CS1002 – Programming Fundamentals

Lecture # 12
Monday, October 12, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus

Muhammad Yousaf

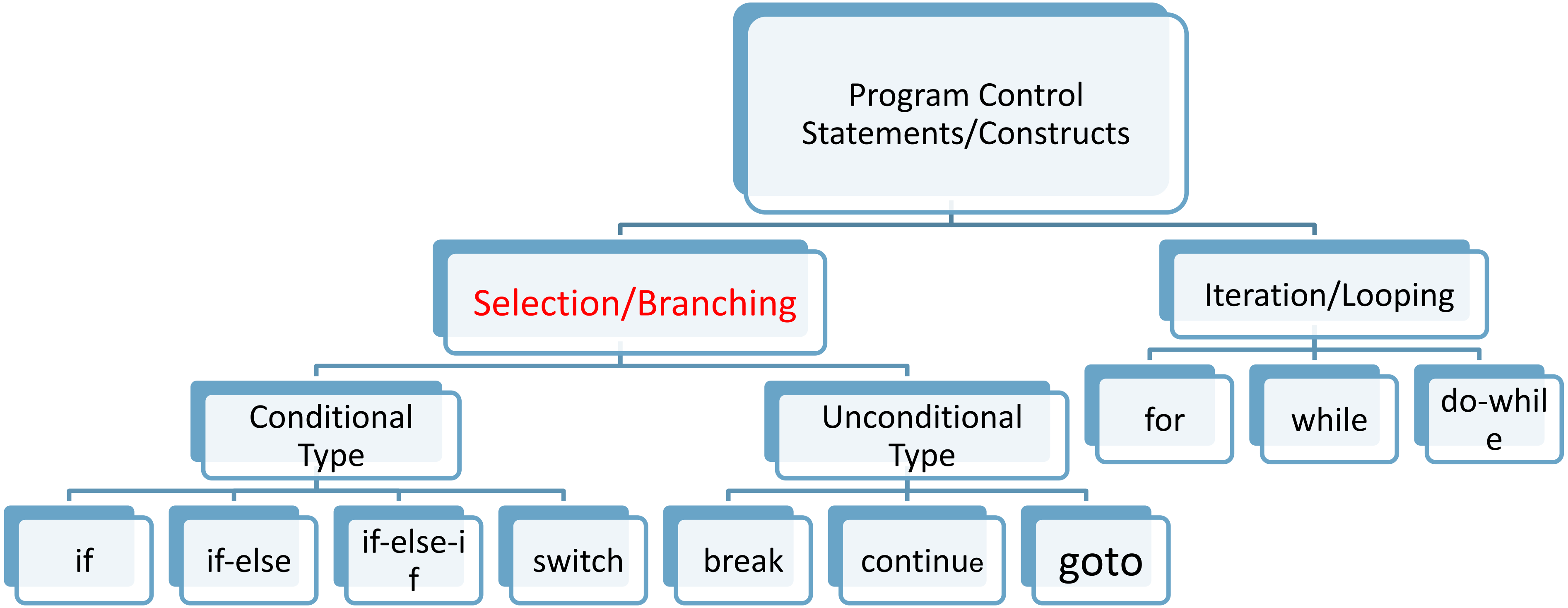
Outline

- Control Structures
    - Selection (if-else)
    - Repetition (Loops)
- Selection
    - One-way selection
    - Two-way selection
    - Compound selection
- Relational, Logical and Bitwise operators
- Switch statement
- Assert Function

2. CONTROL STATEMENTS INCLUDE

Selection Statements	Iteration Statements	Jump Statements
•• if	•• for	•• goto
•• if-else	•• while	•• break
•• switch	•• do-while	•• continue
		•• return

PROGRAM CONTROL STATEMENTS/CONSTRUCTS IN ‘C/C++’



Program Control Statements/Constructs

Selection/Branching

Iteration/Looping

Conditional Type

Unconditional Type

for

while

do-while

if

if-else

if-else-if

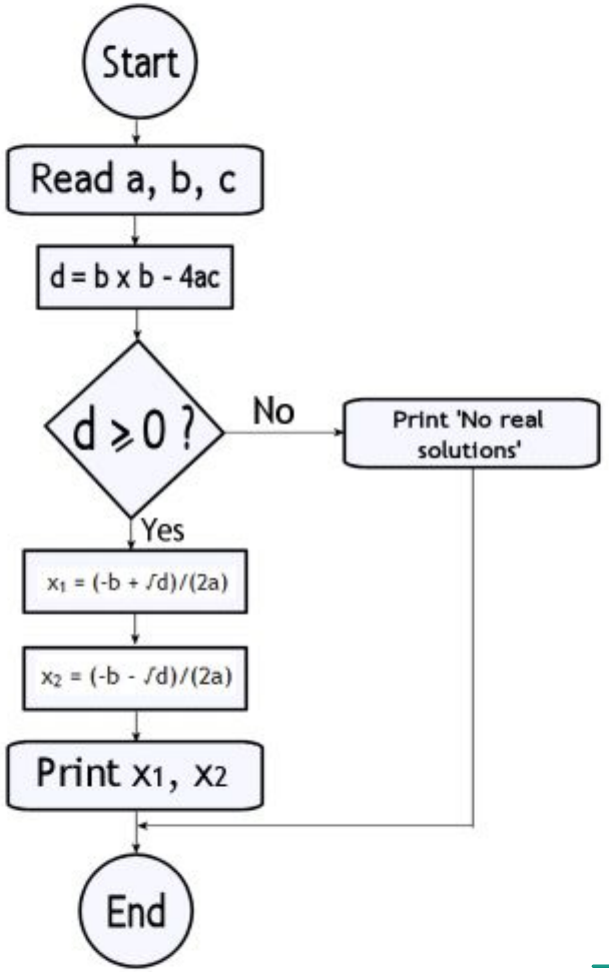
switch

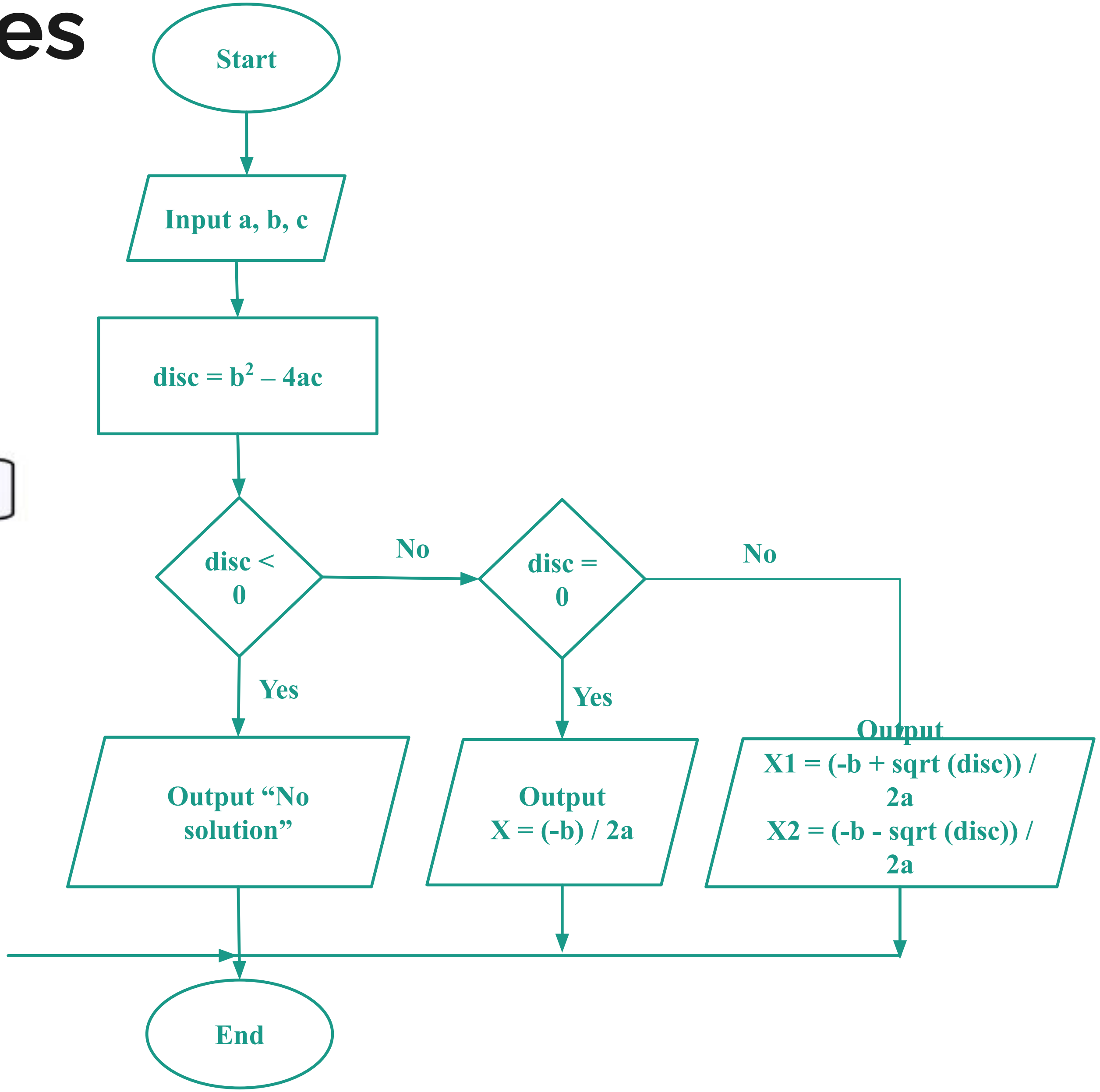
break

continue

goto

Control Structures





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

Control Structures (cont'd.)

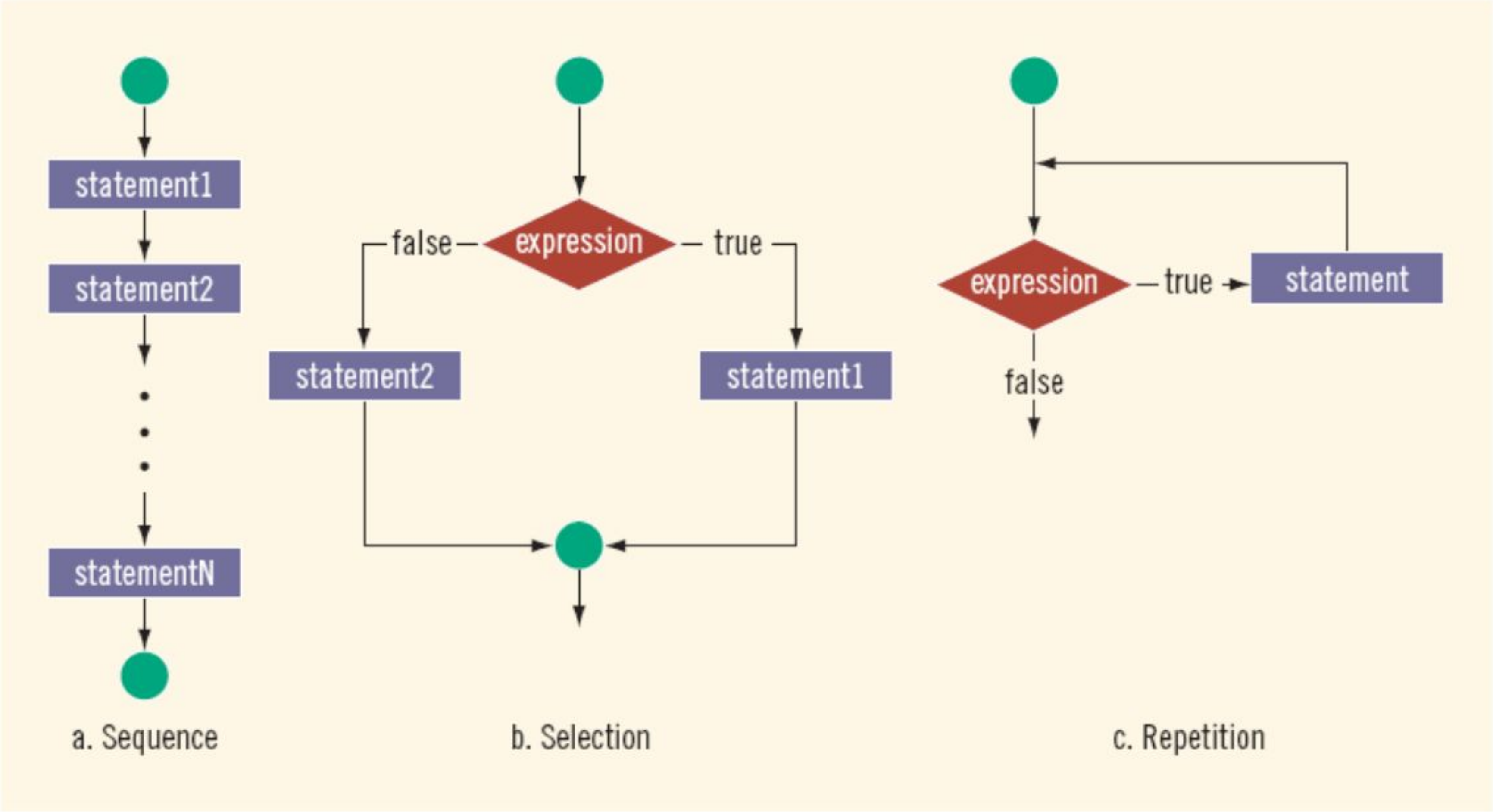
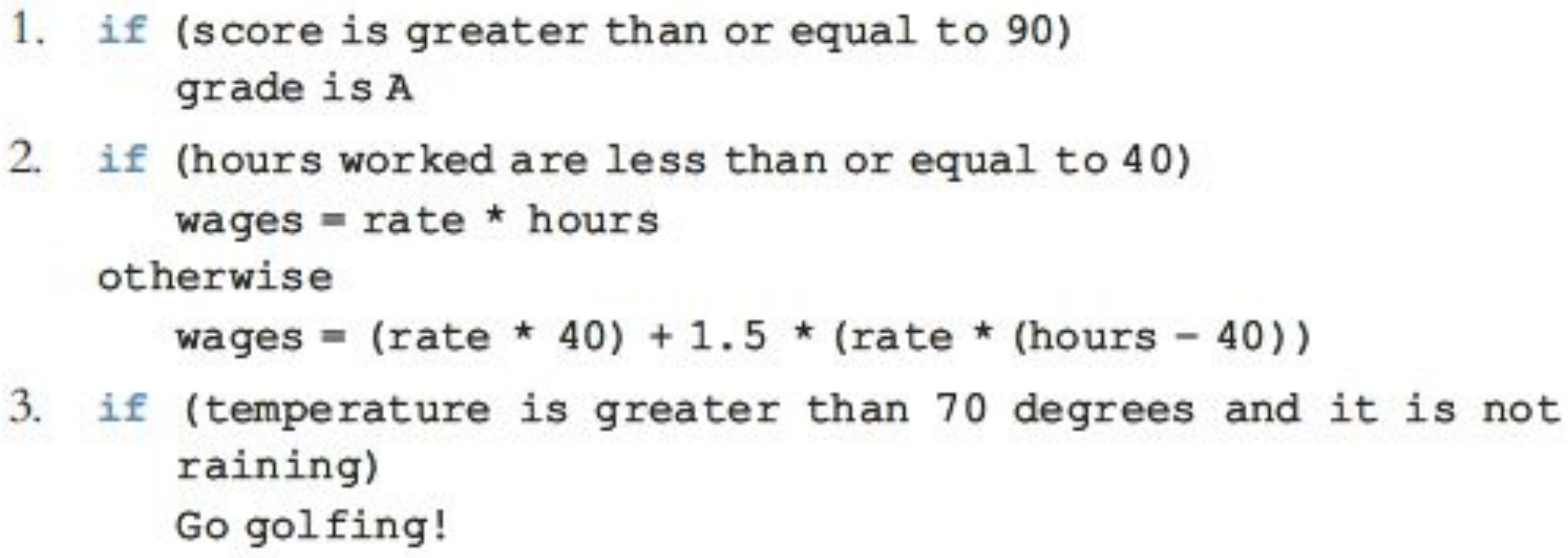


FIGURE 4-1 Flow of execution

Control Structures (cont'd.)

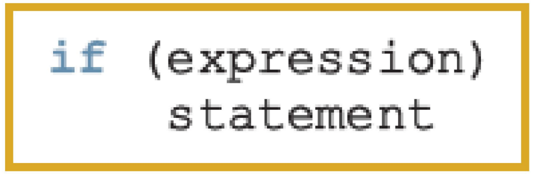
- You must understand the nature of conditional statements and how to use them.
- Consider the following three statements:



```
1.  if (score is greater than or equal to 90)
        grade is A
2.  if (hours worked are less than or equal to 40)
        wages = rate * hours
    otherwise
        wages = (rate * 40) + 1.5 * (rate * (hours - 40))
3.  if (temperature is greater than 70 degrees and it is not
        raining)
        Go golfing!
```

One-Way Selection

- The syntax of one-way selection is:



```
if (expression)
    statement
```

- The statement is executed if the value of the expression is **true**
- The statement is bypassed if the value is **false**; program goes to the next statement
- **if** is a reserved word

One-Way Selection (cont'd.)

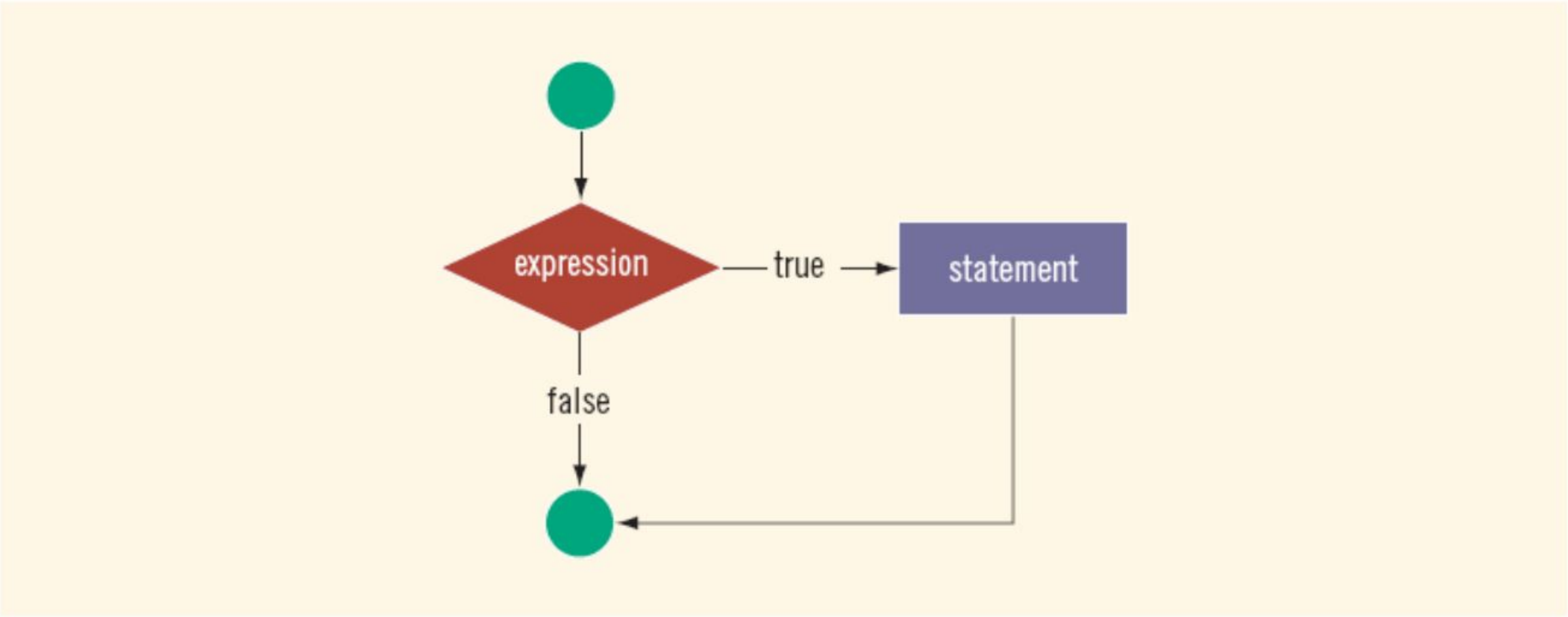
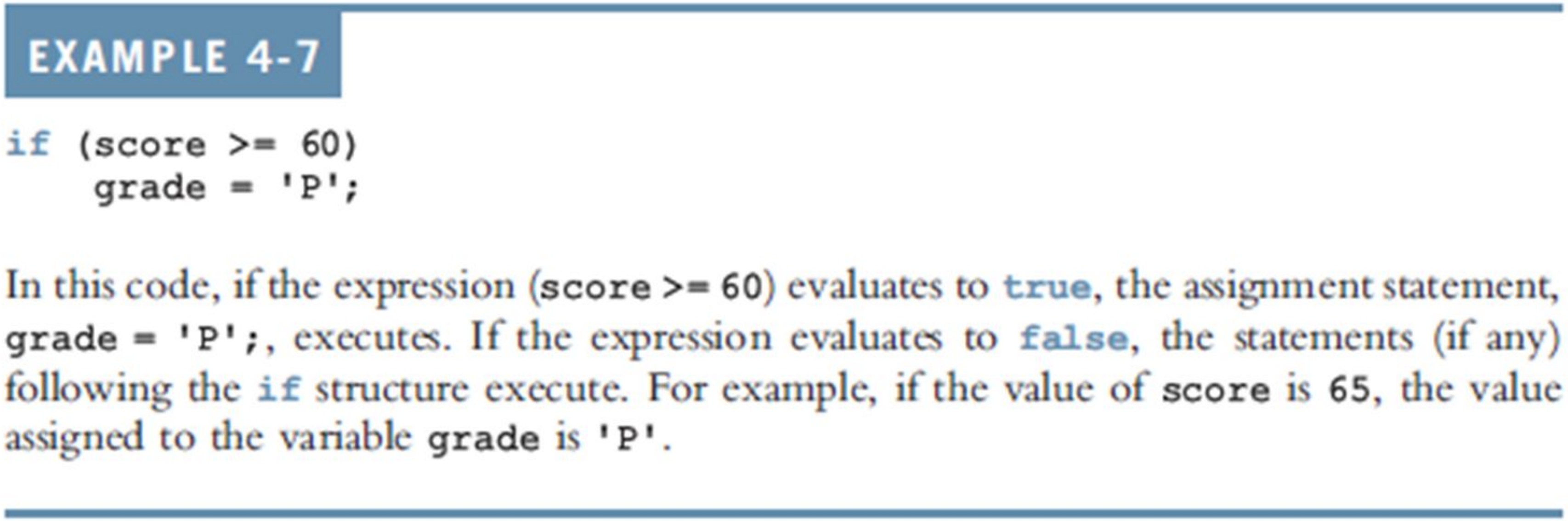


FIGURE 4-2 One-way selection

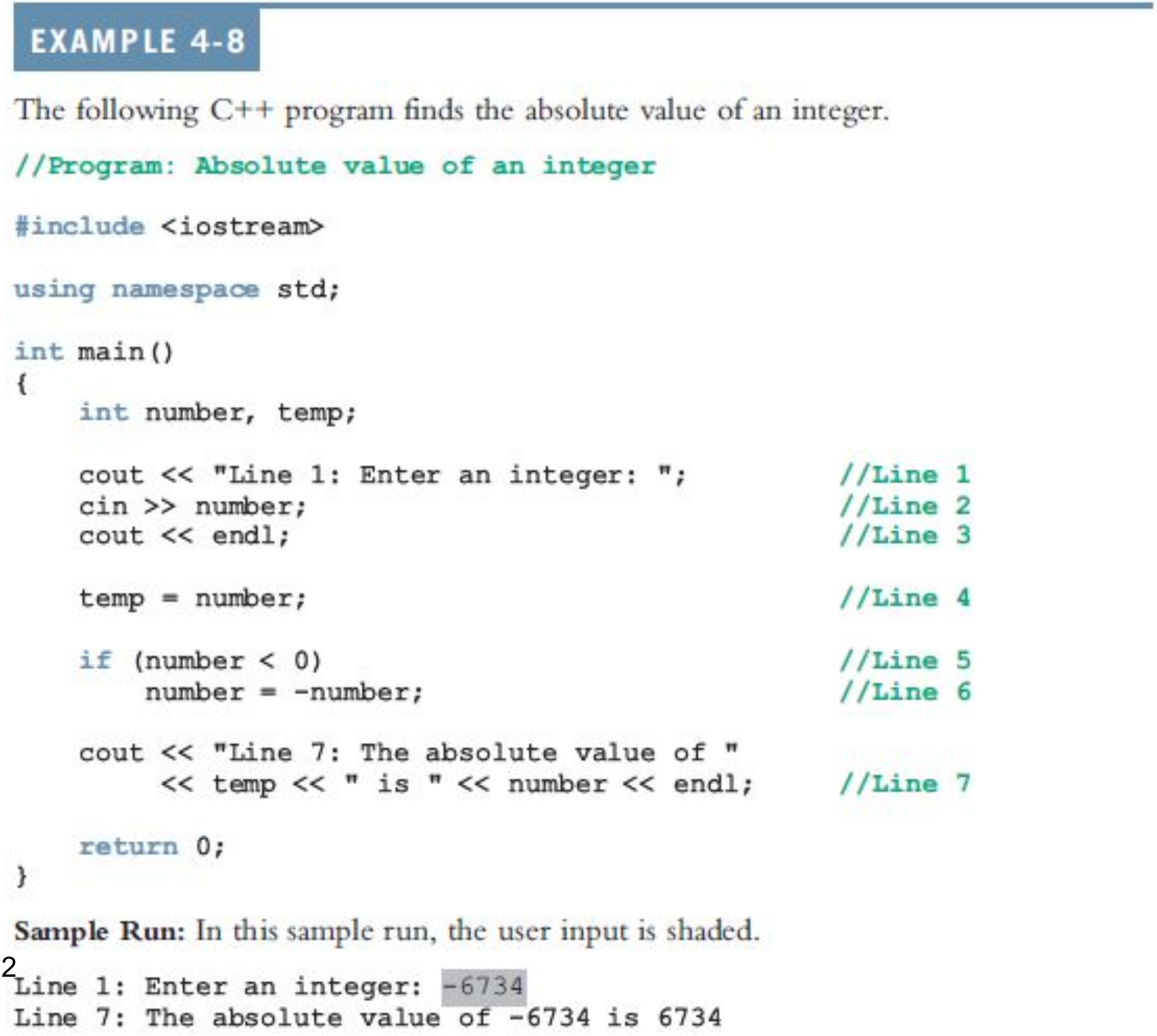
One-Way Selection (cont'd.)



EXAMPLE 4-7

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

In this code, if the expression (`score >= 60`) evaluates to `true`, the assignment statement, `grade = 'P';`, executes. If the expression evaluates to `false`, the statements (if any) following the `if` structure execute. For example, if the value of `score` is `65`, the value assigned to the variable `grade` is `'P'`.



EXAMPLE 4-8

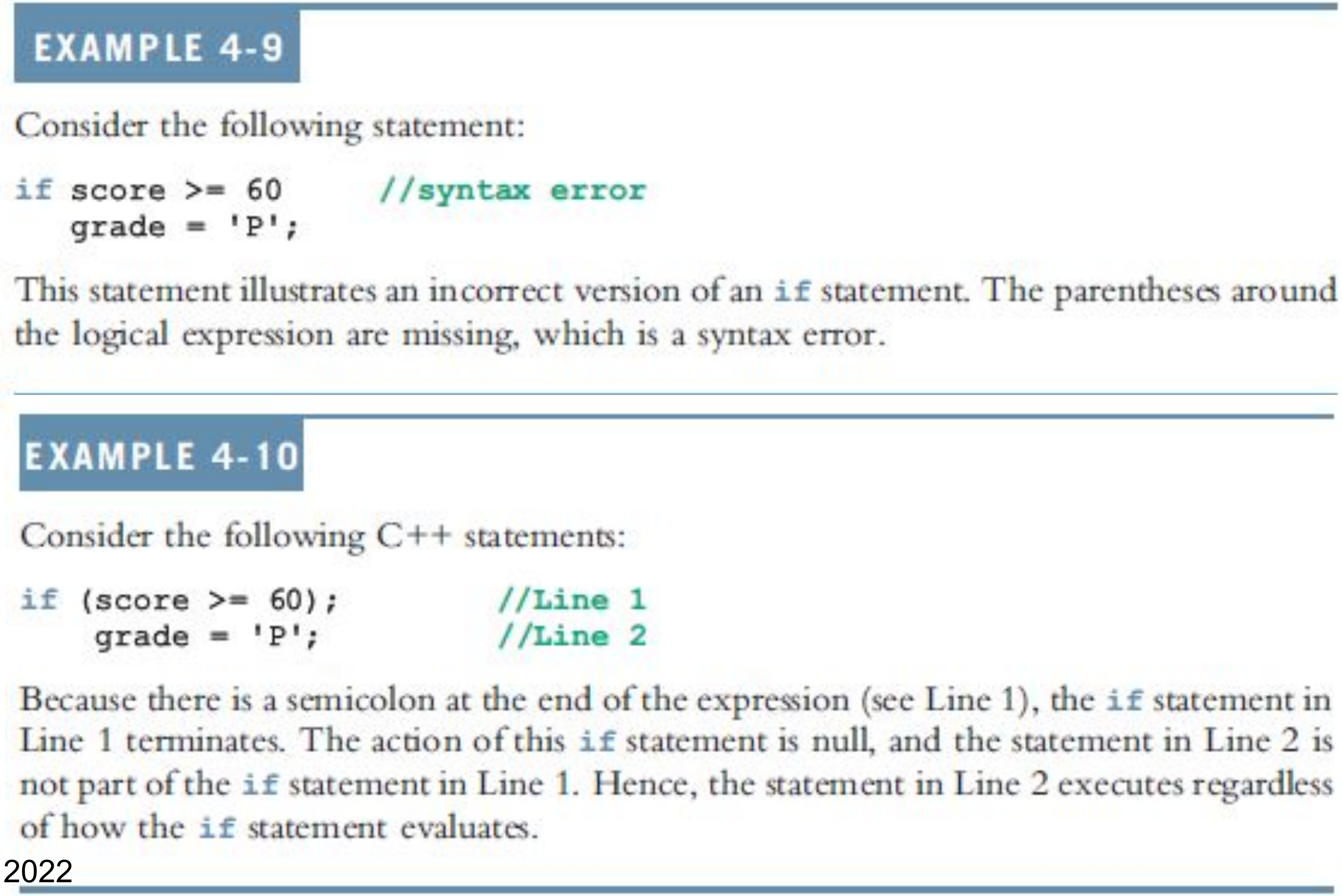
The following C++ program finds the absolute value of an integer.

```
//Program: Absolute value of an integer

#include <iostream>

using namespace std;

int main()
{
    int number, temp;

    cout << "Line 1: Enter an integer: ";         //Line 1
    cin >> number;                                //Line 2
    cout << endl;                                 //Line 3

    temp = number;                                //Line 4

    if (number < 0)                               //Line 5
        number = -number;                         //Line 6

    cout << "Line 7: The absolute value of "
         << temp << " is " << number << endl;     //Line 7

    return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

```
Line 1: Enter an integer: -6734
Line 7: The absolute value of -6734 is 6734
```

One-Way Selection (cont'd.)



EXAMPLE 4-9

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.

EXAMPLE 4-10

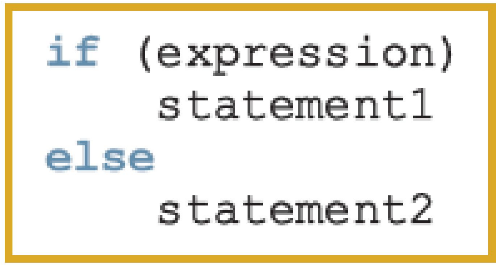
Consider the following C++ statements:

```
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.

Two-Way Selection

- Two-way selection takes the form:



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

- If expression is true, statement1 is executed; otherwise, statement2 is executed
- statement1 and statement2 are any C++ statements
- **else** is a reserved word

Two-Way Selection (cont'd.)

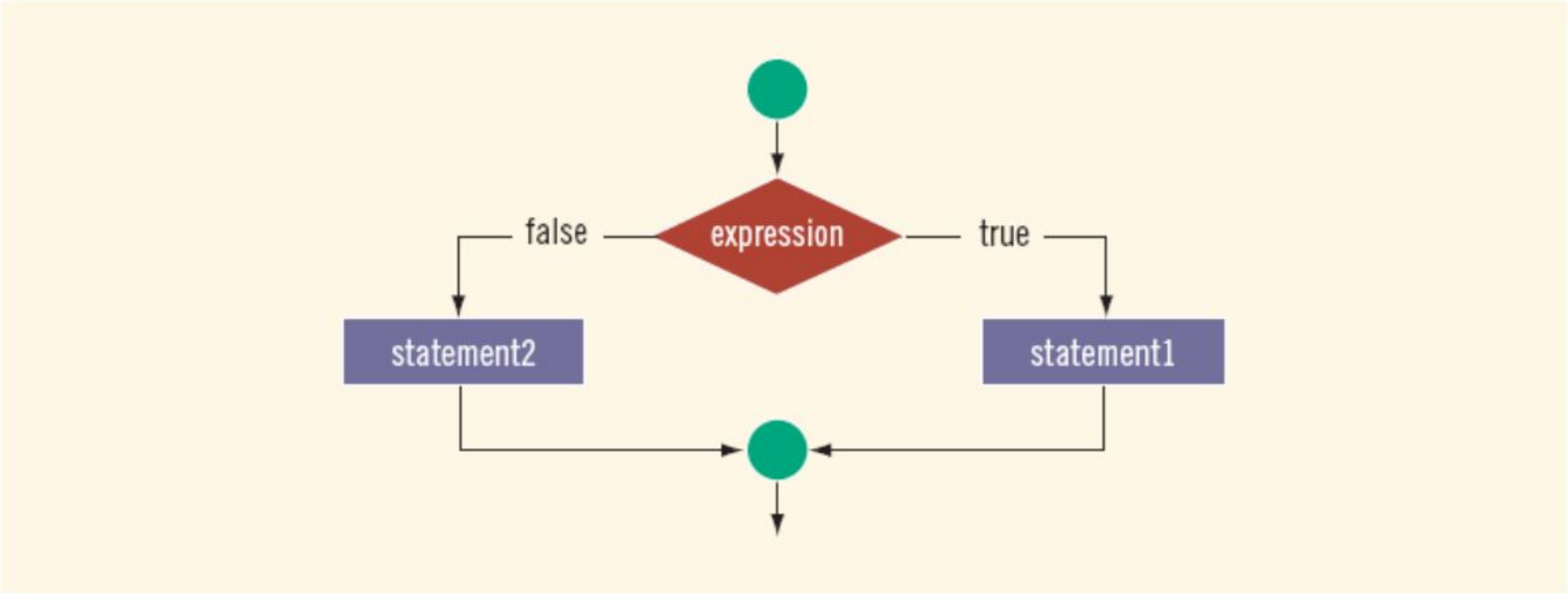
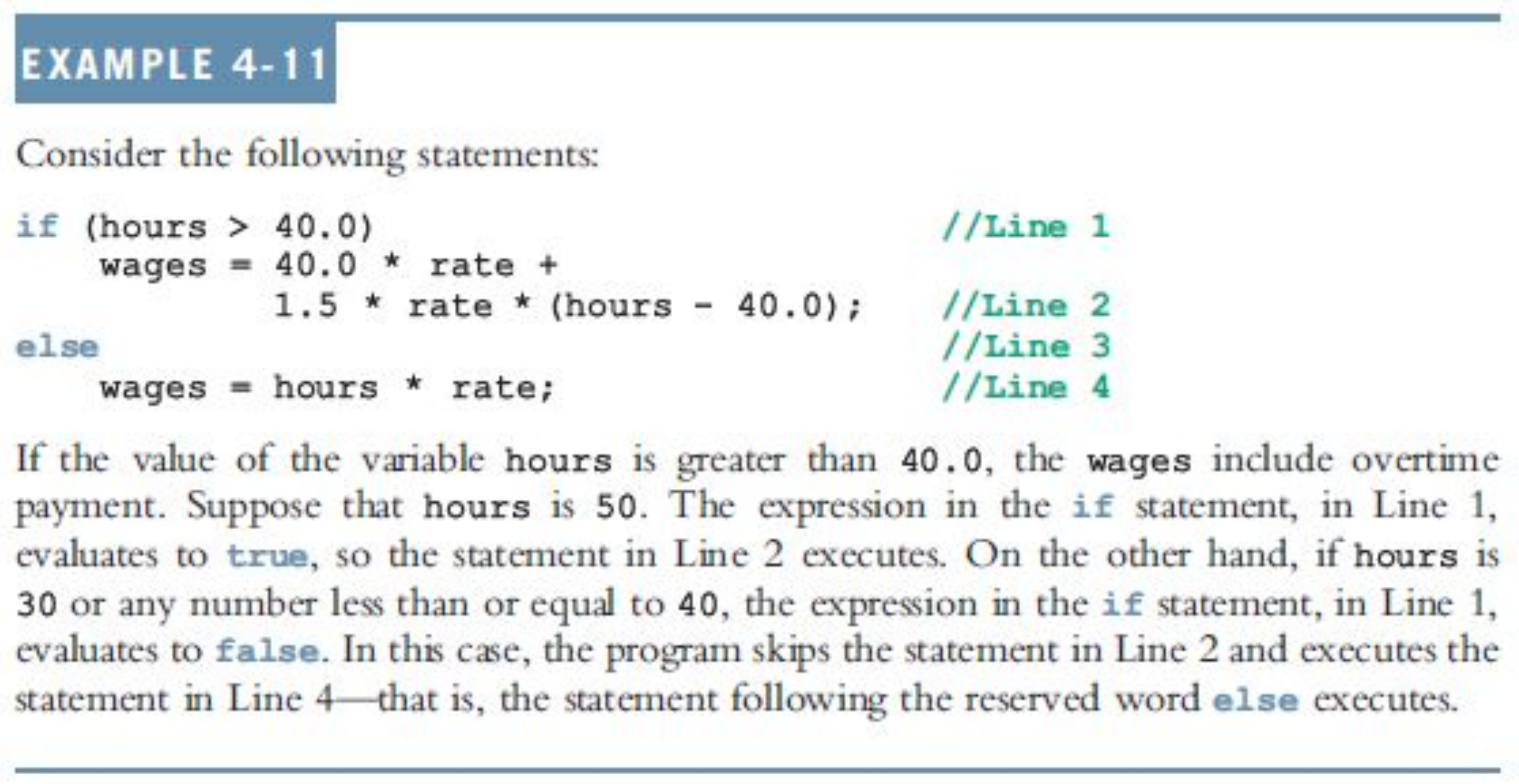


FIGURE 4-3 Two-way selection

Two-Way Selection (cont'd.)



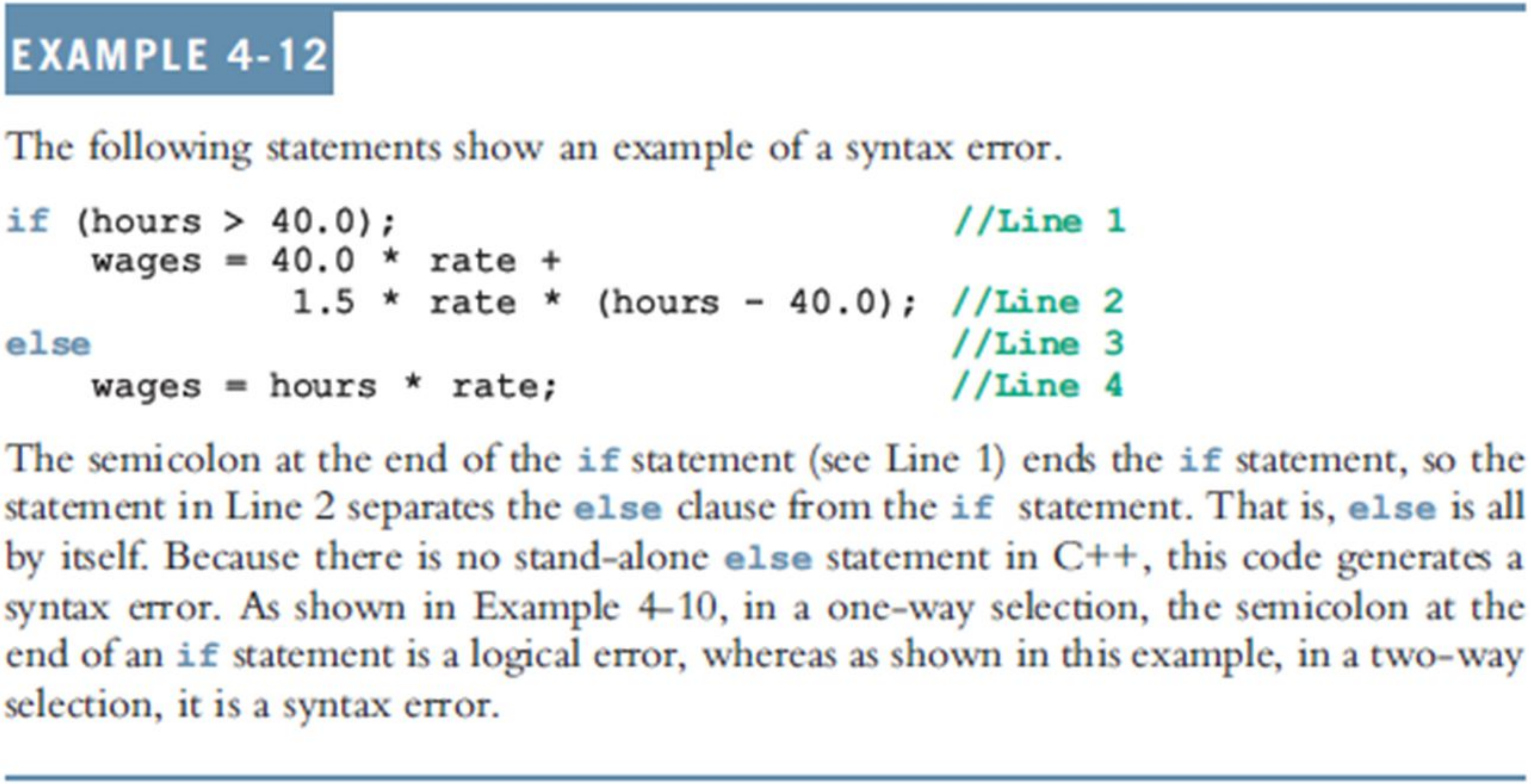
EXAMPLE 4-11

Consider the following statements:

```
if (hours > 40.0)                                //Line 1
    wages = 40.0 * rate +
            1.5 * rate * (hours - 40.0);         //Line 2
else                                             //Line 3
    wages = hours * rate;                        //Line 4
```

If the value of the variable hours is greater than 40.0, the wages include overtime payment. Suppose that hours is 50. The expression in the if statement, in Line 1, evaluates to true, so the statement in Line 2 executes. On the other hand, if hours is 30 or any number less than or equal to 40, the expression in the if statement, in Line 1, evaluates to false. In this case, the program skips the statement in Line 2 and executes the statement in Line 4—that is, the statement following the reserved word else executes.

Two-Way Selection (cont'd.)



EXAMPLE 4-12

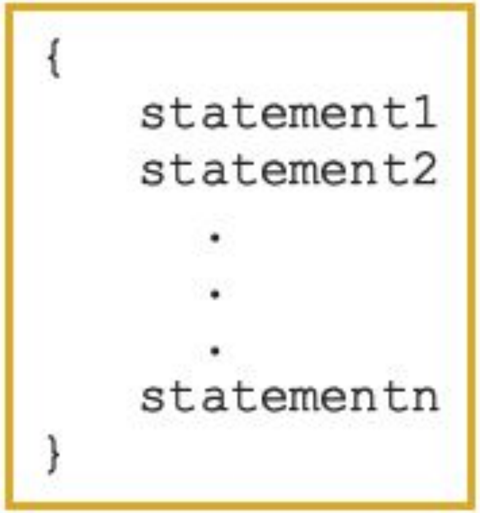
The following statements show an example of a syntax error.

```
if (hours > 40.0);                                //Line 1
    wages = 40.0 * rate +
             1.5 * rate * (hours - 40.0);        //Line 2
else                                             //Line 3
    wages = hours * rate;                        //Line 4
```

The semicolon at the end of the `if` statement (see Line 1) ends the `if` statement, so the statement in Line 2 separates the `else` clause from the `if` statement. That is, `else` is all by itself. Because there is no stand-alone `else` statement in C++, this code generates a syntax error. As shown in Example 4-10, in a one-way selection, the semicolon at the end of an `if` statement is a logical error, whereas as shown in this example, in a two-way selection, it is a syntax error.

Compound (Block of) Statements

- Compound statement (block of statements):



```
{
    statement1
    statement2
       .
       .
       .
    statementn
}
```

- A compound statement is a single statement

Compound (Block of) Statements (cont'd.)

```
if (age > 18)
{
    cout << "Eligible to vote." << endl;
    cout << "No longer a minor." << endl;
}
else
{
    cout << "Not eligible to vote." << endl;
    cout << "Still a minor." << endl;
}
```

Decision Making: Equality and Relational Operators

if structure

- Decision based on truth or false of condition
  - If condition met, body executed
  - Else, body not executed

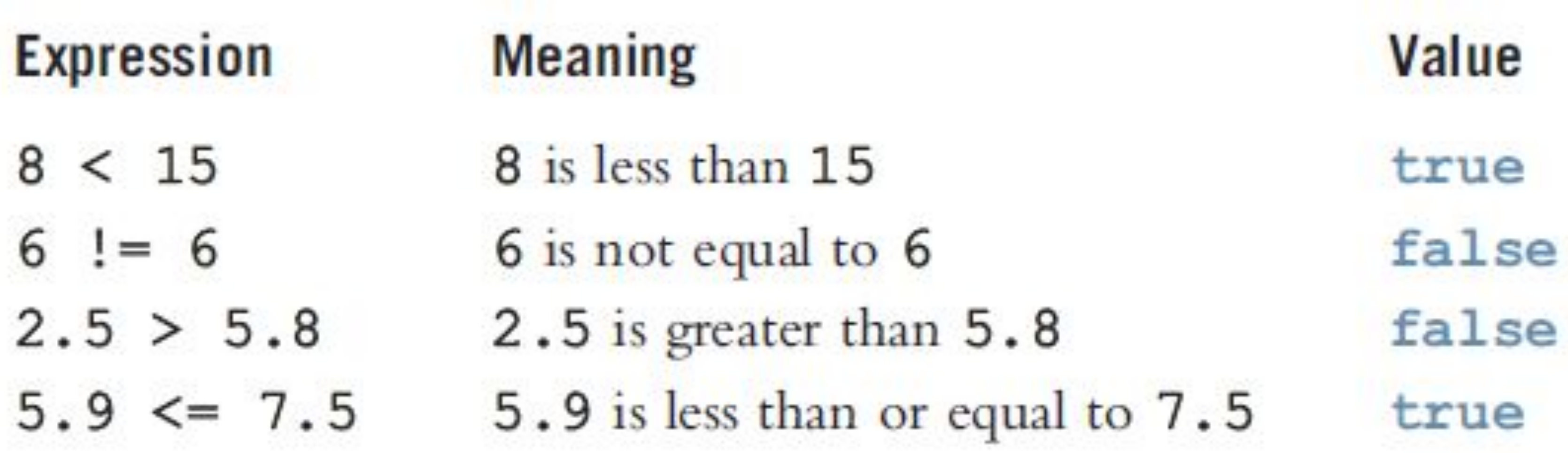
Equality and relational operators

- Equality operators
  - Same level of precedence
- Relational operators
  - Same level of precedence
- Associate left to right

Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Relational Operators and Simple Data Types



Expression	Meaning	Value
`8 < 15`	8 is less than 15	`true`
`6 != 6`	6 is not equal to 6	`false`
`2.5 > 5.8`	2.5 is greater than 5.8	`false`
`5.9 <= 7.5`	5.9 is less than or equal to 7.5	`true`

Comparing Characters

- Expression with relational operators
  - 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

Comparison of Characters

- For characters
  - Respective ASCIII values are compared
- 'R' > 'T' is false (82 > 84)
- '+' < '*' is false (43 < 42)
- 'A' <= 'a' is true (65<97)

Relational and Equality Operators (cont.)

- The relational operators have very low precedence and associate left-to-right
- The equality operators have very-very low precedence and associate left-to-right
- Some examples:

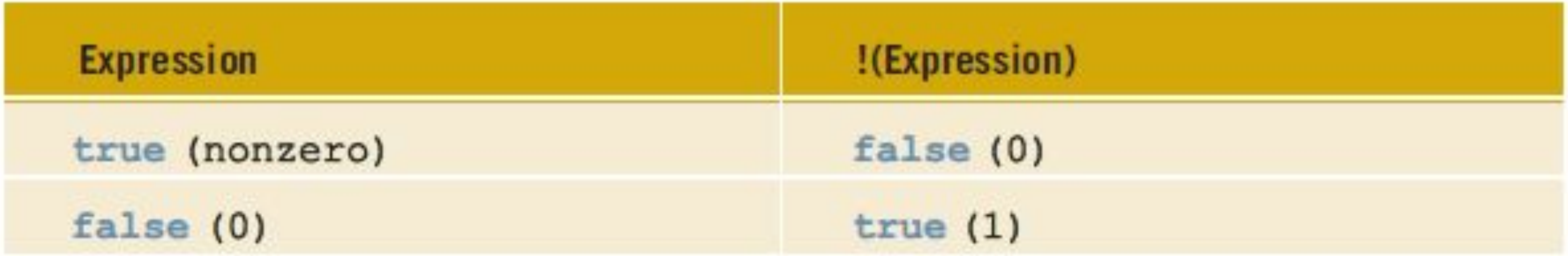
```
17 < x              foo == 3.14
 age != 21      x+1 >= 4*y-z
```

Precedence

Operators	Precedence
()	highest (applied first)
* / %	
+ -	
< <= > >=	
== !=	
=	
	lowest (applied last)

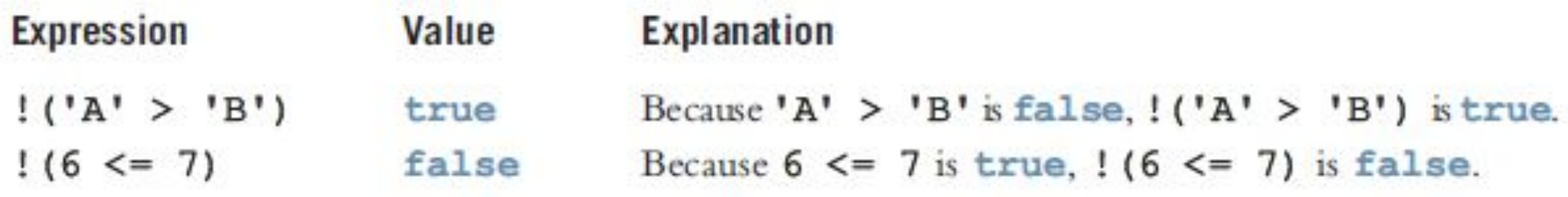
Logical (Boolean) Operators and Logical Expressions

Operator	Description
!	NOT
&&	AND
||	OR



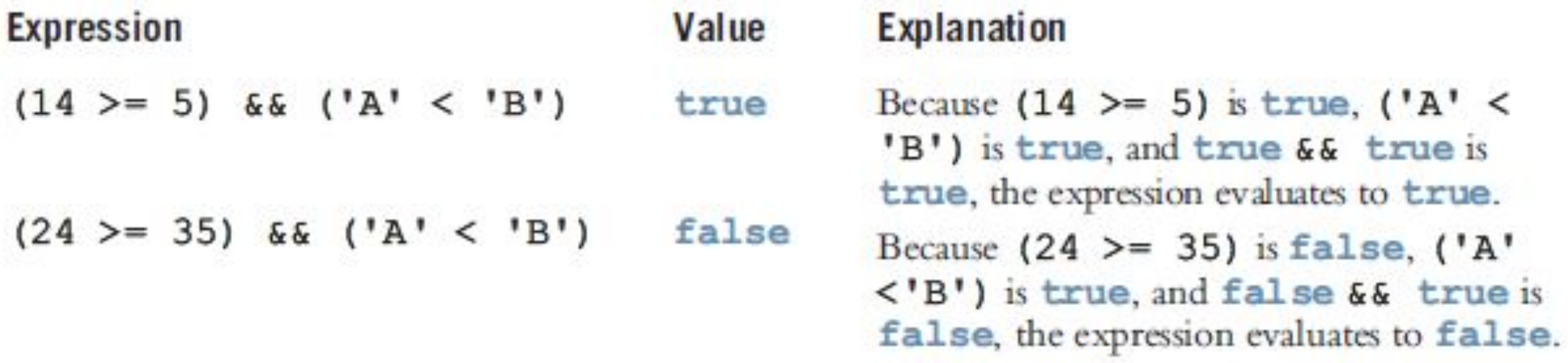
Expression	!(Expression)
`true (nonzero)`	`false (0)`
`false (0)`	`true (1)`

Conti...



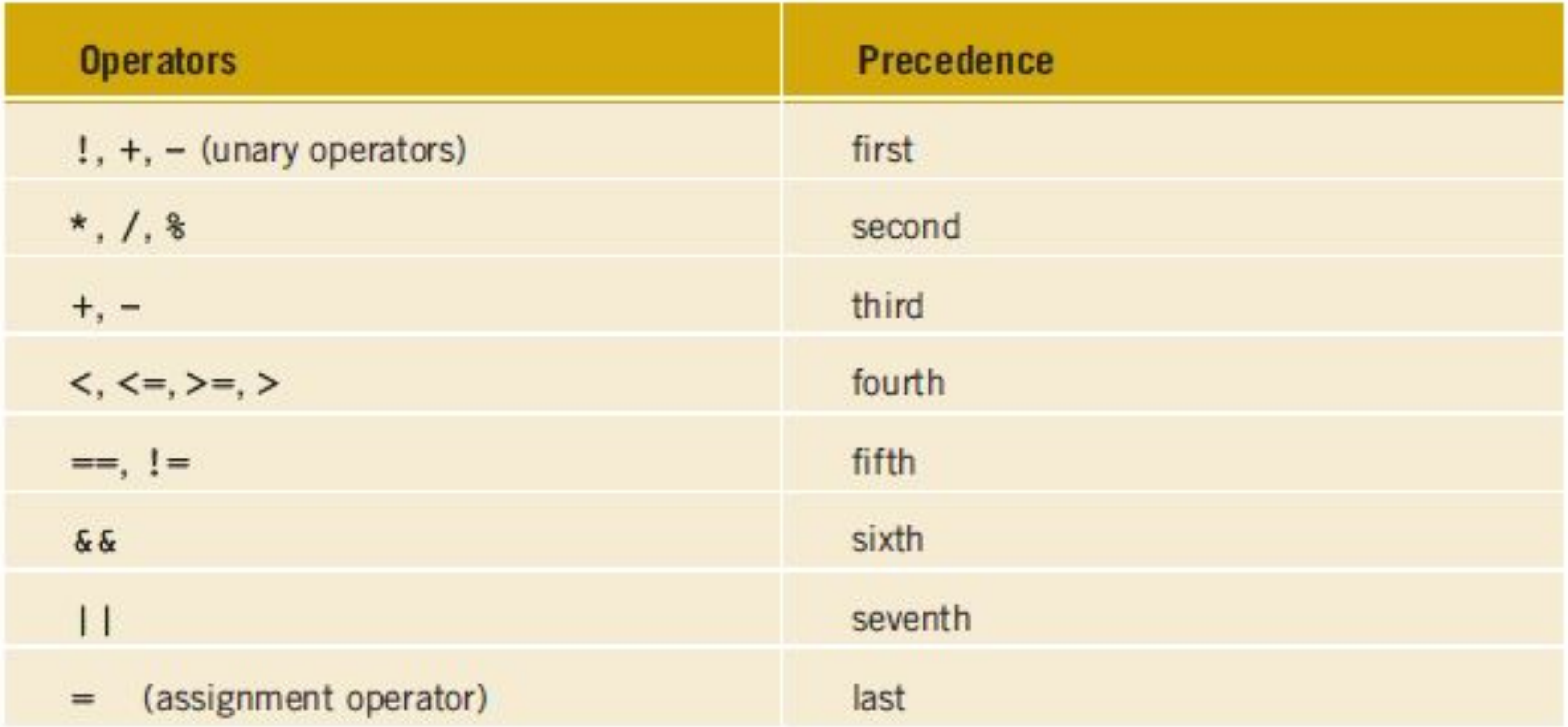
Expression	Value	Explanation
!('A' > 'B')	true	Because 'A' > 'B' is false, !('A' > 'B') is true.
!(6 <= 7)	false	Because 6 <= 7 is true, !(6 <= 7) is false.

- AND & OR operators work just like AND/OR-Gate as you studied in Physics in intermediate
- AND = True iff all conditions are TRUE
- OR = False iff all results are FASLE



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.

Order of precedence



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

Associativity for same level of precedence

Operators	Associativity
() ++ (postfix) -- (postfix)	left to right
+ (unary) - (unary)	right to left
++ (prefix) -- (prefix) * / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
?:	right to left
= + = - = * = / =	right to left
, (comma operator)	left to right

- 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 / Expression
!found	**false** Because found is **true**, !found is **false**
hours > 40.0	**true** Because hours is 45.3 and 45.3 > 40.0 is **true**, the expression hours > 40.0 evaluates to **true**
!age	**false** Age is 20, which is non zero 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**

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

Expression	Value / Expression
hours + overTime <= 75.0	**true** hours + overTime is 45.30 + 15.00 = 60.30 and 60.30 <= 75.0 is true, it follows that hours + overtime <= 75 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**, count <=100 is **true**. Therefore (count >= 20) && (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** evaluates to **true**.

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

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

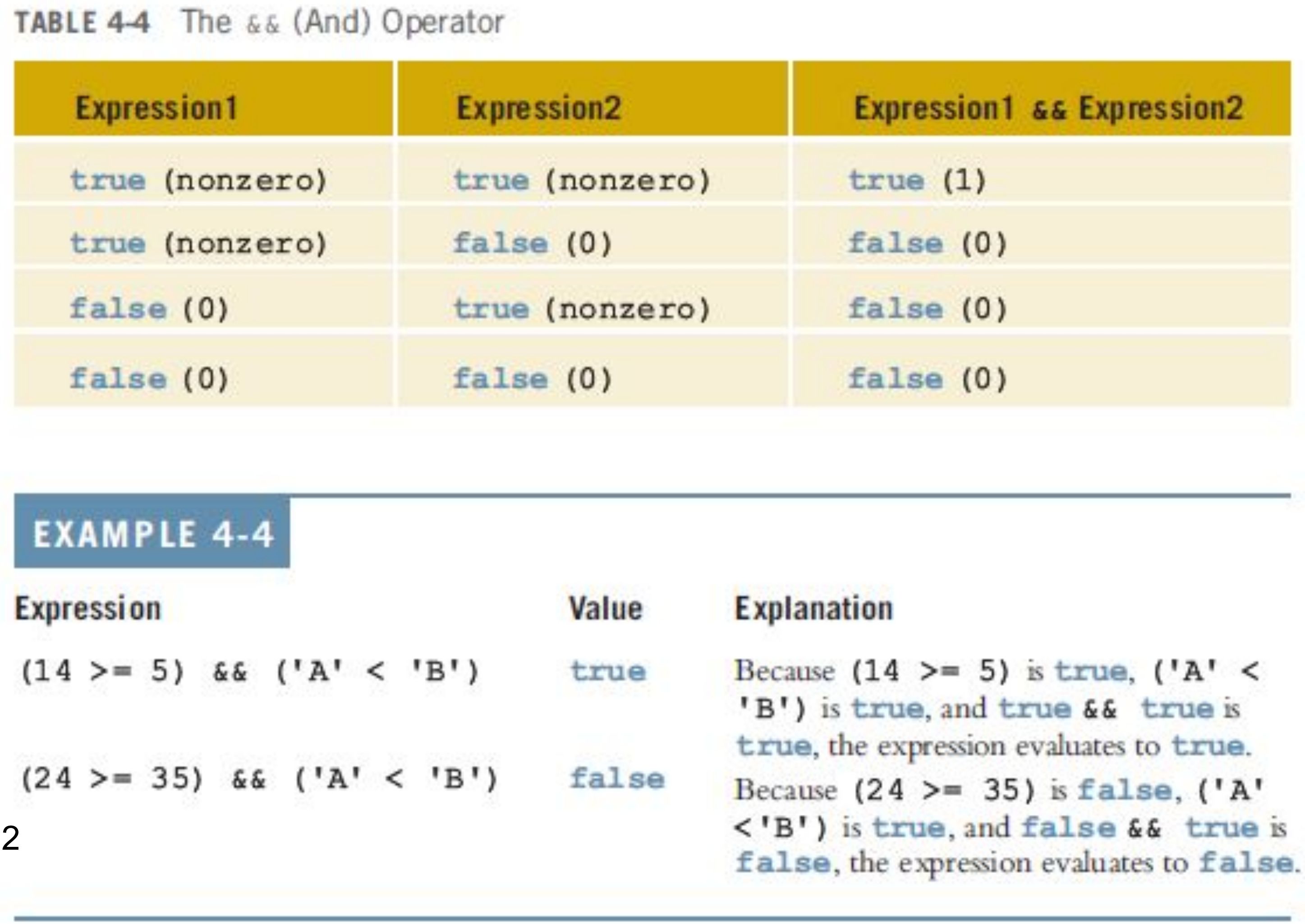


TABLE 4-4 The && (And) Operator

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 4-4

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.

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

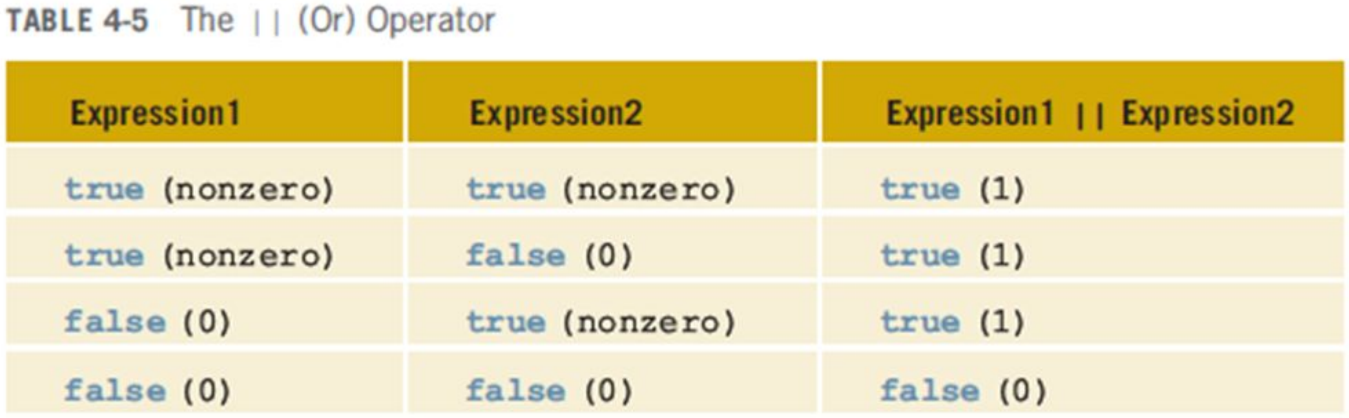
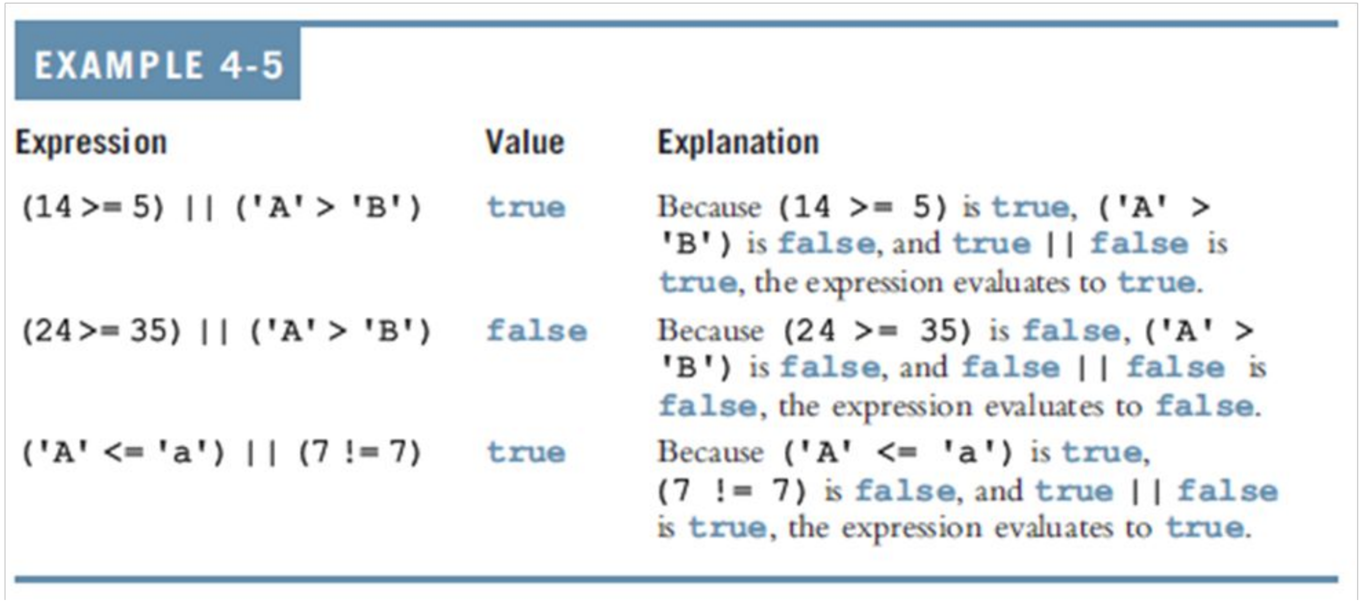


TABLE 4-5 The || (Or) Operator

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 4-5

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.

Conditional Operator (?:)

- Conditional operator (?:) takes three arguments
  - Ternary operator
- Syntax for using the conditional operator:

  expression1 ? expression2 : expression3

- If **expression1** is true, the result of the conditional expression is **expression2**
- Otherwise, the result is **expression3**

Conditional Operator (?:)

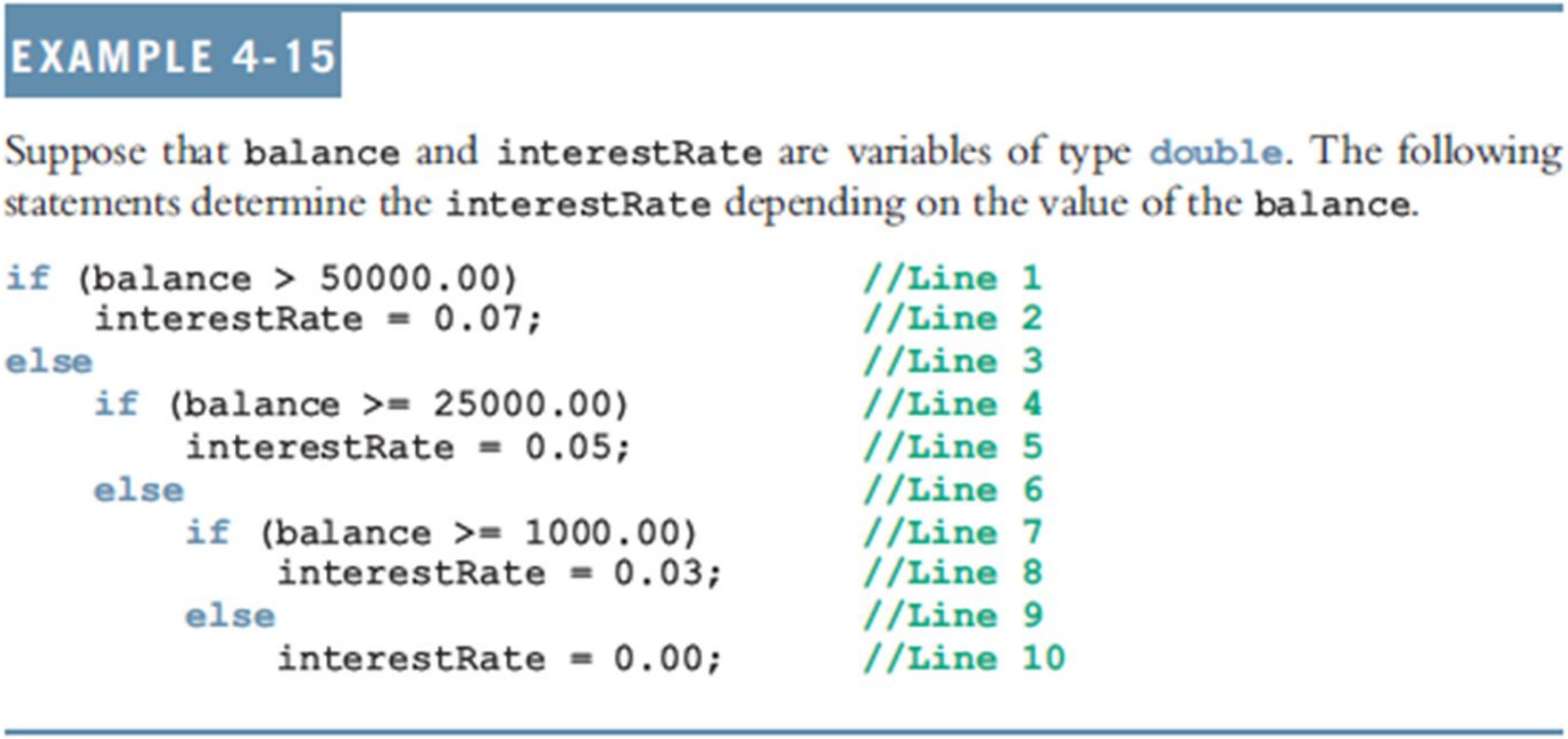
Consider the following statement

```
if(x >= y)
    large = x;
else
    large = y;
```

You can use the conditional operator to simplify the writing of this **if...else** statement as follows:

```
large = (x >= y)? x: y ;
```

Multiple Selections: Nested if (cont'd.)

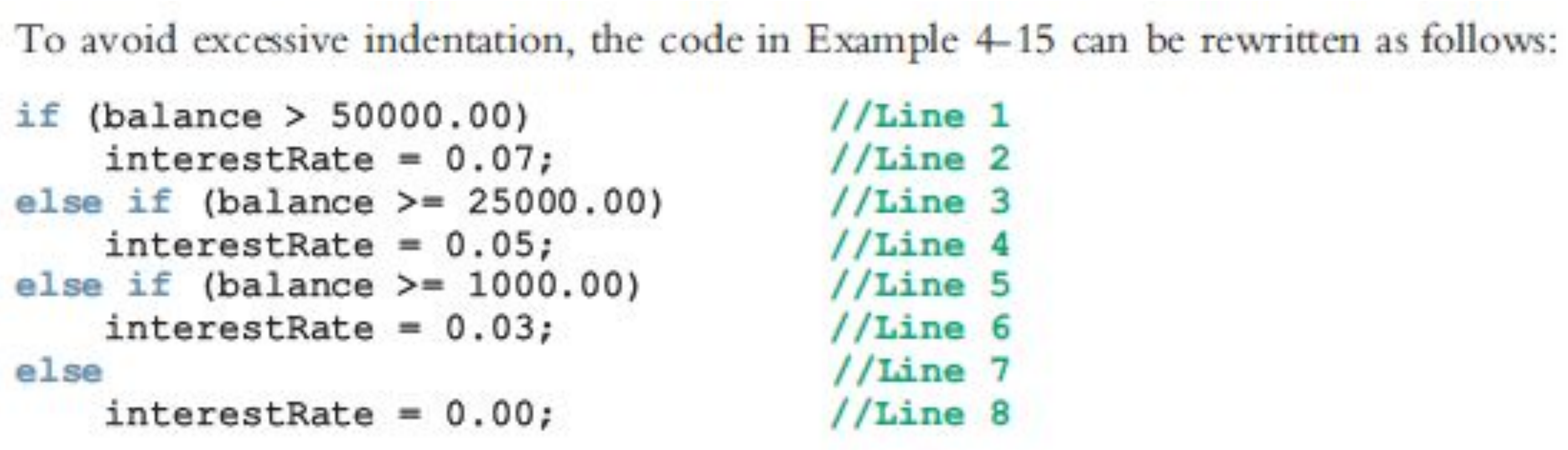


EXAMPLE 4-15

Suppose that balance and interestRate are variables of type double. The following statements determine the interestRate depending on the value of the balance.

```
if (balance > 50000.00)                    //Line 1
    interestRate = 0.07;                   //Line 2
else                                       //Line 3
    if (balance >= 25000.00)               //Line 4
        interestRate = 0.05;               //Line 5
    else                                   //Line 6
        if (balance >= 1000.00)            //Line 7
            interestRate = 0.03;           //Line 8
        else                               //Line 9
            interestRate = 0.00;           //Line 10
```

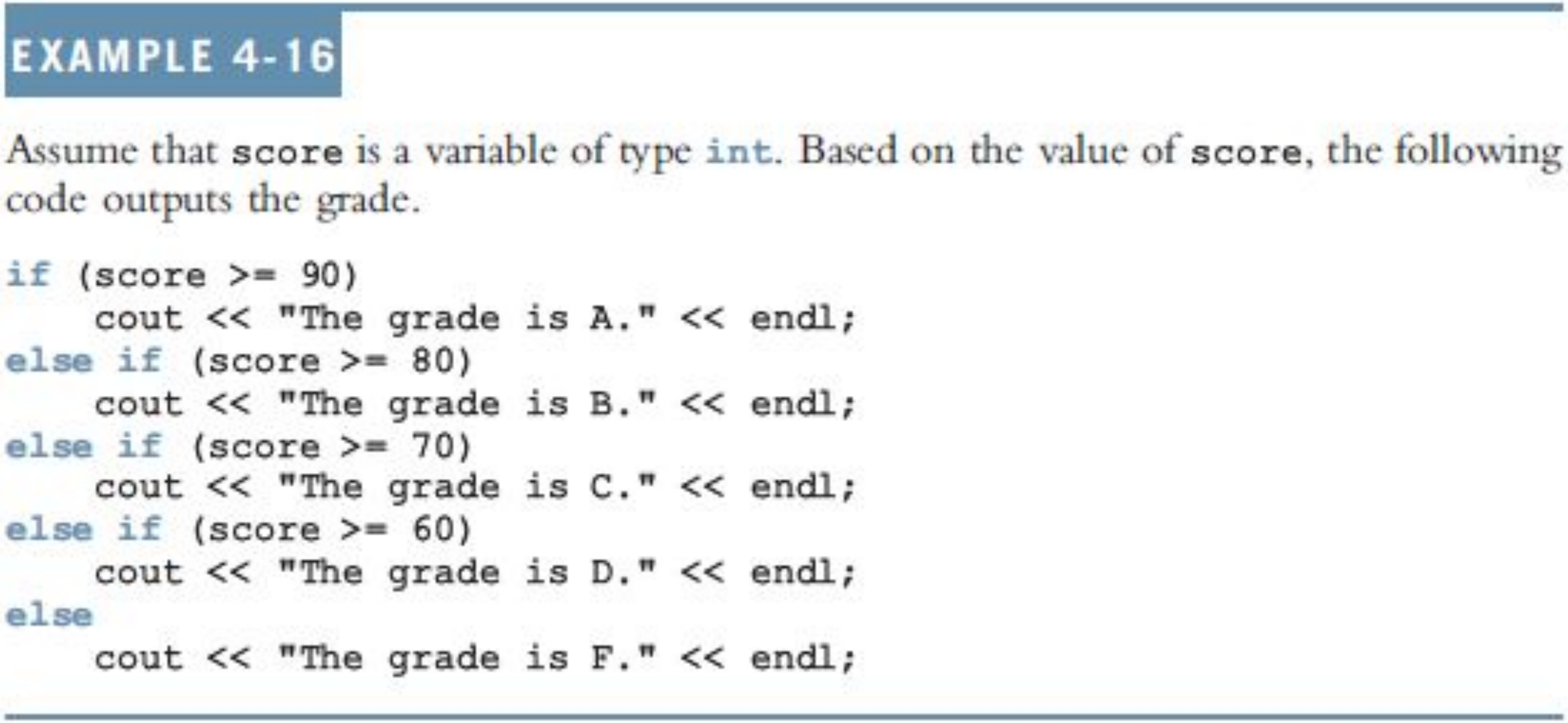
Multiple Selections: Nested if (cont'd.)



To avoid excessive indentation, the code in Example 4-15 can be rewritten as follows:

```
if (balance > 50000.00)                 //Line 1
    interestRate = 0.07;                //Line 2
else if (balance >= 25000.00)           //Line 3
    interestRate = 0.05;                //Line 4
else if (balance >= 1000.00)            //Line 5
    interestRate = 0.03;                //Line 6
else                                    //Line 7
    interestRate = 0.00;                //Line 8
```

Multiple Selections: Nested if (cont'd.)



EXAMPLE 4-16

Assume that score is a variable of type int. Based on the value of score, the following code outputs the grade.

```
if (score >= 90)
    cout << "The grade is A." << endl;
else if (score >= 80)
    cout << "The grade is B." << endl;
else if (score >= 70)
    cout << "The grade is C." << endl;
else if (score >= 60)
    cout << "The grade is D." << endl;
else
    cout << "The grade is F." << endl;
```

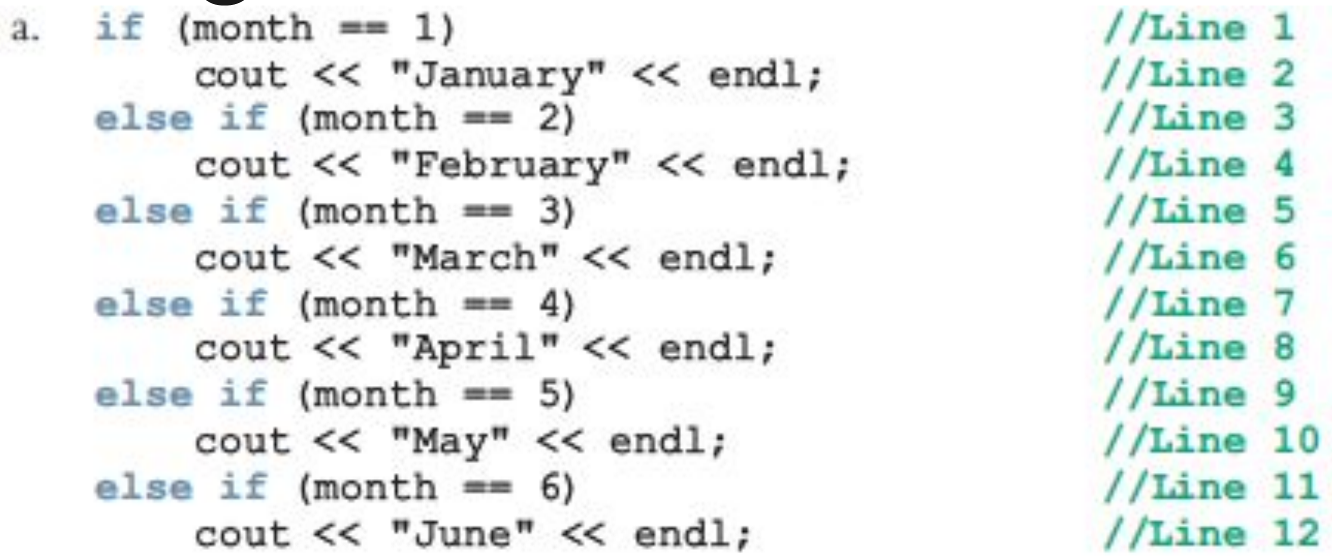
if-else **Pairing**

Assume that all the variables are properly declared and consider the following statements:

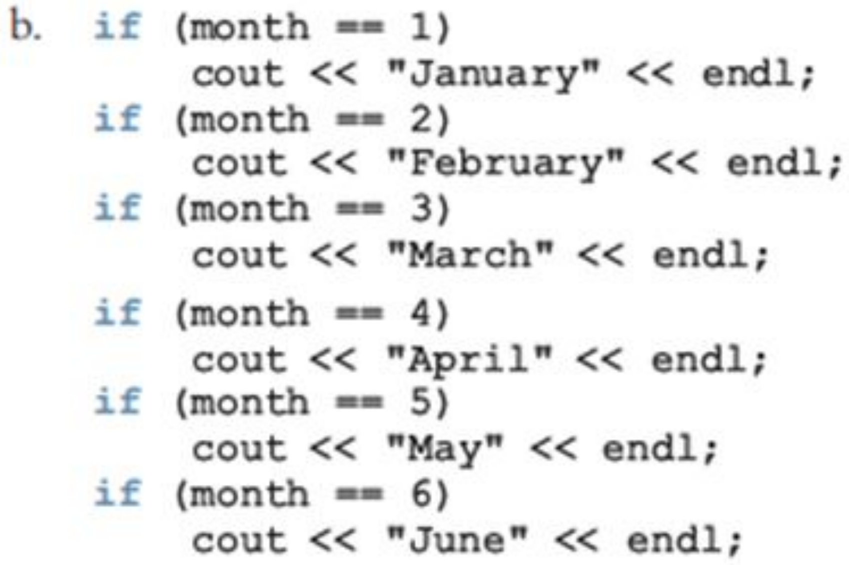
```
if(gender == 'M')                   //Line 1
    if(age < 21)                    //Line 2
        policyRate = 0.05;          //Line 3
    else                                //Line 4
        policyRate = 0.035;         //Line 5
else if (gender == 'F’)          //Line 6
    if(age < 21)                    //Line 7
        policyRate = 0.04;          //Line 8
else                                    //Line 9
        policyRate = 0.03;          //Line 10
```

In this code, the else in Line 4 is paired with the if in Line 2. Note that for the else in Line 4, the most recent incomplete if is the if in Line 2. The else in Line 6 is paired with the if in Line 1. The else in Line 9 is paired with the if in Line 7. Once again the indentation does not determine the pairing, but it communicates the pairing

Comparing if…else Statements with a Series of if State



a.
```
if (month == 1)                                  //Line 1
    cout << "January" << endl;                   //Line 2
else if (month == 2)                             //Line 3
    cout << "February" << endl;                  //Line 4
else if (month == 3)                             //Line 5
    cout << "March" << endl;                     //Line 6
else if (month == 4)                             //Line 7
    cout << "April" << endl;                     //Line 8
else if (month == 5)                             //Line 9
    cout << "May" << endl;                       //Line 10
else if (month == 6)                             //Line 11
    cout << "June" << endl;                      //Line 12
```



b.
```
if (month == 1)
    cout << "January" << endl;
if (month == 2)
    cout << "February" << endl;
if (month == 3)
    cout << "March" << endl;
if (month == 4)
    cout << "April" << endl;
if (month == 5)
    cout << "May" << endl;
if (month == 6)
    cout << "June" << endl;
```

Short-Circuit Evaluation

- **Short-circuit evaluation:** evaluation of a logical expression stops as soon as the value of the expression is known
- **Example:**

Assume x = 21, y=5, z = 3, ch = ‘B’

```
    (x >= 20) || ( y == 10) //Line 1
    (ch == 'A') && (z < 7)  //Line 2
```

Short-Circuit Analysis

```
int a=10;
int b = 5;
```

1. `(a > 5 && b == 5 && a > 10 && (b = 15));`
2. `((a = 5) || b == 5 && a > 10 && (b = 15));`
3. `a = 10, b = 5;`
4. `(a > 5 && b == 5 || a > 10 && (b = 15));`
5. `(a > 5 && b == 5 && a > 10 || (b = 15));`
6. `(a = 5 || b == 5 && a > 10 && (b = 5));`

Comparing Floating-Point Numbers for Equality: A Precaution

- Comparison of floating-point numbers for equality may not behave as you would expect
- **Example:**
  - 1.0 == 3.0/7.0 + 2.0/7.0 + 2.0/7.0 evaluates to false
  - Why? 3.0/7.0 + 2.0/7.0 + 2.0/7.0 = 0.99999999999999989
- **Solution:** use a tolerance value
  - Example: fabs(x – y) < 0.000001

```
#include<iostream>
#include <iomanip>
#include<cmath>

using namespace std;

int main()
{
    double x = 1.0;
    double y = 3.0 / 7.0 + 2.0 / 7.0 + 2.0 / 7.0;
    cout << fixed << showpoint << setprecision(17);
    cout << "3.0 / 7.0 + 2.0 / 7.0 + 2.0 / 7.0 = "
        << 3.0 / 7.0 + 2.0 / 7.0 + 2.0 / 7.0 << endl;
    cout << "x = " << x << endl << "y = " << y << endl;
    if (x == y)
        cout << "x and y are same" << endl;
    else
        cout << "x and y are not same" << endl;
    if (fabs(x - y)<0.000001)
        cout << "x and y are same within the tolerance 0.000001" << endl;
    else
        cout << "x and y are not same within the tolerance 0.000001" <<endl;
    return 0;
}
```

Sample Run:

```
3.0 / 7.0 + 2.0 / 7.0 + 2.0 / 7.0 = 0.99999999999999989
x = 1.00000000000000000
y = 0.99999999999999989
x and y are not the same.
x and y are the same within the tolerance 0.000001.
```

Associativity of Relational Operators: A Precaution

```
#include<iostream>
using namespace std;
int main()
{
    int x;
    cout << "Enter and integer =  ";
    cin >> x;
    cout << endl;

    if (0 <= x <= 10)
        cout << x << " is within 0 and 10" << endl;
    else
        cout << x << " is not within 0 and 10" << endl;
    return 0;
}
```

Associativity of Relational Operators: A Precaution (cont'd.)

- x = 7

0 <= x <= 10	= 0 <= 7 <= 10	
	= (0 <=7) <= 10	Because relationship operators are evaluated from left to right
	= 1 <= 10	Because 0<=7 is true, 0<=7 evaluates to 1
	= 1 (true)	

- x = 30

0 <= x <= 10	= 0 <= 30 <= 10	
	= (0 <=30) <= 10	Because relationship operators are evaluated from left to right
	= 1 <= 10	Because 0<=30 is true, 0<=30 evaluates to 1
	= 1 (true)	

Solution:
0 <= x && x <= 10

Input Failure and the if Statement

- If input stream enters a fail state
    - All subsequent input statements associated with that stream are **ignored**
    - Program continues to execute
    - May produce **erroneous results**
- Can use **if** statements to check status of input stream
- If stream enters the fail state, include instructions that **stop program execution**

```
if(cin)
```

Confusion Between the Equality (==) and Assignment (=) Operators

- C++ allows you to use any expression that can be evaluated to either true or false as an expression in the if statement:

```
 if (x = 5)
        cout << "The value is five." << endl;
```

- The appearance of = in place of == resembles a silent killer
  - It is not a syntax error
  - It is a logical error

Program Style and Form (Revisited): Indentation

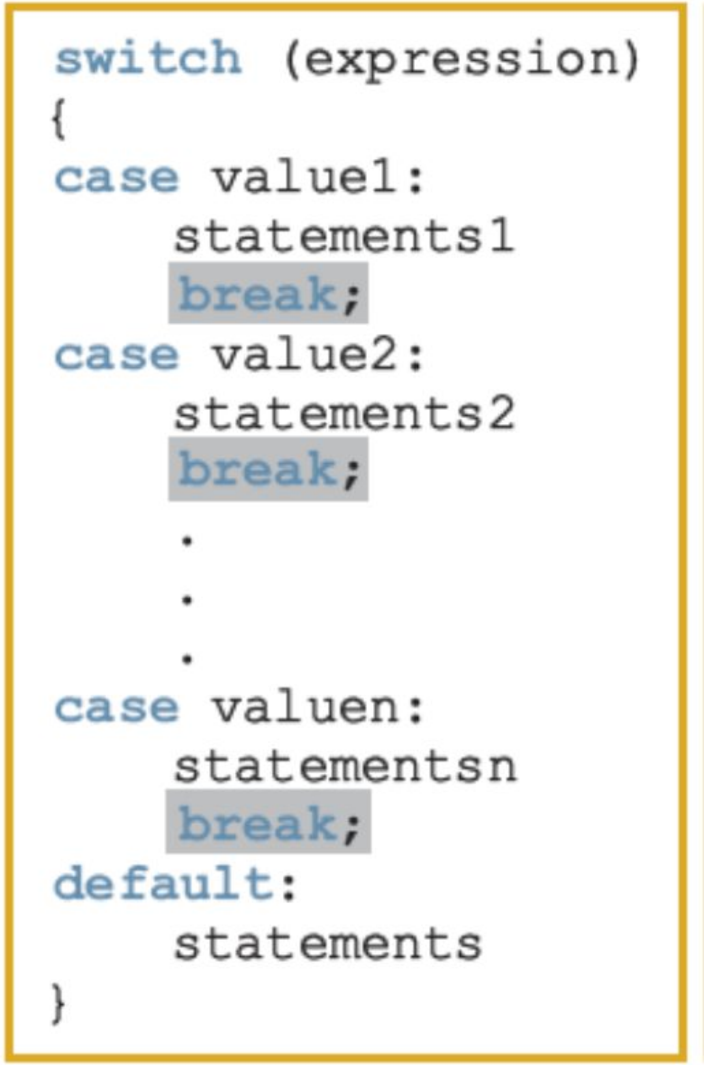
- If your program is properly indented
  - Spot and fix errors quickly
  - Show the natural grouping of statements
- Insert a **blank line between statements** that are naturally separate
- Two commonly used styles for placing braces
  - **On a line** by themselves
  - Or left brace is placed after the expression, and the right brace is on a line by itself

Using Pseudocode to Develop, Test, and Debug a Program

- Pseudocode, or just pseudo
- Informal mixture of C++ and **ordinary language**
- Helps you quickly **develop the correct structure** of the program and avoid making common errors
- Use a **wide range of values** in a walk-through to evaluate the program

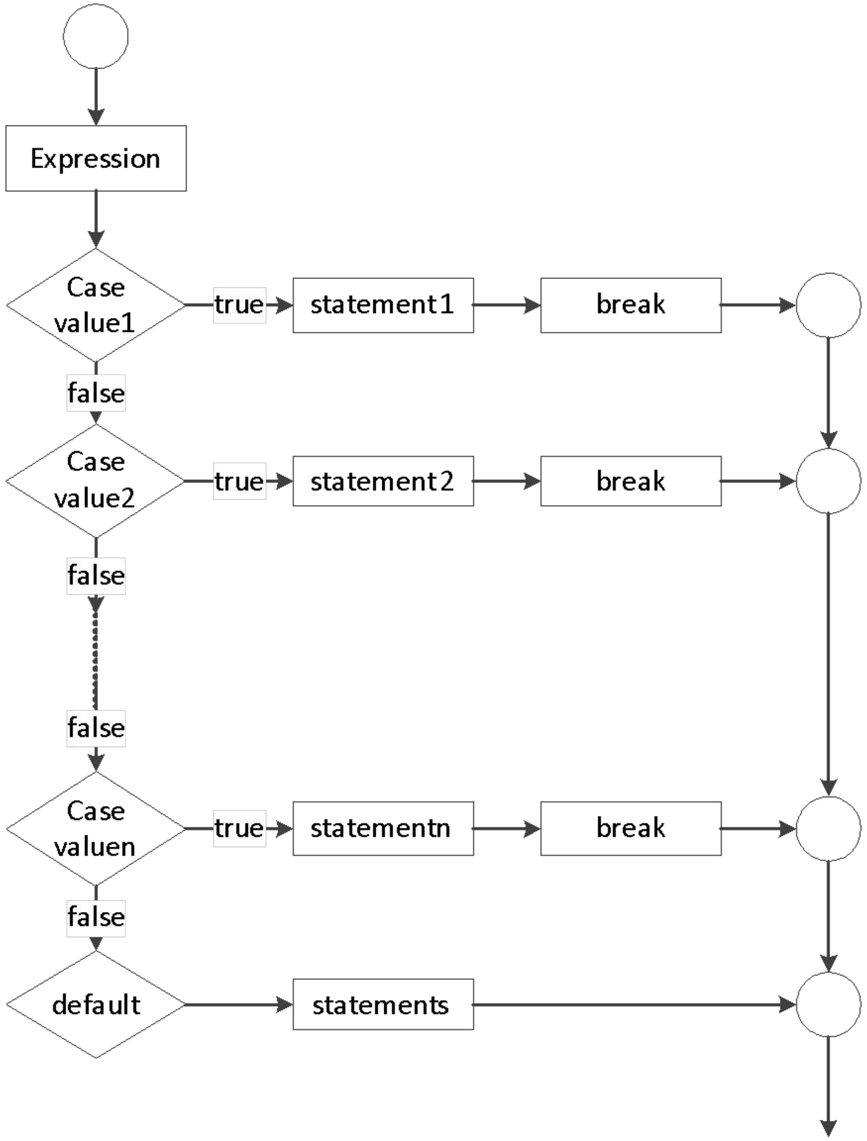
switch Structures

- **switch structure:** Alternate to **if-else**
- switch **(integral)** expression is evaluated first
- Value of the expression determines which corresponding action is taken
- Expression is sometimes called the selector



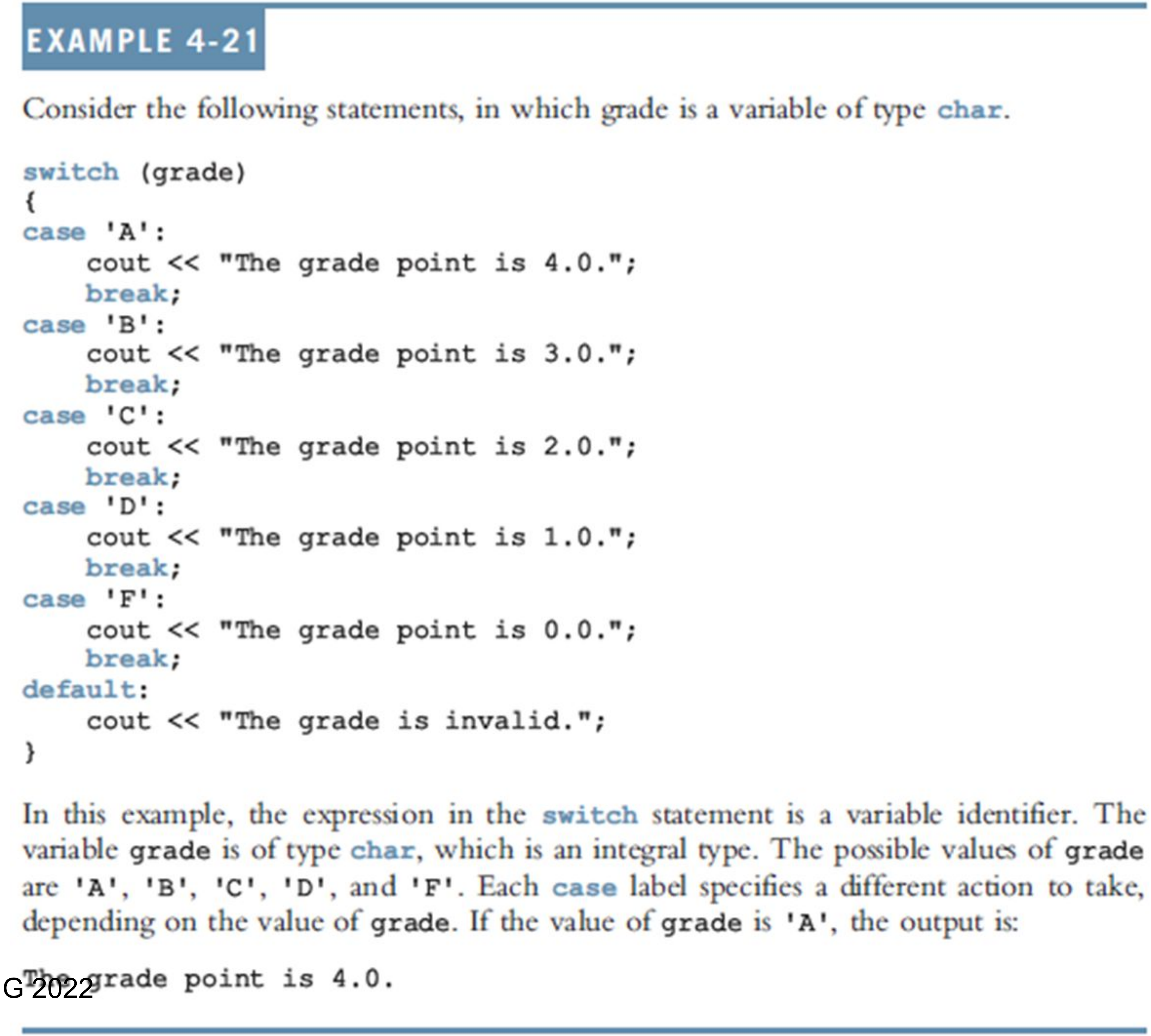
```
switch (expression)
{
case value1:
      statements1
      break;
case value2:
      statements2
      break;
         .
         .
         .
case valuen:
      statementsn
      break;
default:
      statements
}
```

switch **Structures (cont'd.)**



switch **Structures (cont'd.)**

- One or more statements may follow a case label
- **Braces are not needed to turn multiple** statements into a single compound statement
- The break statement may or may not appear after each statement
- **switch**, **case**, **break**, and **default** are reserved words



EXAMPLE 4-21

Consider the following statements, in which grade is a variable of type char.

```
switch (grade)
{
case 'A':
    cout << "The grade point is 4.0.";
    break;
case 'B':
    cout << "The grade point is 3.0.";
    break;
case 'C':
    cout << "The grade point is 2.0.";
    break;
case 'D':
    cout << "The grade point is 1.0.";
    break;
case 'F':
    cout << "The grade point is 0.0.";
    break;
default:
    cout << "The grade is invalid.";
}
```

In this example, the expression in the switch statement is a variable identifier. The variable grade is of type char, which is an integral type. The possible values of grade are 'A', 'B', 'C', 'D', and 'F'. Each case label specifies a different action to take, depending on the value of grade. If the value of grade is 'A', the output is:

```
The grade point is 4.0.
```

```
int main()                                              //Line 3
{                                                       //Line 4
    int testscore;                                      //Line 5

    std::cout << "Enter the test score: ";              //Line 6
    std::cin >> testscore;                              //Line 7
    std::cout << std::endl;                             //Line 8
    switch (testscore / 10)                             //Line 9
    {                                                   //Line 10
    case 0:                                             //Line 11
    case 1:                                             //Line 12
    case 2:                                             //Line 13
    case 3:                                             //Line 14
    case 4:                                             //Line 15
    case 5:                                             //Line 16
        std::cout << "The grade is F." << std::endl;    //Line 17
        break;                                          //Line 18
    case 6:                                             //Line 19
        std::cout << "The grade is D." << std::endl;    //Line 20
        break;                                          //Line 21
    case 7:                                             //Line 22
        std::cout << "The grade is C." << std::endl;    //Line 23
        break;                                          //Line 24
    case 8:                                             //Line 25
        std::cout << "The grade is B." << std::endl;    //Line 26
        break;                                          //Line 27
    case 9:                                             //Line 28
    case 10:                                            //Line 29
        std::cout << "The grade is A." << std::endl;    //Line 30
        break;                                          //Line 31
    default:                                            //Line 32
        std::cout << “Incorrect marks" << std::endl;    //Line 33
    }                                                   //Line 34
    return 0;                                           //Line 35
}                                                       //Line 36
```

Output = ?

```
#include<iostream>
using namespace std;
int main() {
	int number;
	cout << "Enter a number in the range 0 - 7 : ";
	cin >> number;
	cout << "The number you entered is = " << number << endl;
	switch (number) {
	case 0:
	case 1:
		cout << "Learning to use ";
	case 2:
		cout << "C++'s ";
	case 3:
		cout << "switch structure." << endl;
		break;
	case 4:
		break;
	case 5:
		cout << "This program shows the effect ";
	case 6:
	case 7:
		cout << "of break statement." << endl;
		break;
	default:
		cout << "The number is out of range." << endl;
	}
	cout << "Out of the switch structure" << endl;
	return 0;
}
```

0
2
4
5
7
8

Avoiding Bugs by Avoiding Partially Understood Concepts and Techniques: Revisited

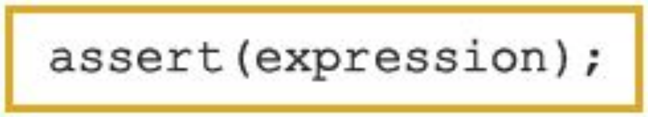
- To output results correctly
- The **switch** structure must include a **break** statement after each **cout** statement

Terminating a Program with the assert Function

- Certain types of errors that are very difficult to catch can occur in a program
    - **Example:** Division by zero can be difficult to catch using any of the programming techniques examined so far
- The predefined function, **assert**, is useful in stopping program execution when certain elusive errors occur

The assert Function (cont'd.)

- **Syntax:**



```
assert(expression);
```

- **expression** is any logical expression
- If **expression** evaluates to **true**, the next statement executes
- If **expression** evaluates to **false**, the program terminates and indicates where in the program the error occurred
- To use **assert**, include **cassert** header file

The assert Function (cont'd.)

- **assert** is useful for enforcing programming constraints during program development
- After developing and testing a program, remove or disable assert statements
- The preprocessor directive **#define NDEBUG** must be placed before the directive **#include <cassert>** to disable the assert statement

assert example

```cpp
#include <iostream>
//#define NDEBUG
#include <cassert>

using namespace std;

int main()
{
   int den, num;
   cout << "Enter two integers" << endl;
   cin >> num >> den;
   assert(den != 0);
   cout << "Moving forward" << endl;
   cout << "num / den = " << num / den << endl;
   return 0;
}
```

Questions