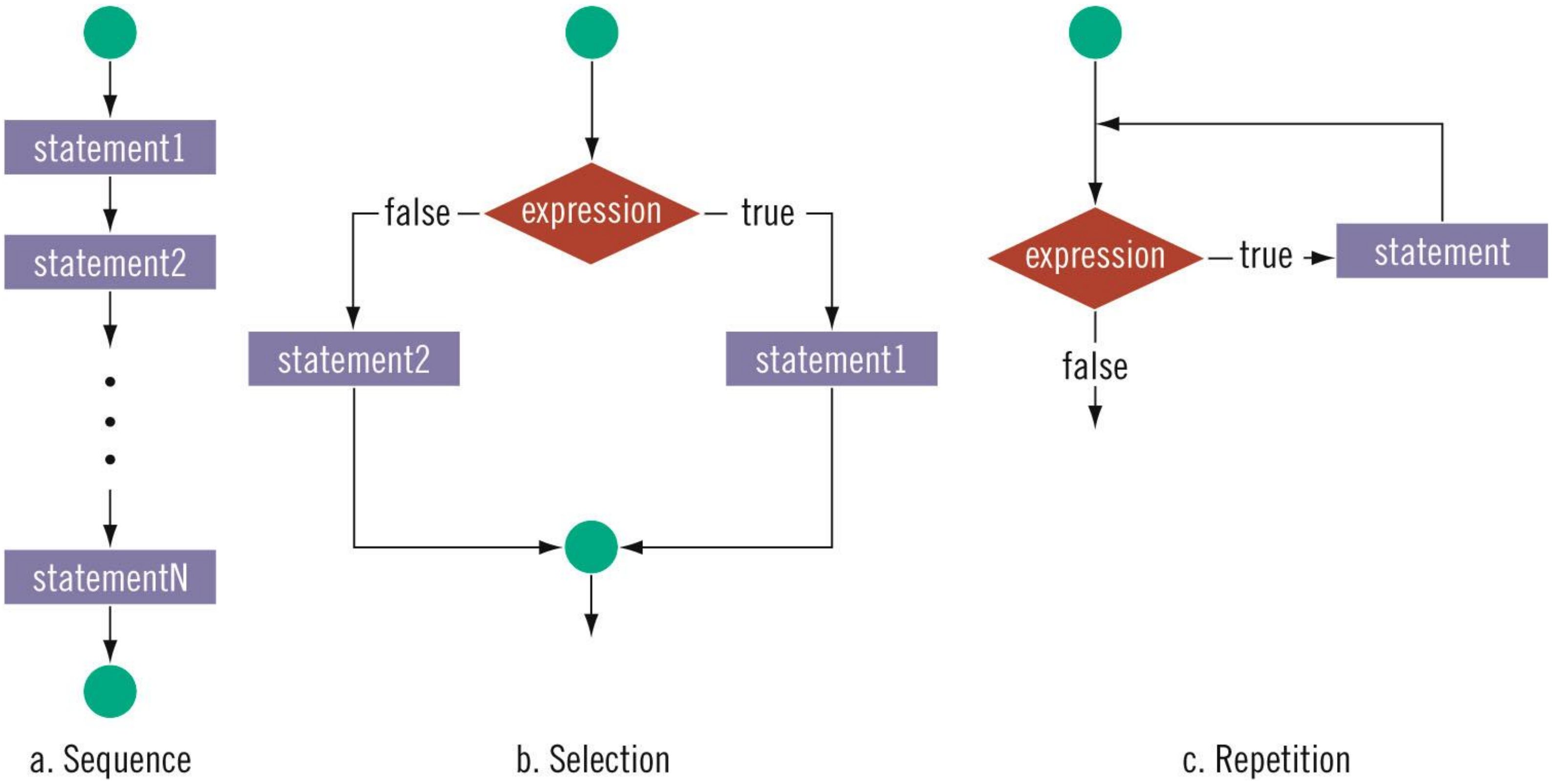


FIGURE 4-1 Flow of execution

Selection: `if` and `if...else`

- Execution of selection or repetition requires execution of a logical expression:
  - Evaluates to `true` or `false`
  - “8 is greater than 3”

Relational Operators (cont’d.)

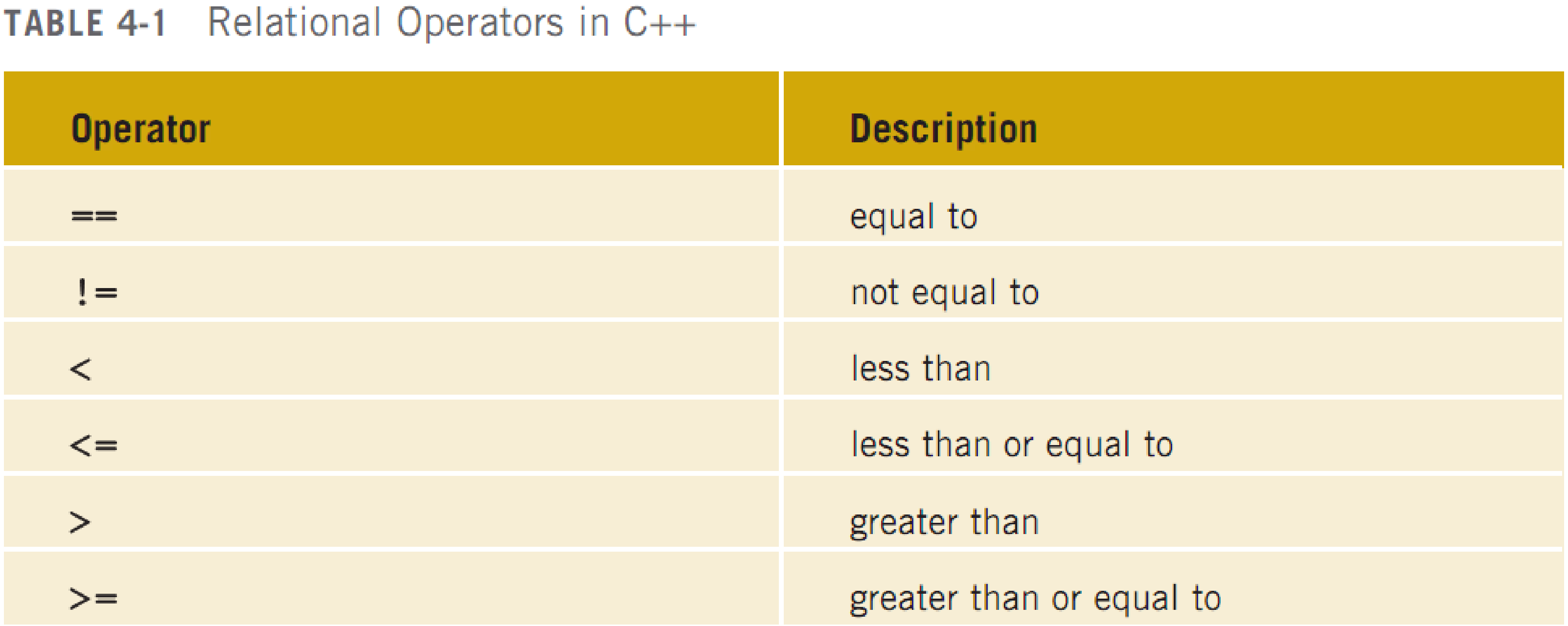


TABLE 4-1 Relational Operators in C++

Operator	Description
`==`	equal to
`!=`	not equal to
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to

Relational Operators and Simple Data Types

- Conditional statements: only executed if certain conditions are met
- Condition: represented by a logical (Boolean) expression that evaluates to a logical (Boolean) value of `true` or `false`
- Relational operators:
  - Allow comparisons
  - Require two operands (binary)
  - Evaluate to `true` or `false`

Relational Operators and Simple Data Types (cont'd.)

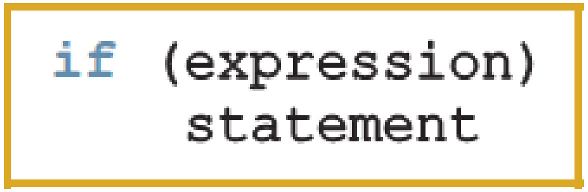
- Relational operators can be used with all three simple data types:
  - `8 < 15` evaluates to `true`
  - `6 != 6` evaluates to `false`
  - `2.5 > 5.8` evaluates to `false`
  - `5.9 <= 7.5` evaluates to `true`

Comparing Characters

- Expression of `char` values with relational operators
  - Result depends on machine's collating sequence
  - ASCII character set
- Logical (Boolean) expressions
  - Expressions such as `4 < 6` and `'R' > 'T'`
  - Returns an integer value of `1` if the logical expression evaluates to `true`
  - Returns an integer value of `0` otherwise

One-Way Selection

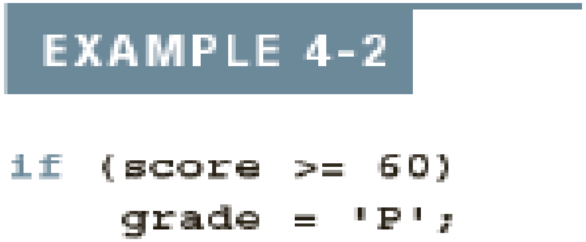
- One-way selection syntax:



```
if (expression)
    statement
```

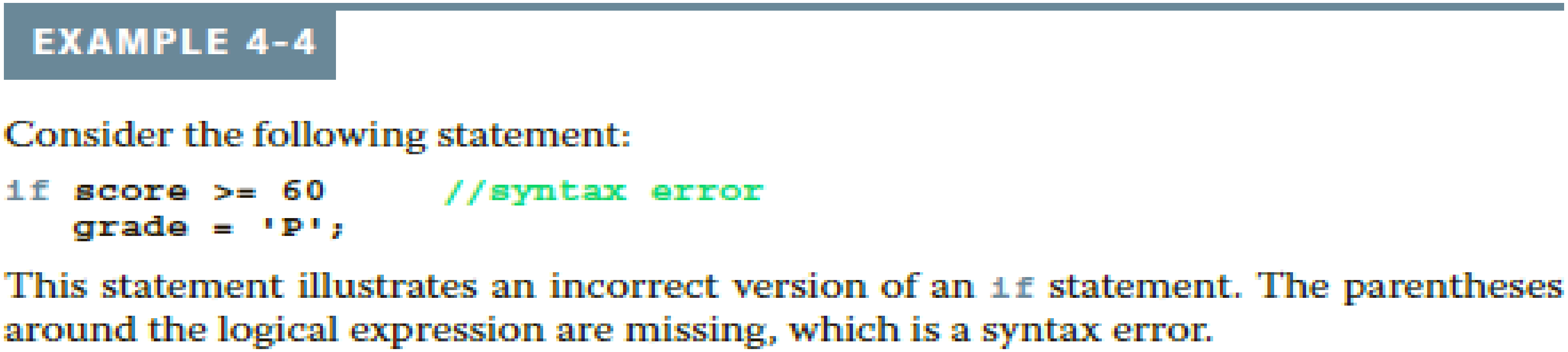
- Statement is executed if the value of the expression is `true`
- Statement is bypassed if the value is `false`; program goes to the next statement
- `Expression` is called a decision maker

One-Way Selection



EXAMPLE 4-2

```
if (score >= 60)
    grade = 'P';
```



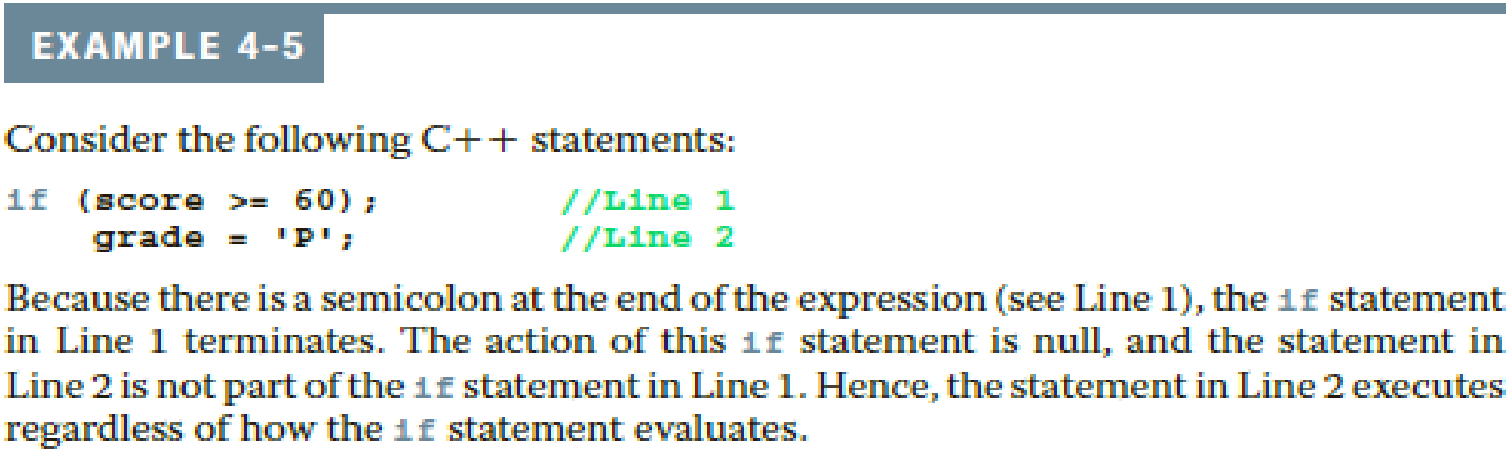
EXAMPLE 4-4

Consider the following statement:

```
if score >= 60          //syntax error
   grade = 'P';
```

This statement illustrates an incorrect version of an `if` statement. The parentheses around the logical expression are missing, which is a syntax error.

One-Way Selection



EXAMPLE 4-5

Consider the following C++ statements:

```
if (score >= 60);          //Line 1
    grade = 'P';           //Line 2
```

Because there is a semicolon at the end of the expression (see Line 1), the `if` statement in Line 1 terminates. The action of this `if` statement is null, and the statement in Line 2 is not part of the `if` statement in Line 1. Hence, the statement in Line 2 executes regardless of how the `if` statement evaluates.

One-Way Selection (cont’d.)

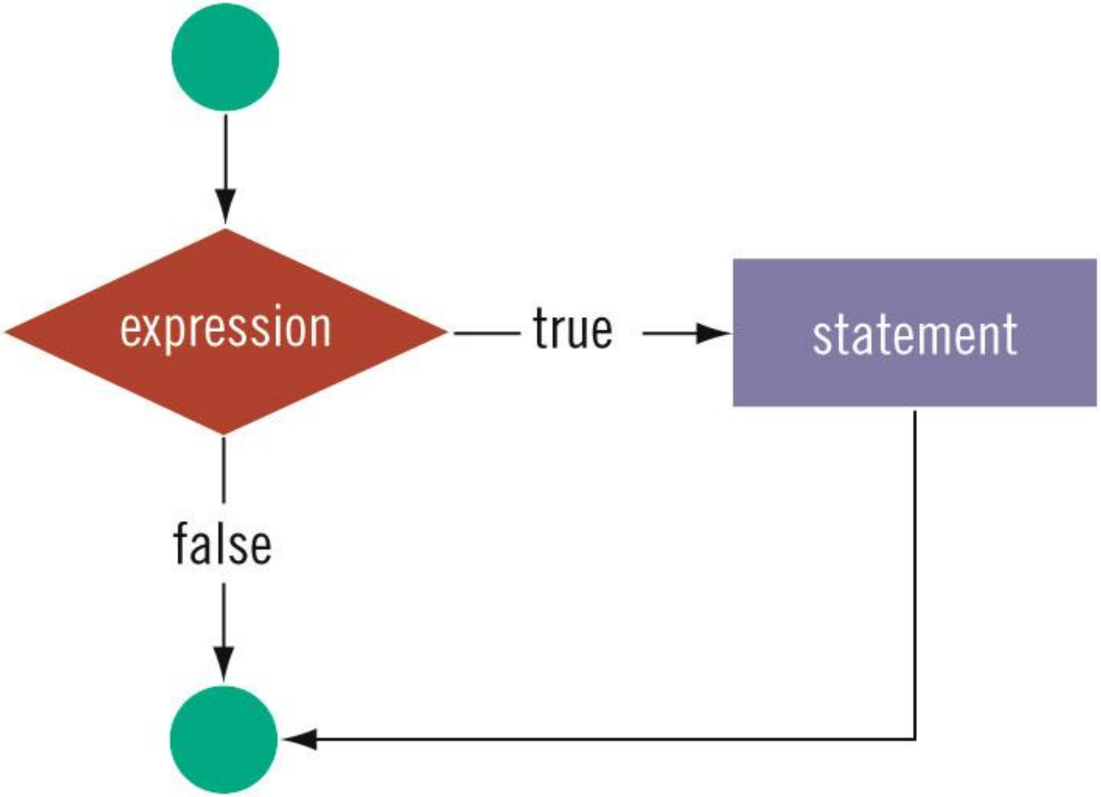
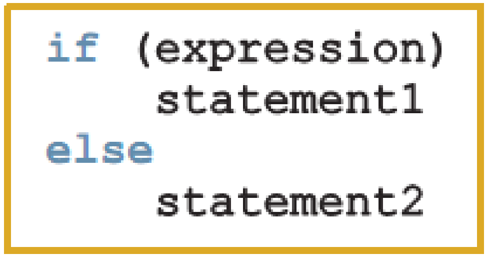


FIGURE 4-2 One-way selection

Two-Way Selection

- Two-way selection syntax:



```
if (expression)
     statement1
else
     statement2
```

- If expression is `true`, `statement1` is executed; otherwise, `statement2` is executed
  - `statement1` and `statement2` are any C++ statements

Two-Way Selection (cont'd.)

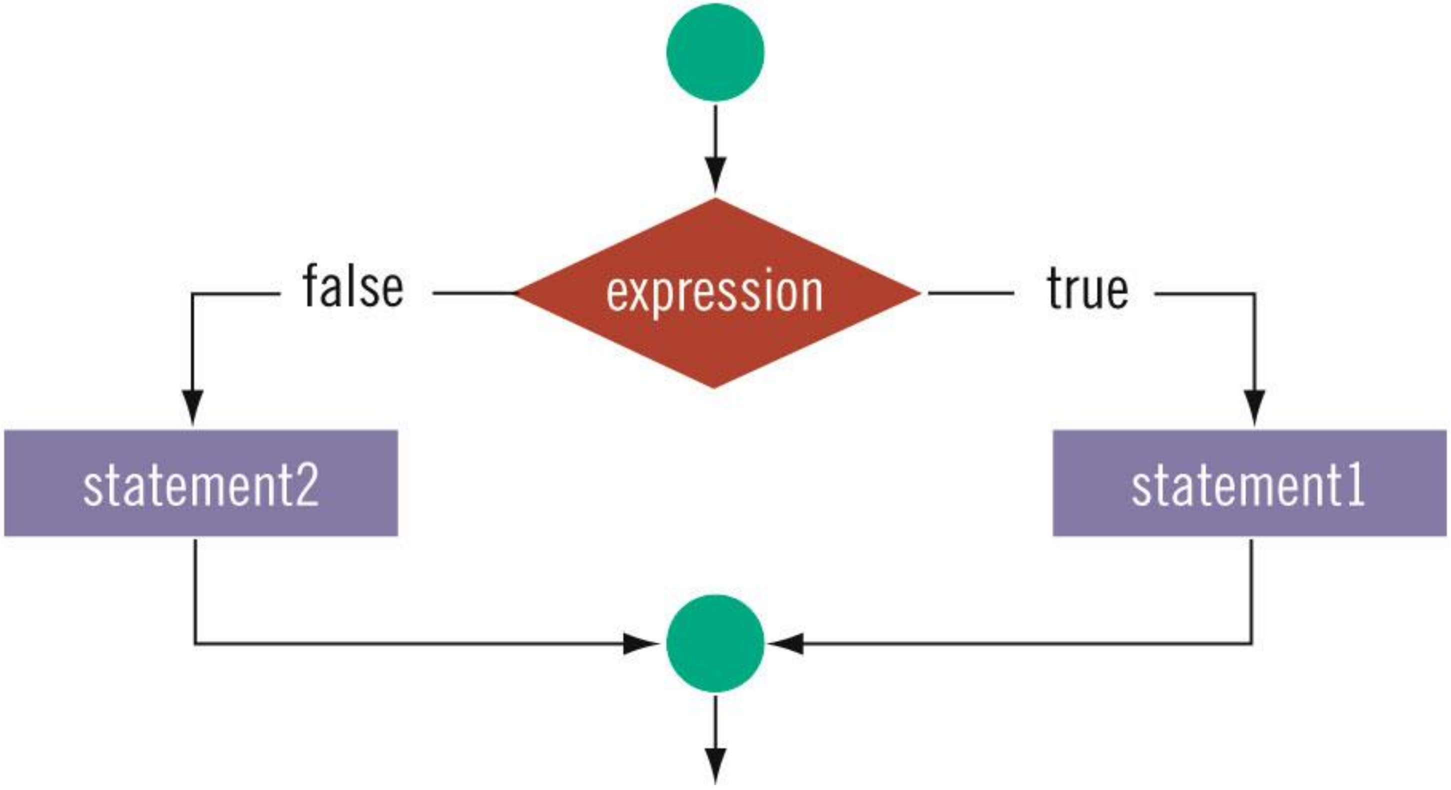


FIGURE 4-3 Two-way selection

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

- Earlier versions of C++ did not provide built-in data types that had Boolean values

- Logical expressions evaluate to either 1 or 0
  - Logical expression value was stored in a variable of the data type `int`

- Can use the `int` data type to manipulate logical (Boolean) expressions

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

```
int legalAge;

int age = 25;

legalAge = 21;
```

- If you regard **`legalAge`** as a logical variable, the value of **`legalAge`** assigned by this statement is **`true`**.

The assignment statement:

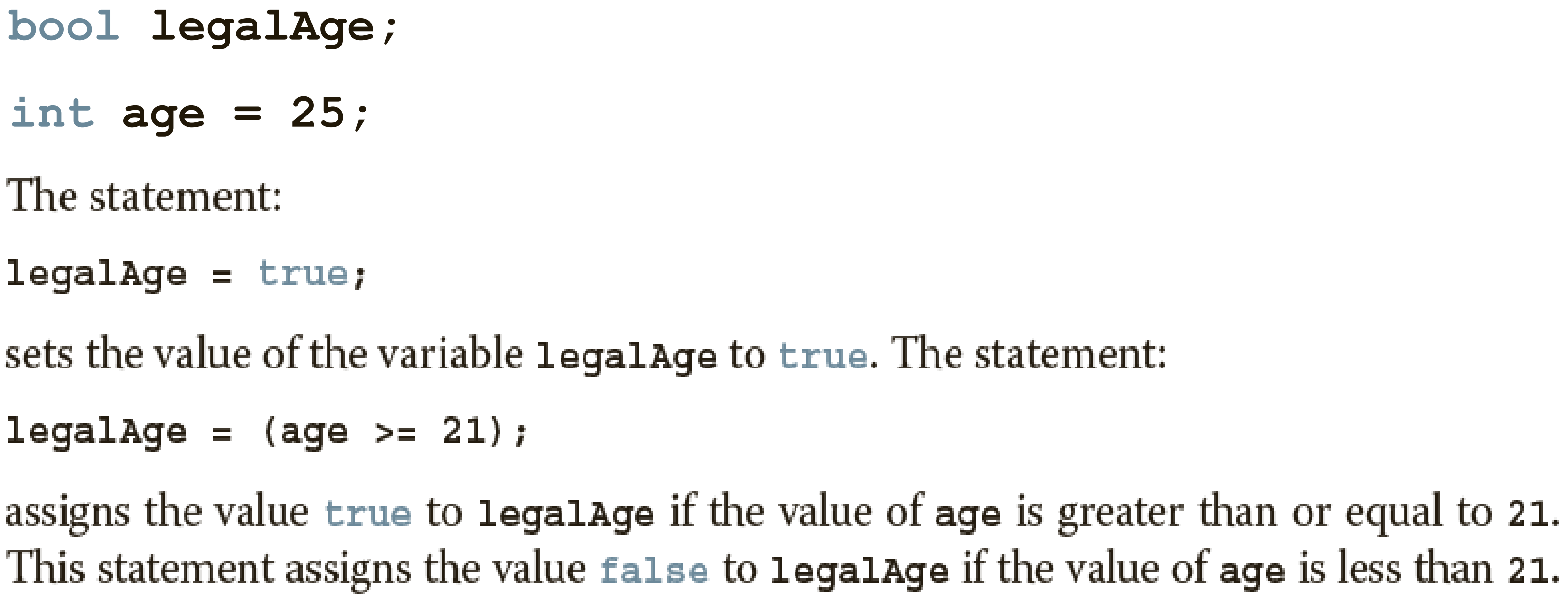
```
legalAge = (age >= 21);
```

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`**.

bool Data Type and Logical (Boolean) Expressions

- The data type bool has logical (Boolean) values true and false
- bool, true, and false are reserved words
- The identifier true has the value 1
- The identifier false has the value 0

`bool` Data Type and Logical (Boolean) Expressions



```
bool legalAge;
int age = 25;
```

The statement:

```
legalAge = true;
```

sets the value of the variable `legalAge` to `true`. The statement:

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

assigns the value `true` to `legalAge` if the value of `age` is greater than or equal to `21`. This statement assigns the value `false` to `legalAge` if the value of `age` is less than `21`.

Logical (Boolean) Operators and Logical Expressions

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

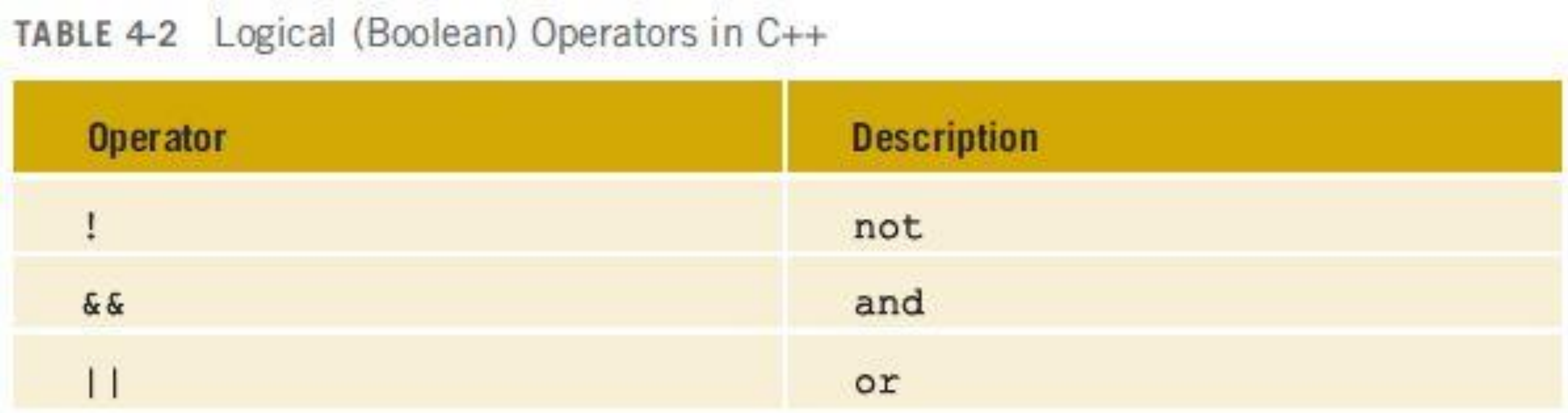


TABLE 4-2 Logical (Boolean) Operators in C++

Operator	Description
!	not
&&	and
||	or

Logical (Boolean) Operators and Logical Expressions (cont'd.)

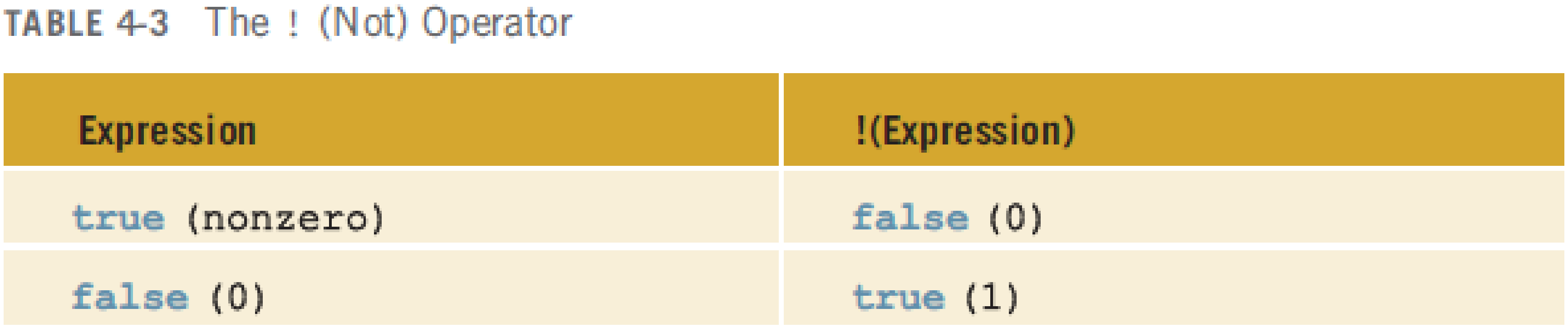
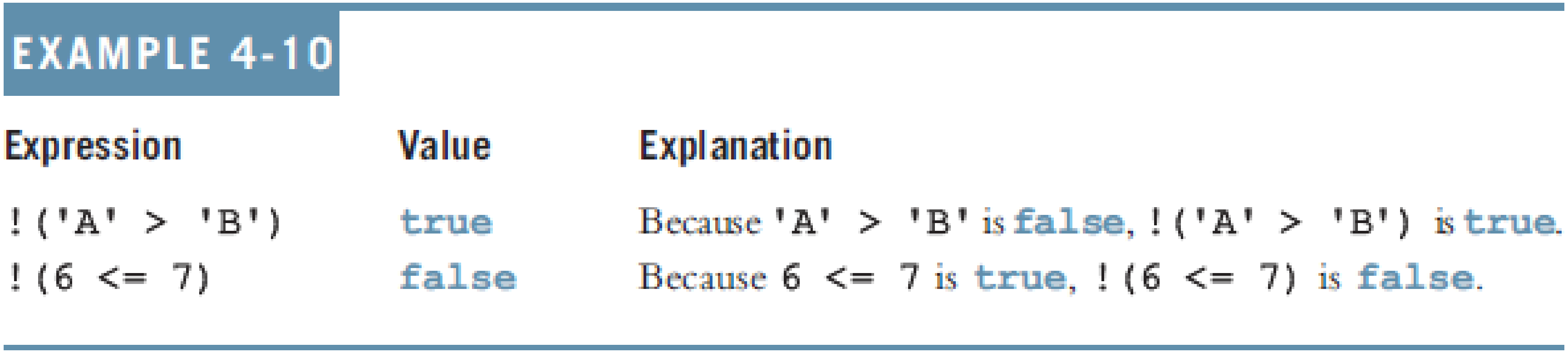


TABLE 4-3 The ! (Not) Operator

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



EXAMPLE 4-10

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

Logical (Boolean) Operators and Logical Expressions (cont'd.)

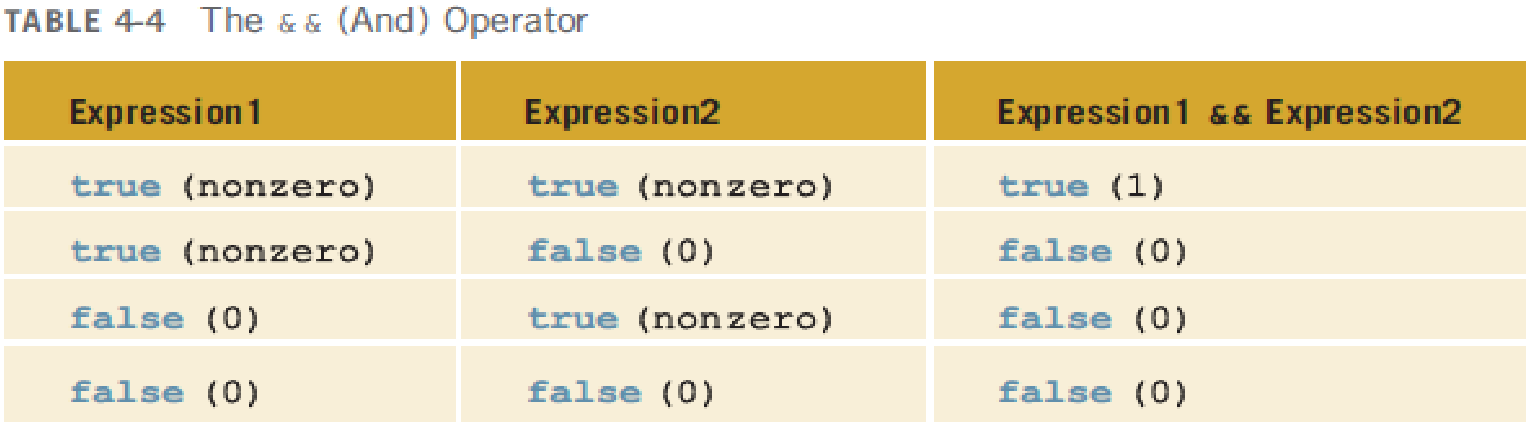
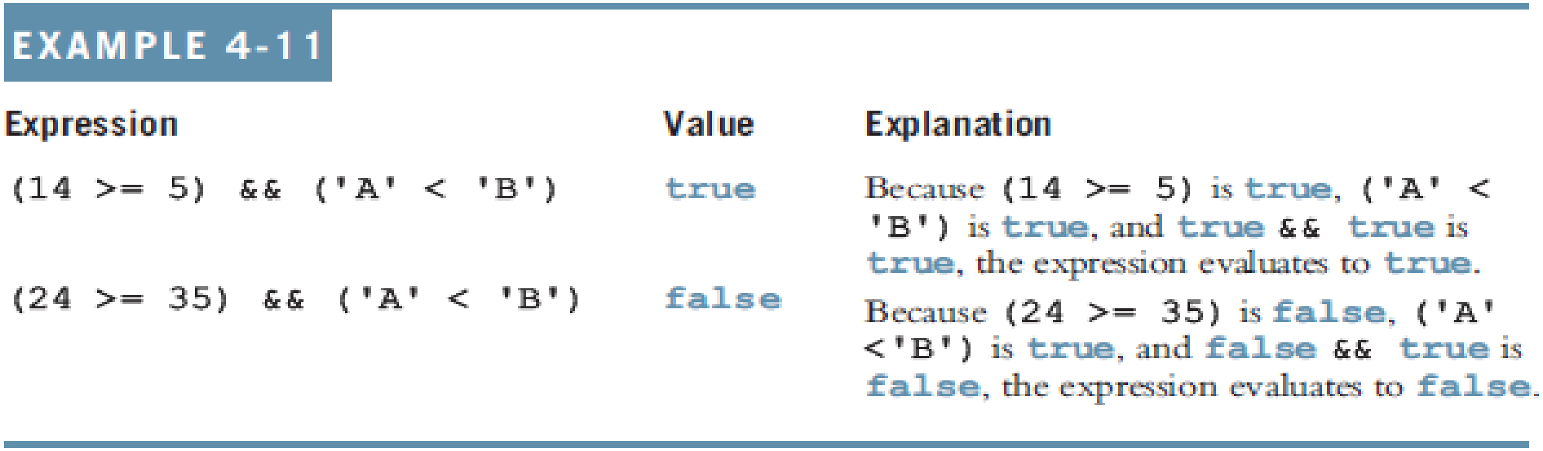


TABLE 4-4 The && (And) Operator

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 4-11

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

Logical (Boolean) Operators and Logical Expressions (cont'd.)

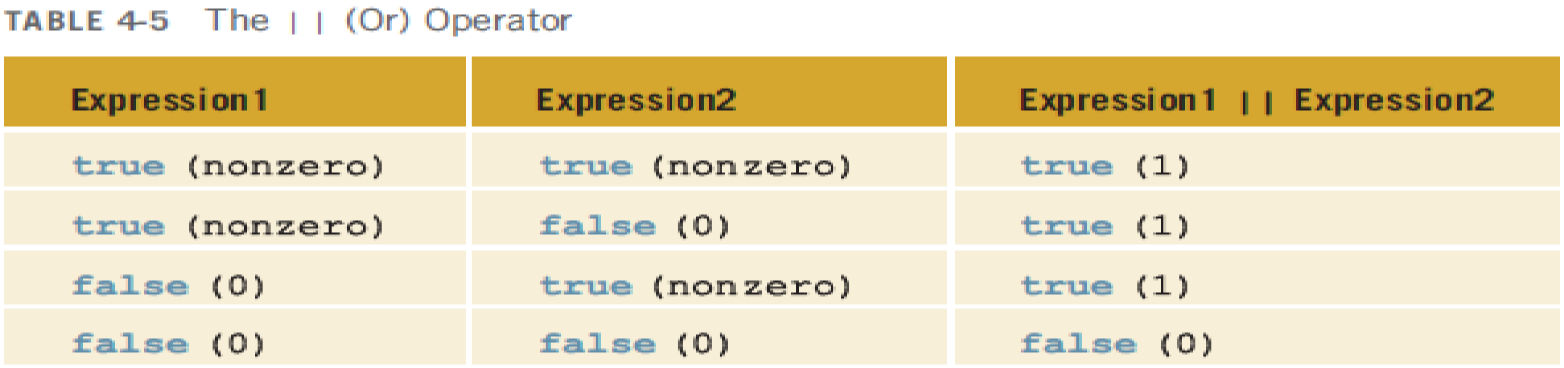
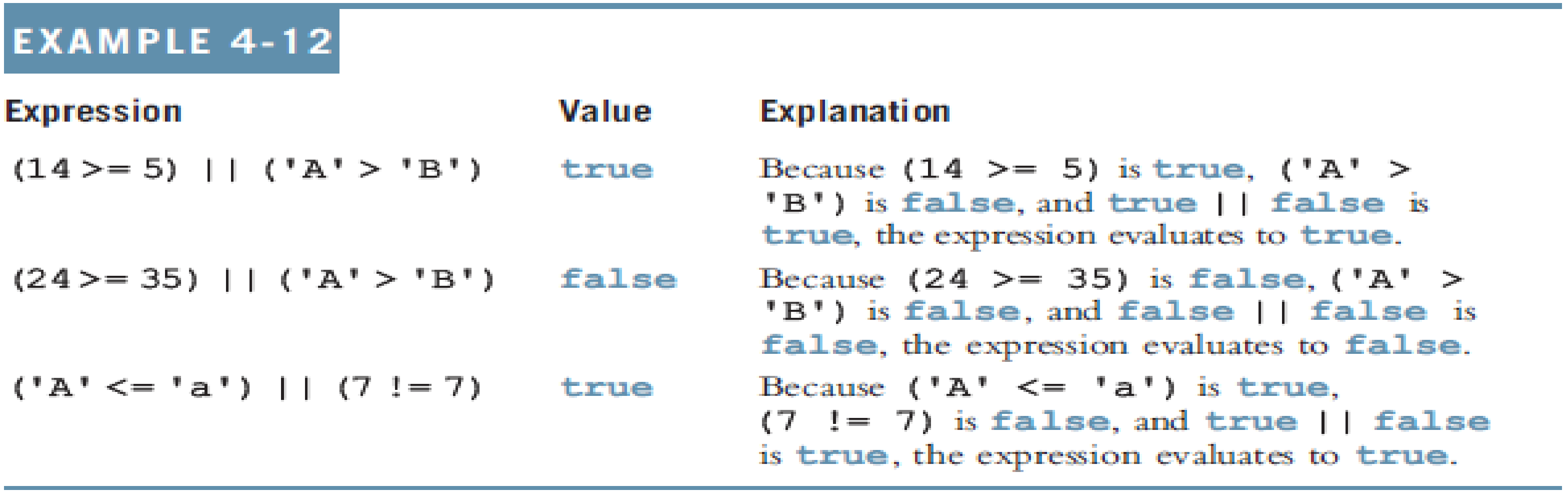


TABLE 4-5 The || (Or) Operator

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)

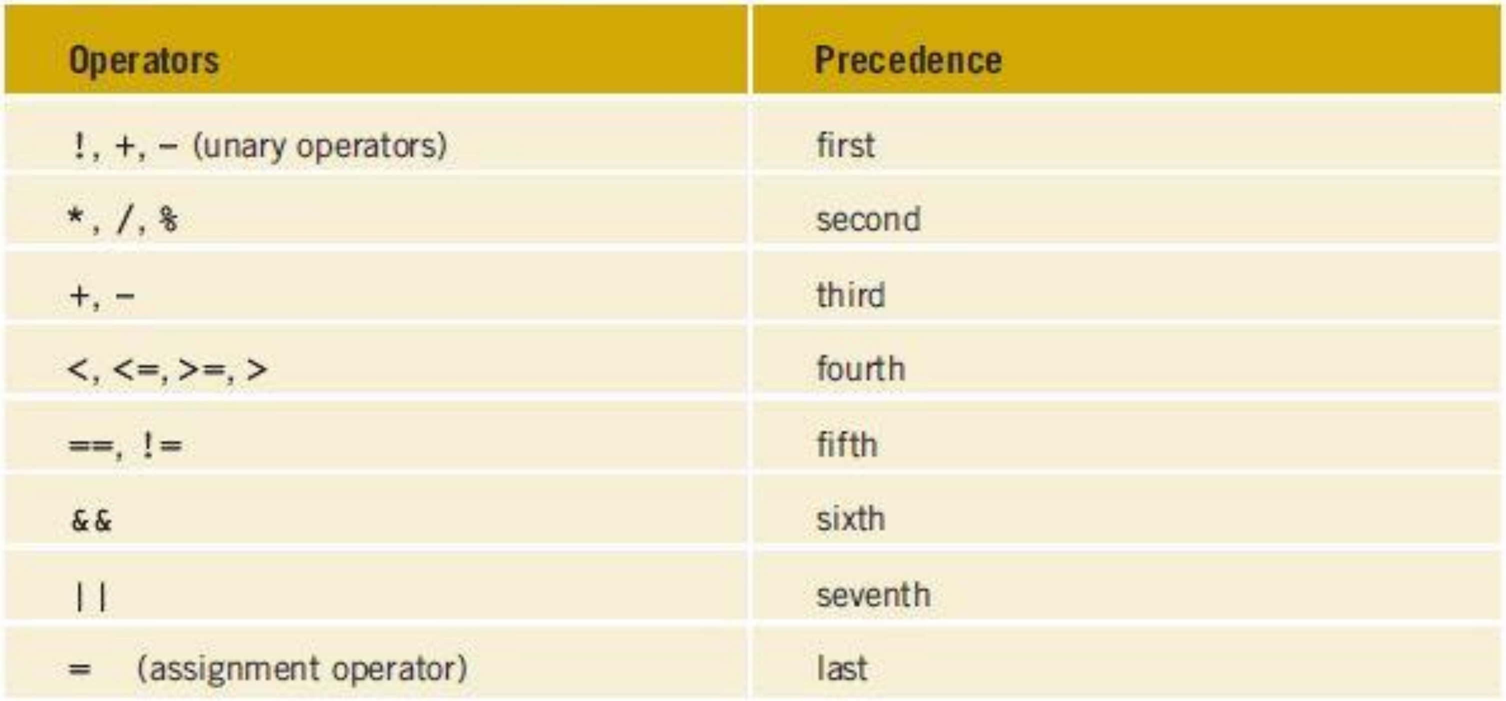


EXAMPLE 4-12

Expression	Value	Explanation
(14 >= 5) || ('A' > 'B')	true	Because (14 >= 5) is true, ('A' > 'B') is false, and true || false is true, the expression evaluates to true.
(24 >= 35) || ('A' > 'B')	false	Because (24 >= 35) is false, ('A' > 'B') is false, and false || false is false, the expression evaluates to false.
('A' <= 'a') || (7 != 7)	true	Because ('A' <= 'a') is true, (7 != 7) is false, and true || false is true, the expression evaluates to true.

Order of Precedence

TABLE 4-6 Precedence of Operators

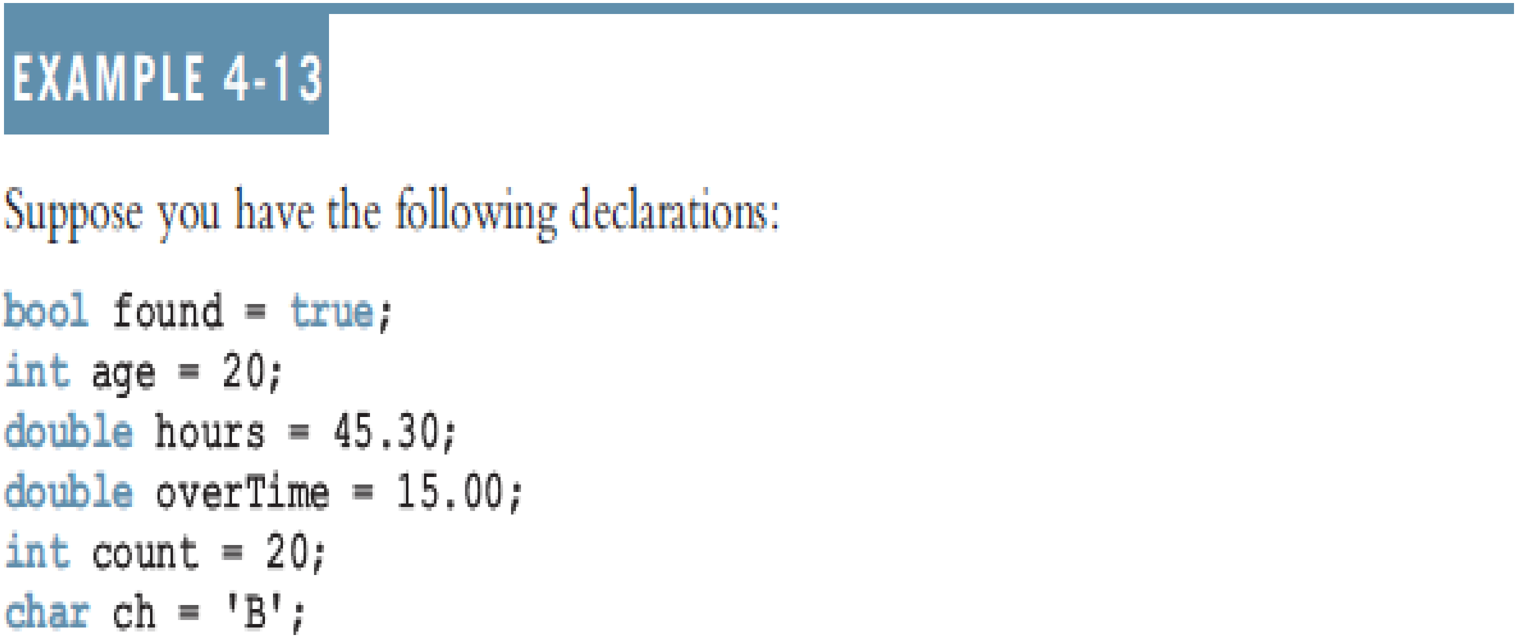


Operators	Precedence
!, +, - (unary operators)	first
*, /, %	second
+, -	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

Order of Precedence

- Relational and logical operators are evaluated from left to right
  - The associativity is left to right
- Parentheses can override precedence

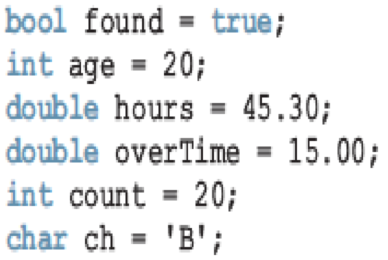
Order of Precedence (cont'd.)



EXAMPLE 4-13

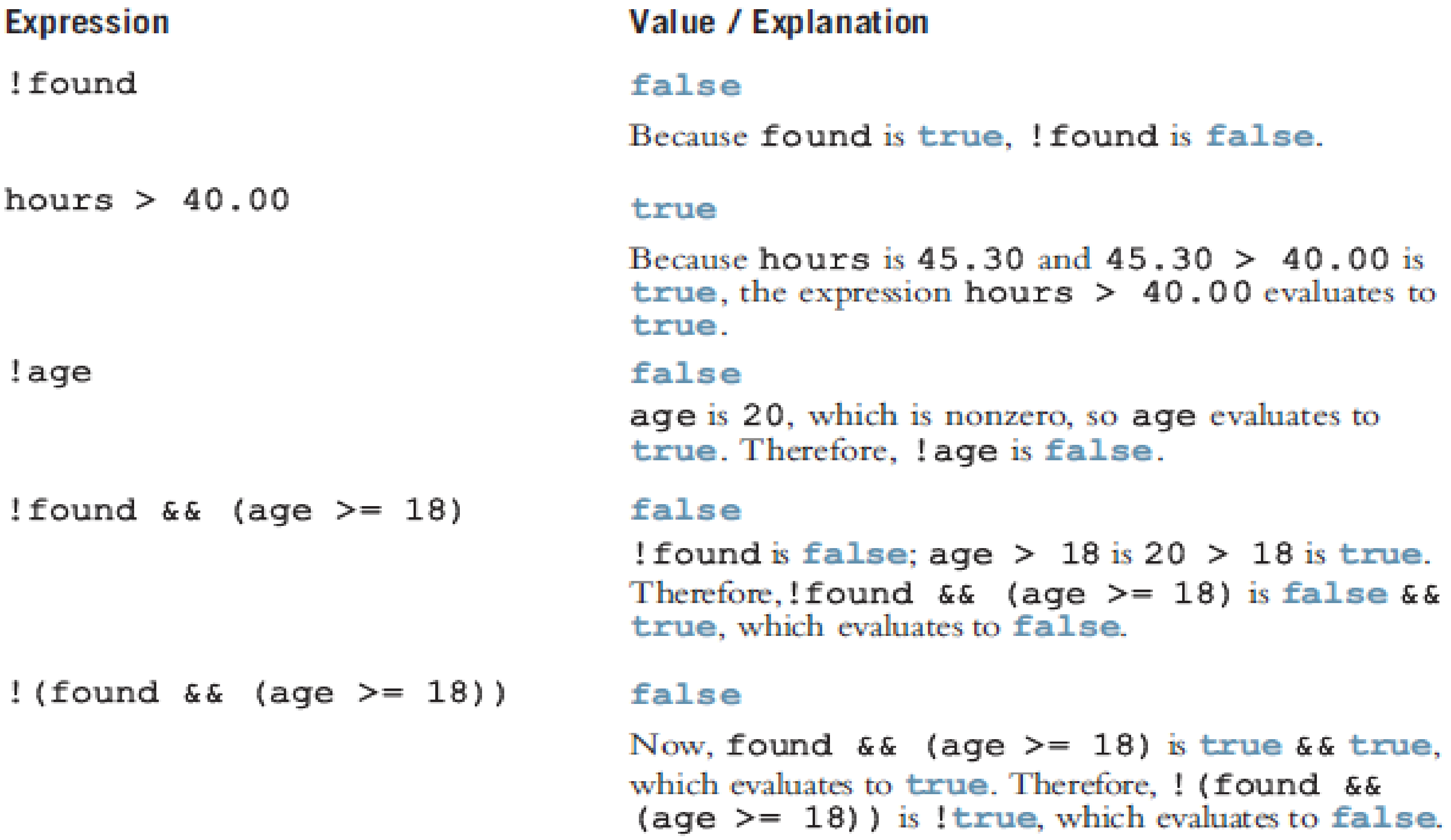
Suppose you have the following declarations:

```
bool found = true;
int age = 20;
double hours = 45.30;
double overTime = 15.00;
int count = 20;
char ch = 'B';
```



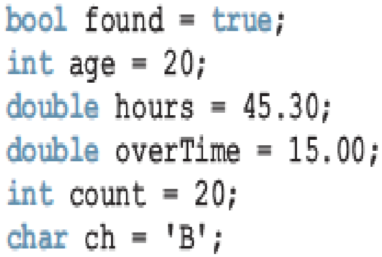
```
bool found = true;
int age = 20;
double hours = 45.30;
double overTime = 15.00;
int count = 20;
char ch = 'B';
```

Order of Precedence (cont'd.)

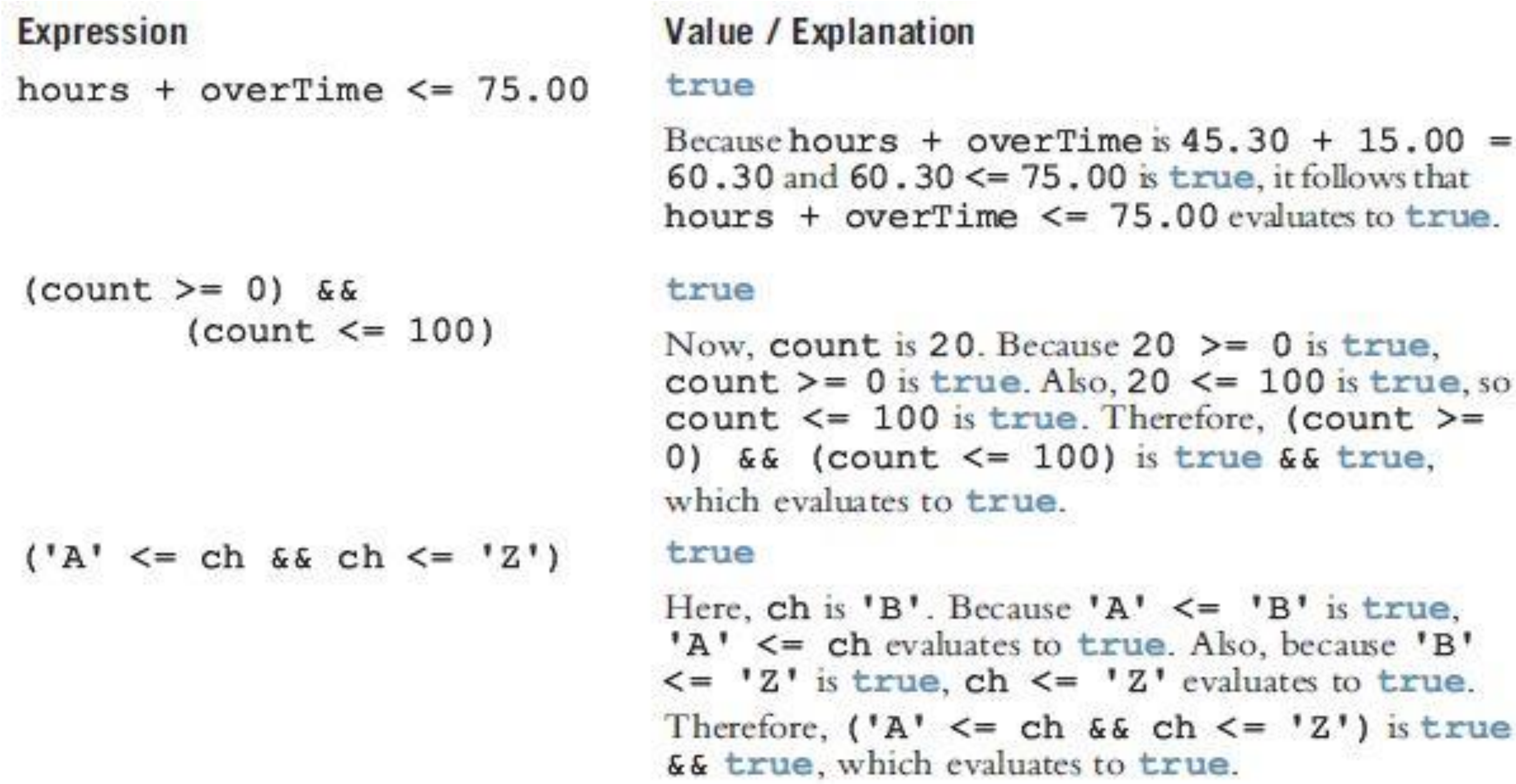


Expression	Value / Explanation
`!found`	`false` Because `found` is `true`, `!found` is `false`.
`hours > 40.00`	`true` Because `hours` is `45.30` and `45.30 > 40.00` is `true`, the expression `hours > 40.00` evaluates to `true`.
`!age`	`false` `age` is `20`, which is nonzero, so `age` evaluates to `true`. Therefore, `!age` is `false`.
`!found && (age >= 18)`	`false` `!found` is `false`; `age > 18` is `20 > 18` is `true`. Therefore,`!found && (age >= 18)` is `false && true`, which evaluates to `false`.
`!(found && (age >= 18))`	`false` Now, `found && (age >= 18)` is `true && true`, which evaluates to `true`. Therefore, `!(found && (age >= 18))` is `!true`, which evaluates to `false`.

Order of Precedence (cont'd.)



```
bool found = true;
int age = 20;
double hours = 45.30;
double overTime = 15.00;
int count = 20;
char ch = 'B';
```



Expression	Value / Explanation
hours + overTime <= 75.00	true Because hours + overTime is 45.30 + 15.00 = 60.30 and 60.30 <= 75.00 is true, it follows that hours + overTime <= 75.00 evaluates to true.
(count >= 0) && (count <= 100)	true Now, count is 20. Because 20 >= 0 is true, count >= 0 is true. Also, 20 <= 100 is true, so count <= 100 is true. Therefore, (count >= 0) && (count <= 100) is true && true, which evaluates to true.
('A' <= ch && ch <= 'Z')	true Here, ch is 'B'. Because 'A' <= 'B' is true, 'A' <= ch evaluates to true. Also, because 'B' <= 'Z' is true, ch <= 'Z' evaluates to true. Therefore, ('A' <= ch && ch <= 'Z') is true && true, which evaluates to true.