Programming Language (630203)
Fall 2010/2011 – Lecture Notes # 4

Control Structures in C++

Objectives of the Lecture

- **Control Structures.**
- **Relational Operators.**
- **Logical Operators.**
- **Evaluate Logical (Boolean) Expressions.**

Control Structures

- A computer can proceed:
    - In **sequence**
    - **Selectively** (branch): making a choice
    - **Repetitively** (iteratively): looping
- Some statements are executed only if certain conditions are met.
- A condition is met if it evaluates to true

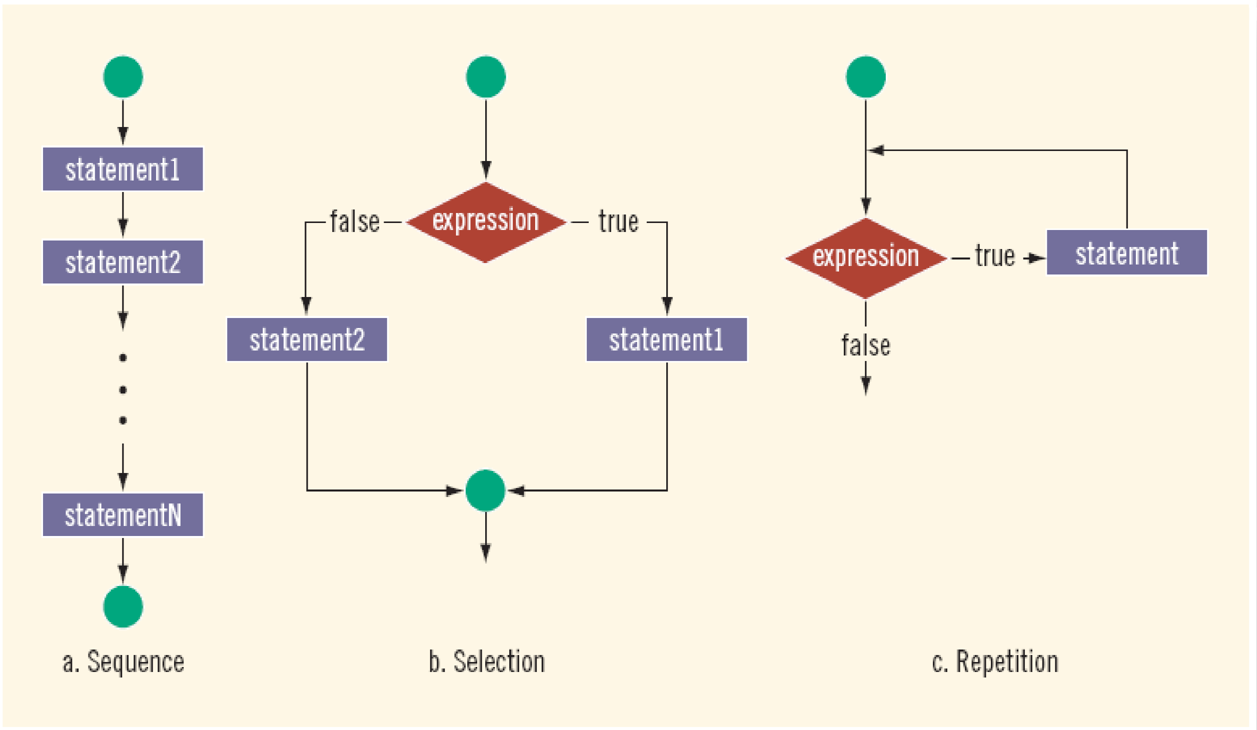


FIGURE 4-1 Flow of execution

Relational Operators

A condition is represented by a logical (Boolean) expression that can be true or false

Relational operators:

- Allow comparisons
- Require two operands (binary)
- Evaluate to true or false

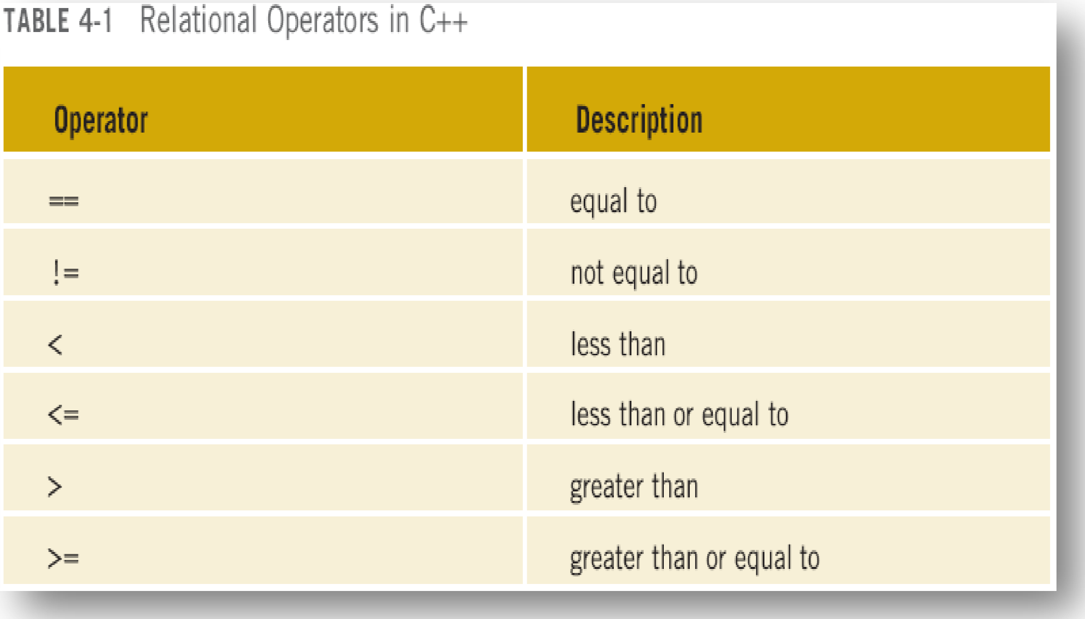


TABLE 4-1 Relational Operators in C++

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

You can use the relational operators 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

- Expressions such as 4 < 6 and 'R' > 'T' returns an integer value of 1 if the logical expression evaluates to true and returns an integer value of 0 otherwise.

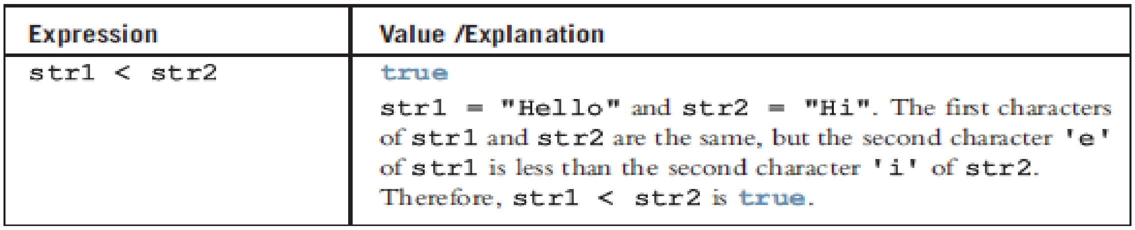
Relational Operators and the string Type

Relational operators can be applied to strings.

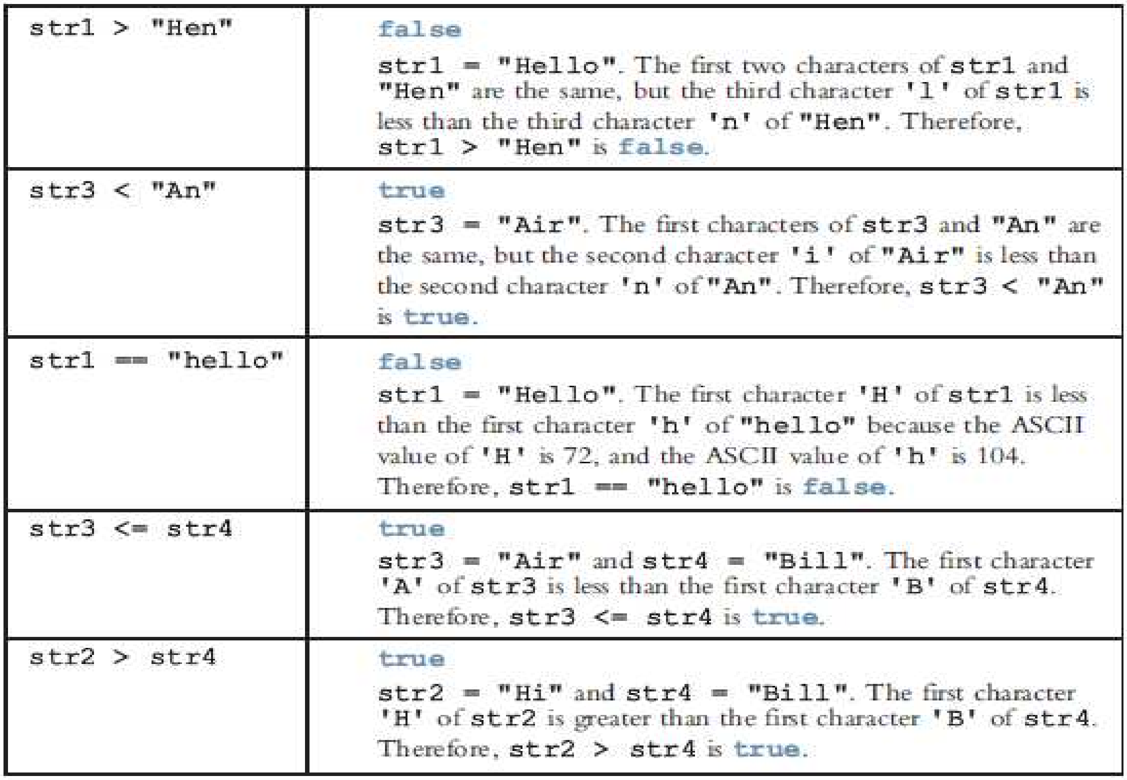
- Strings are compared character by character, starting with the first character.
- Comparison continues until either a mismatch is found or all characters are found equal.
- If two strings of different lengths are compared and the comparison is equal to the last character of the shorter string.
    - The shorter string is less than the larger string

- Suppose we have the following declarations:

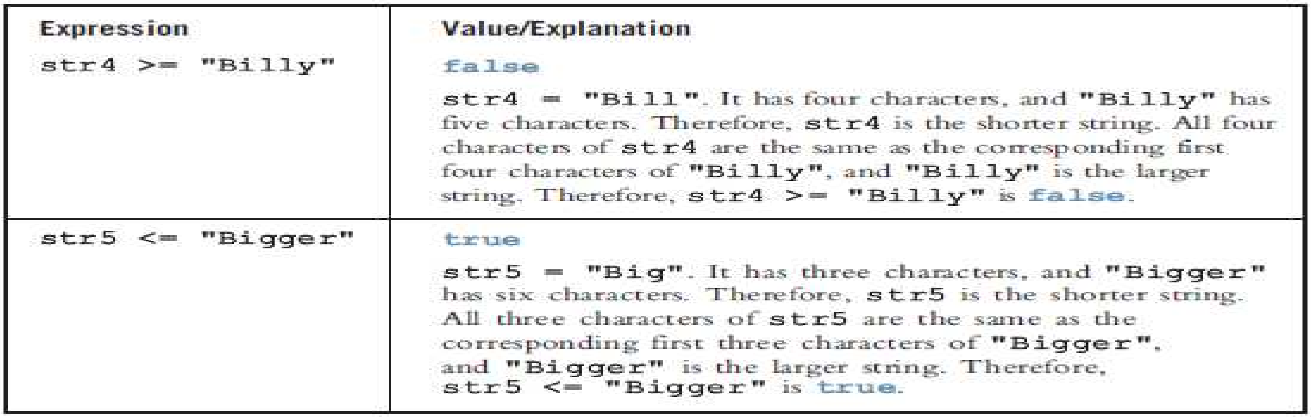
```
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str4 = "Big";
```



Expression	Value /Explanation
`str1 < str2`	`true` `str1 = "Hello"` and `str2 = "Hi"`. The first characters of `str1` and `str2` are the same, but the second character `'e'` of `str1` is less than the second character `'i'` of `str2`. Therefore, `str1 < str2` is `true`.



`str1 > "Hen"`	`false` `str1 = "Hello"`. The first two characters of `str1` and `"Hen"` are the same, but the third character `'l'` of `str1` is less than the third character `'n'` of `"Hen"`. Therefore, `str1 > "Hen"` is `false`.
`str3 < "An"`	`true` `str3 = "Air"`. The first characters of `str3` and `"An"` are the same, but the second character `'i'` of `"Air"` is less than the second character `'n'` of `"An"`. Therefore, `str3 < "An"` is `true`.
`str1 == "hello"`	`false` `str1 = "Hello"`. The first character `'H'` of `str1` is less than the first character `'h'` of `"hello"` because the ASCII value of `'H'` is 72, and the ASCII value of `'h'` is 104. Therefore, `str1 == "hello"` is `false`.
`str3 <= str4`	`true` `str3 = "Air"` and `str4 = "Bill"`. The first character `'A'` of `str3` is less than the first character `'B'` of `str4`. Therefore, `str3 <= str4` is `true`.
`str2 > str4`	`true` `str2 = "Hi"` and `str4 = "Bill"`. The first character `'H'` of `str2` is greater than the first character `'B'` of `str4`. Therefore, `str2 > str4` is `true`.



Expression	Value/Explanation
`str4 >= "Billy"`	`false` `str4 = "Bill"`. It has four characters, and `"Billy"` has five characters. Therefore, `str4` is the shorter string. All four characters of `str4` are the same as the corresponding first four characters of `"Billy"`, and `"Billy"` is the larger string. Therefore, `str4 >= "Billy"` is `false`.
`str5 <= "Bigger"`	`true` `str5 = "Big"`. It has three characters, and `"Bigger"` has six characters. Therefore, `str5` is the shorter string. All three characters of `str5` are the same as the corresponding first three characters of `"Bigger"`, and `"Bigger"` is the larger string. Therefore, `str5 <= "Bigger"` is `true`.

Logical Operators

Logical (Boolean) operators enable you to combine logical expressions

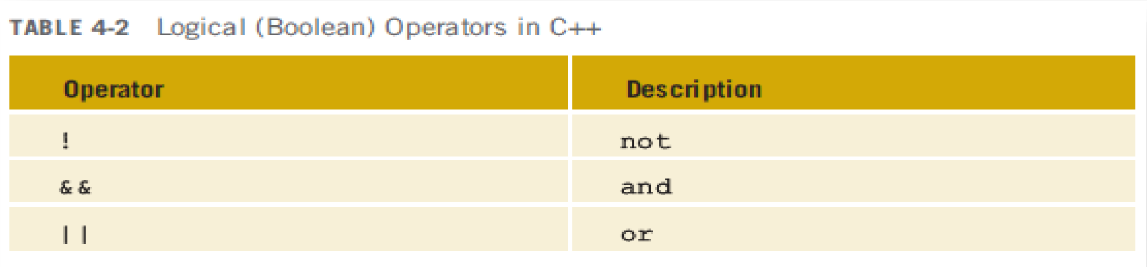


TABLE 4-2 Logical (Boolean) Operators in C++

Operator	Description
!	not
&&	and
||	or

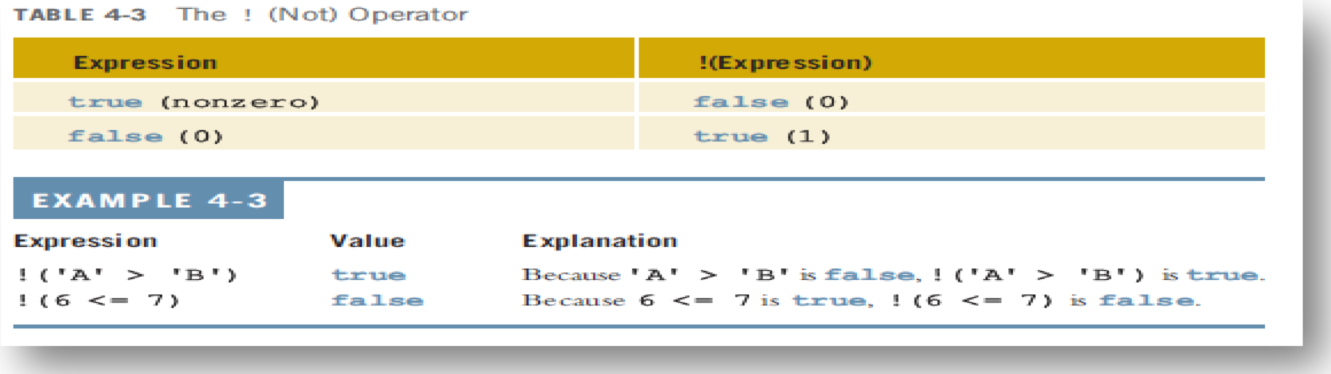


TABLE 4-3 The ! (Not) Operator

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

EXAMPLE 4-3

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.

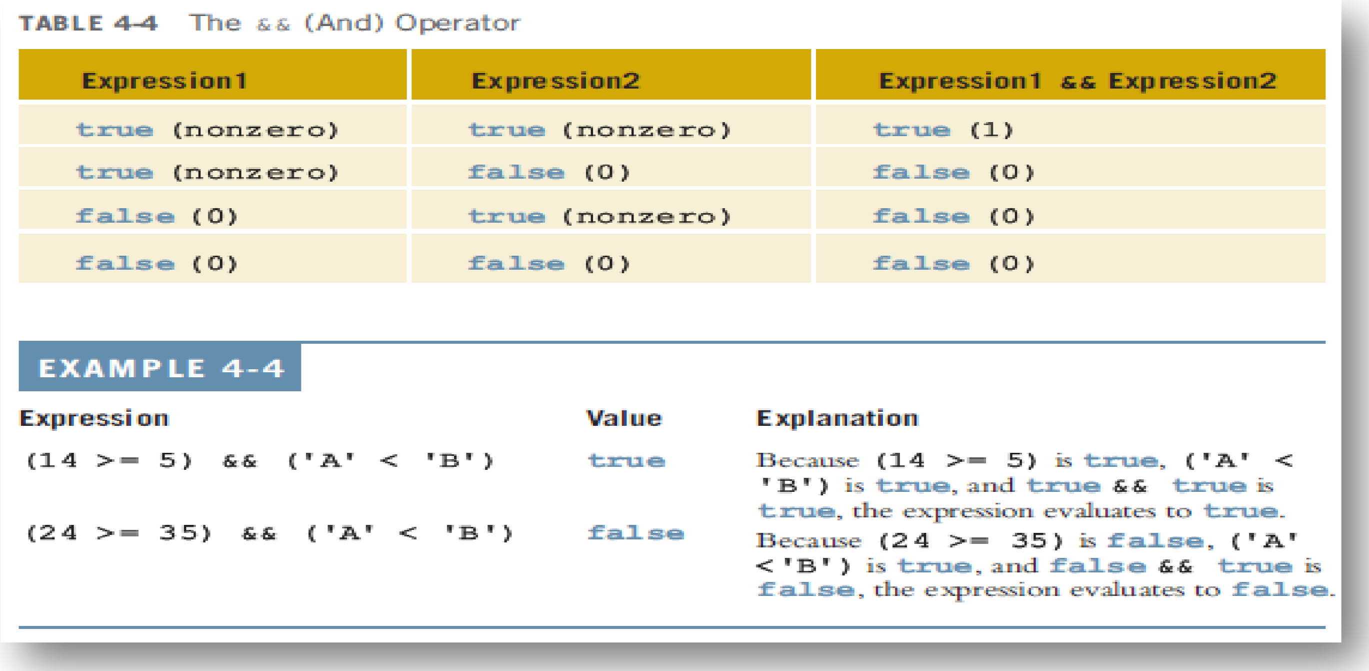


TABLE 4-4 The && (And) Operator

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 4-4

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.

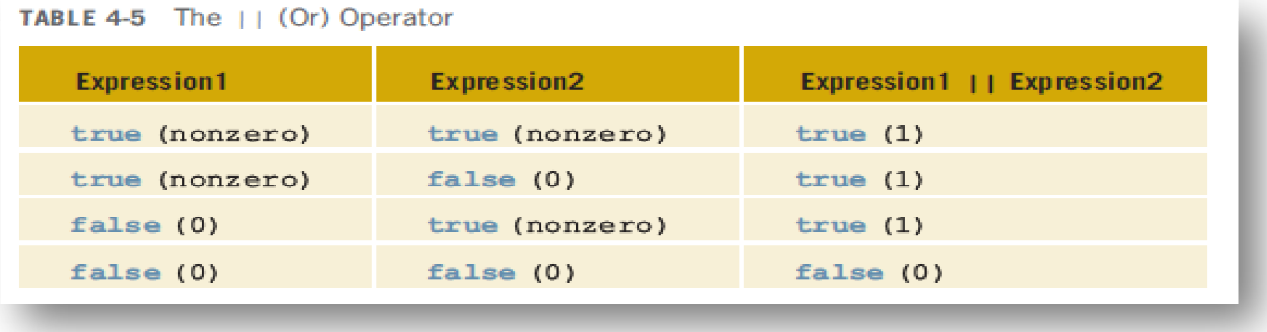
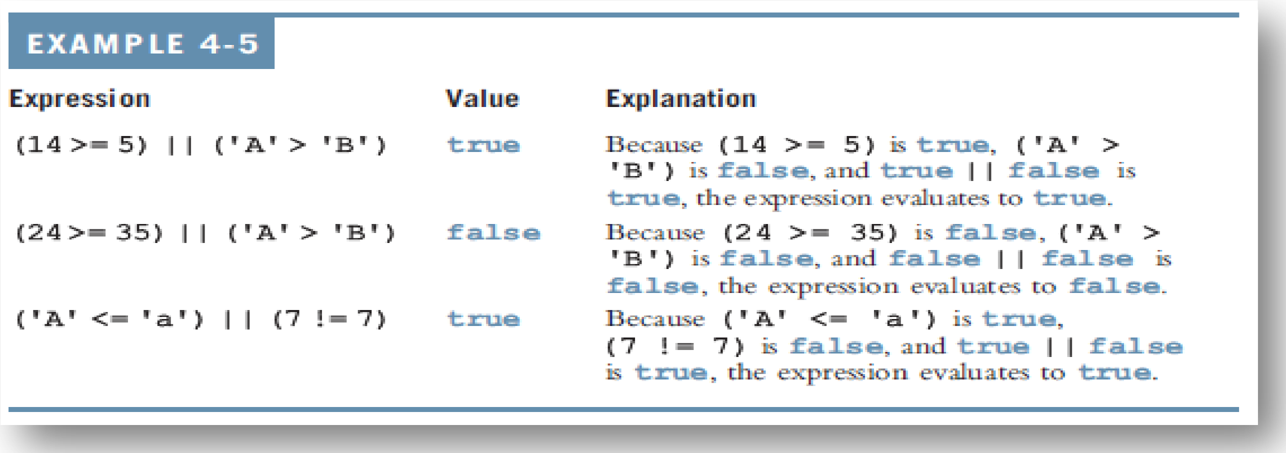


TABLE 4-5 The || (Or) Operator

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 4-5

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

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

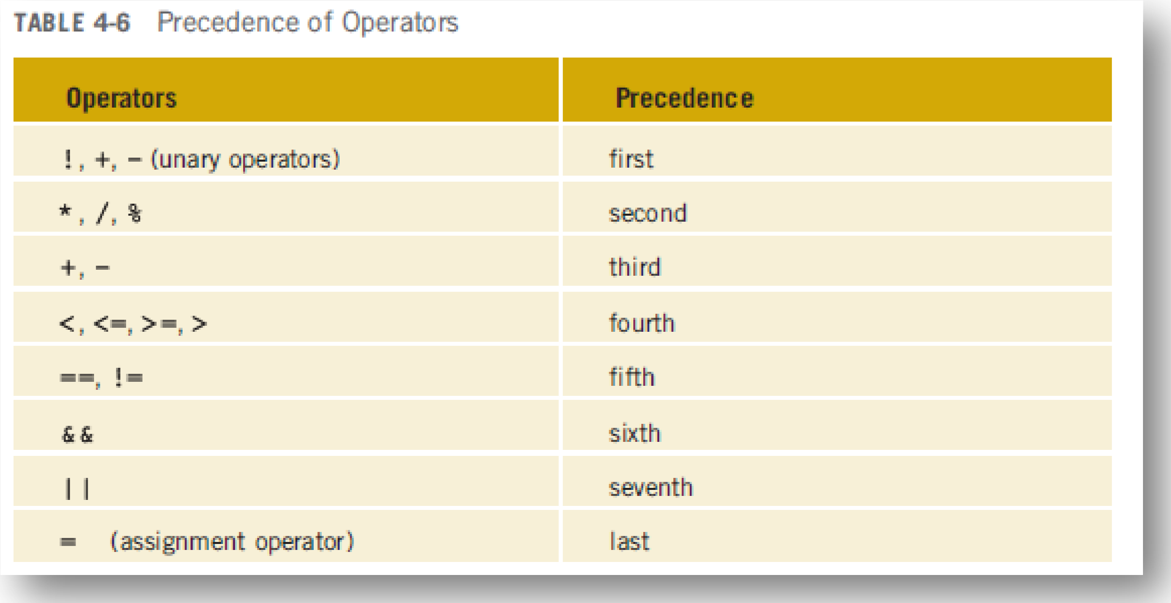
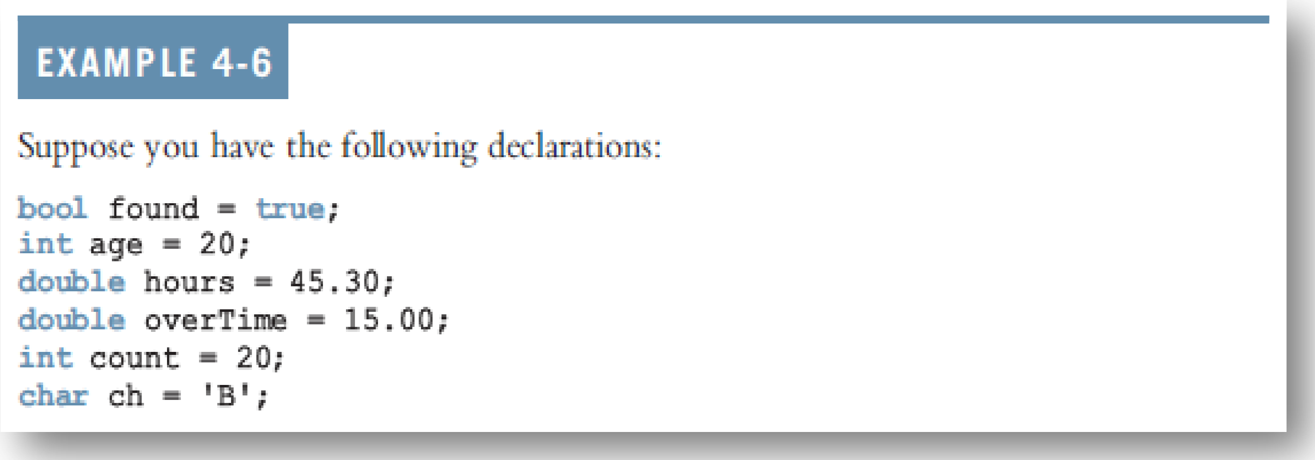


TABLE 4-6 Precedence of Operators

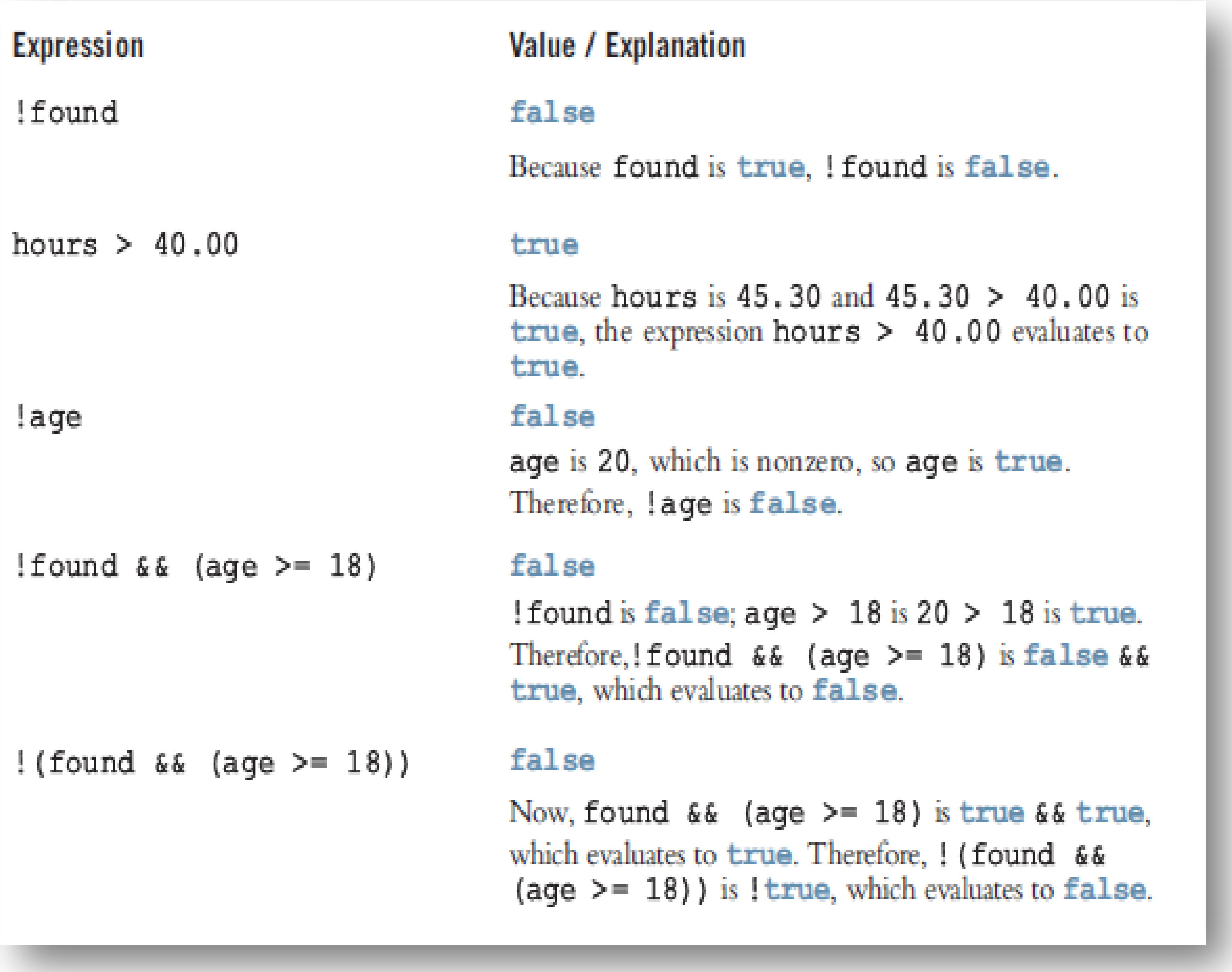
Operators	Precedence
!, +, - (unary operators)	first
*, /, %	second
+, -	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last



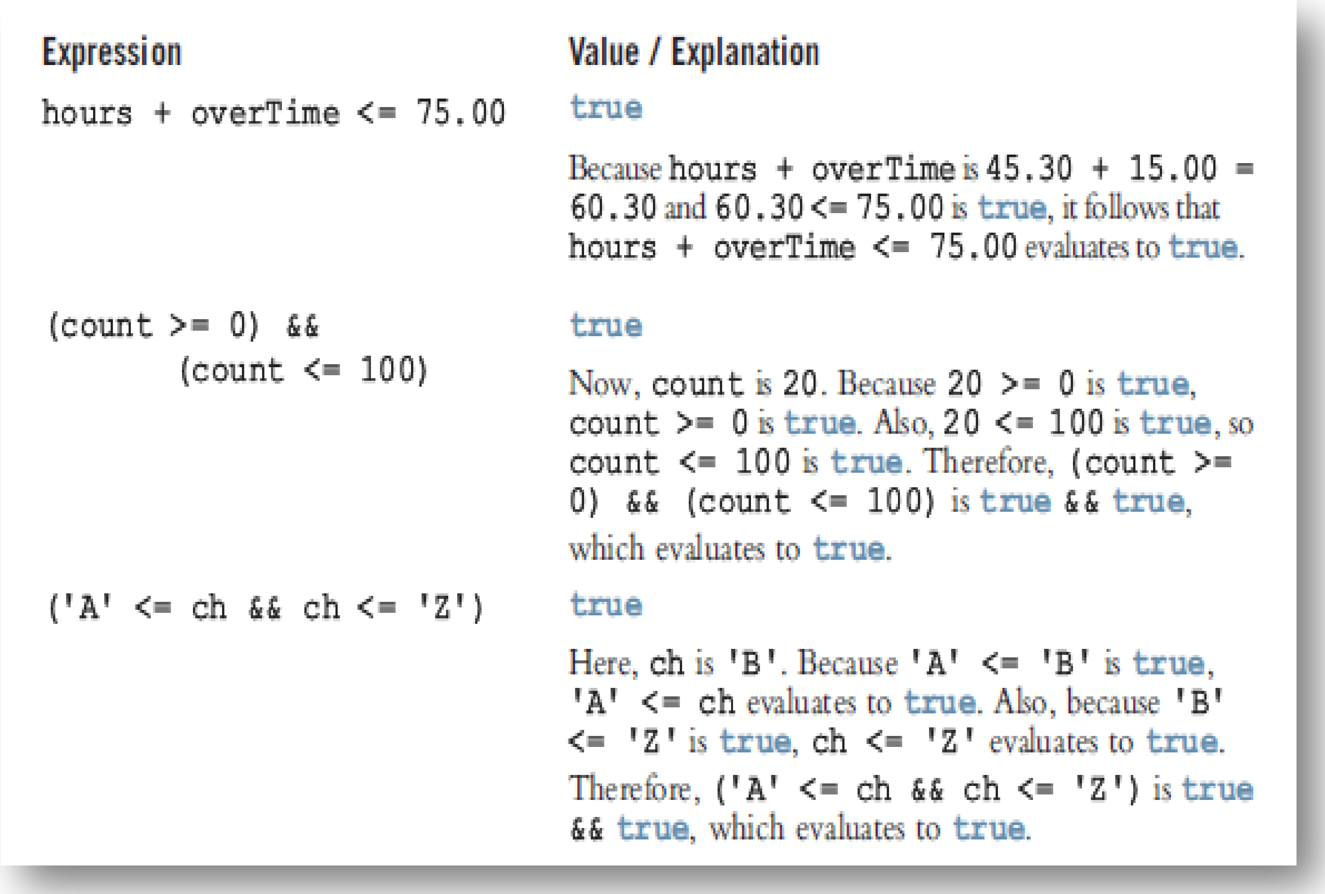
EXAMPLE 4-6

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';
```



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` is `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`.



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.

int Data Type and Logical (Boolean) Expressions

- Logical expressions evaluate to either 1 or 0
- You can use the int data type to manipulate 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