Selections

Selection

- Like all high-level programming languages, Java provides selection statements: statements that let you choose actions with alternative courses.

```
if (radius < 0) {
  System.out.println("Incorrect input");
}
else {
  area = radius * radius * 3.14159;
  System.out.println("Area is " + area);
}
```

Boolean Data Type

- Selection statements use conditions that are Boolean expressions.
- A Boolean expression is an expression that evaluates to a Boolean value: true or false.
- The Boolean data type declares a variable with the value either true or false.

Boolean Data Type

- The result of the comparison is a Boolean value: true or false.

TABLE 3.1 Relational Operators

Java Operator	*Mathematics Symbol*	*Name*	*Example (radius is 5)*	*Result*
`<`	$<$	less than	`radius < 0`	`false`
`<=`	$\leq$	less than or equal to	`radius <= 0`	`false`
`>`	$>$	greater than	`radius > 0`	`true`
`>=`	$\geq$	greater than or equal to	`radius >= 0`	`true`
`==`	$=$	equal to	`radius == 0`	`false`
`!=`	$\neq$	not equal to	`radius != 0`	`true`

Boolean Data Type

- A variable that holds a Boolean value is known as a Boolean variable.
- The Boolean data type is used to declare Boolean variables.
- A Boolean variable can hold one of the two values: true or false.
- For example, the following statement assigns true to the variable lightsOn:

```
boolean lightsOn = true;
```

- true and false are literals, just like a number such as 10.

Selection Statements

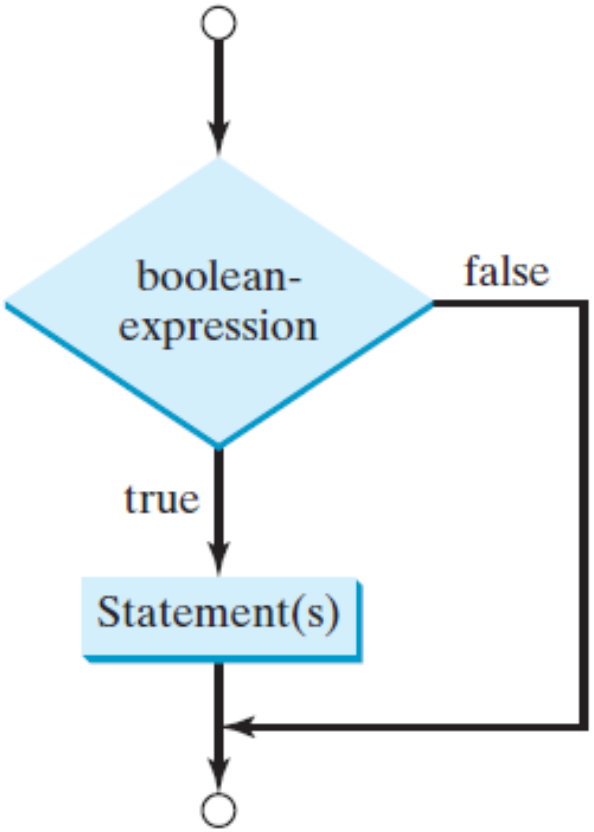
- Java has several types of selection statements:
  - one-way if statements,
  - two-way if-else statements,
  - nested if statements,
  - multi-way if-else statements,
  - switch statements,
  - and conditional expressions.

One Way if Statement

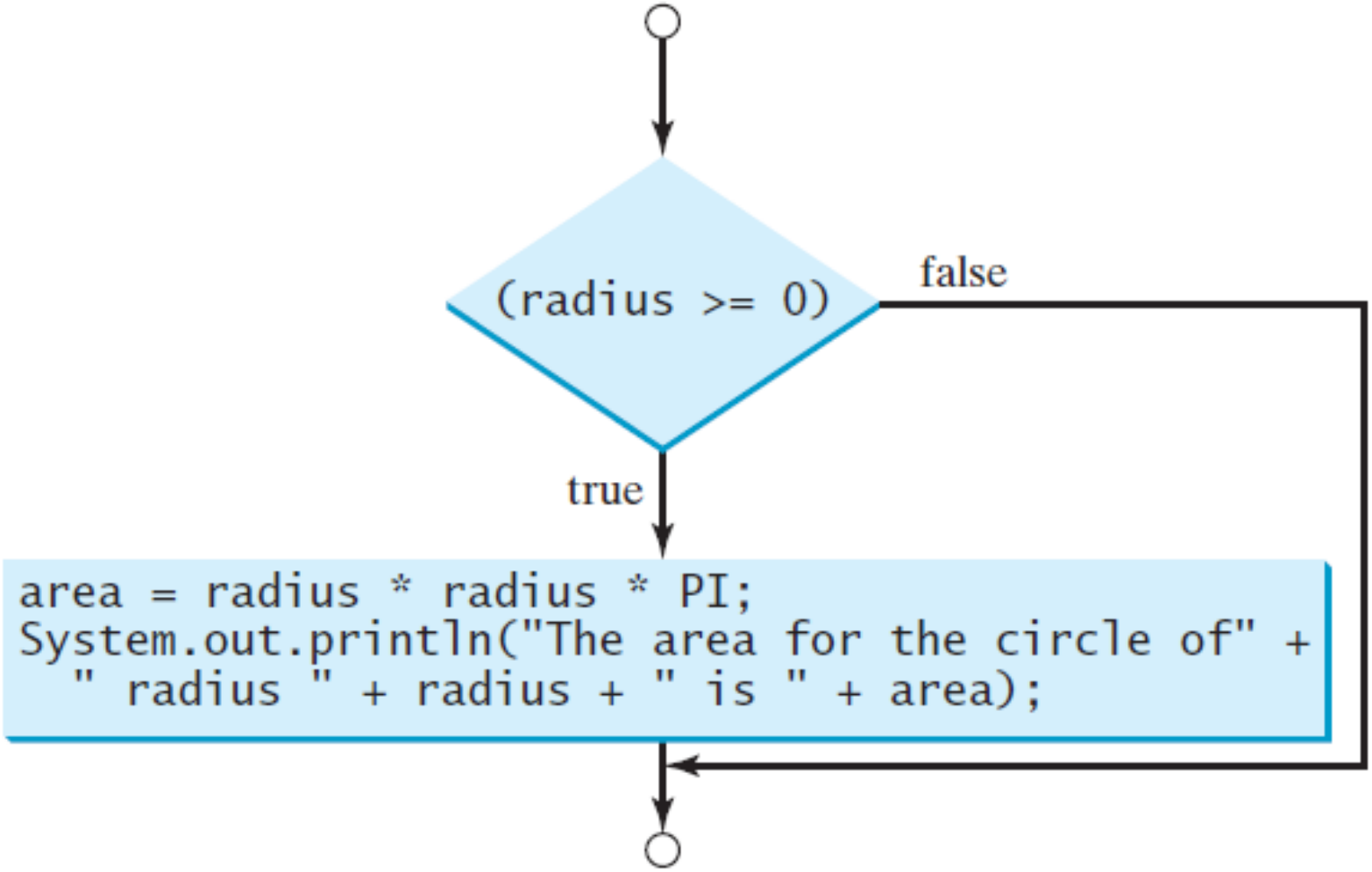
- An if statement is a construct that enables a program to specify alternative paths of execution.
- A one-way if statement executes an action if and only if the condition is true.

```
if (boolean-expression) {
        statement(s);
}
```

One Way if Statement (flowchart)



One Way if Statement (flowchart)



One Way if Statement

- The block braces can be omitted if they enclose a single statement.

```
if (i > 0) {
  System.out.println("i is positive");
}
```

```
if (i > 0)
  System.out.println("i is positive");
```

Example

LISTING 3.2 SimpleIfDemo.java

```
1  import java.util.Scanner;
2
3  public class SimpleIfDemo {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6      System.out.println("Enter an integer: ");
7      int number = input.nextInt();
8
9      if (number % 5 == 0)
10       System.out.println("HiFive");
11
12     if (number % 2 == 0)
13       System.out.println("HiEven");
14   }
15 }
```

enter input

check 5

check even

Example

- The program prompts the user to enter an integer (lines 6–7) and displays HiFive if it is divisible by 5 (lines 9–10) and HiEven if it is divisible by 2 (lines 12–13).

```
Enter an integer:  4 ↵Enter
HiEven
```

```
Enter an integer:  30 ↵Enter
HiFive
HiEven
```

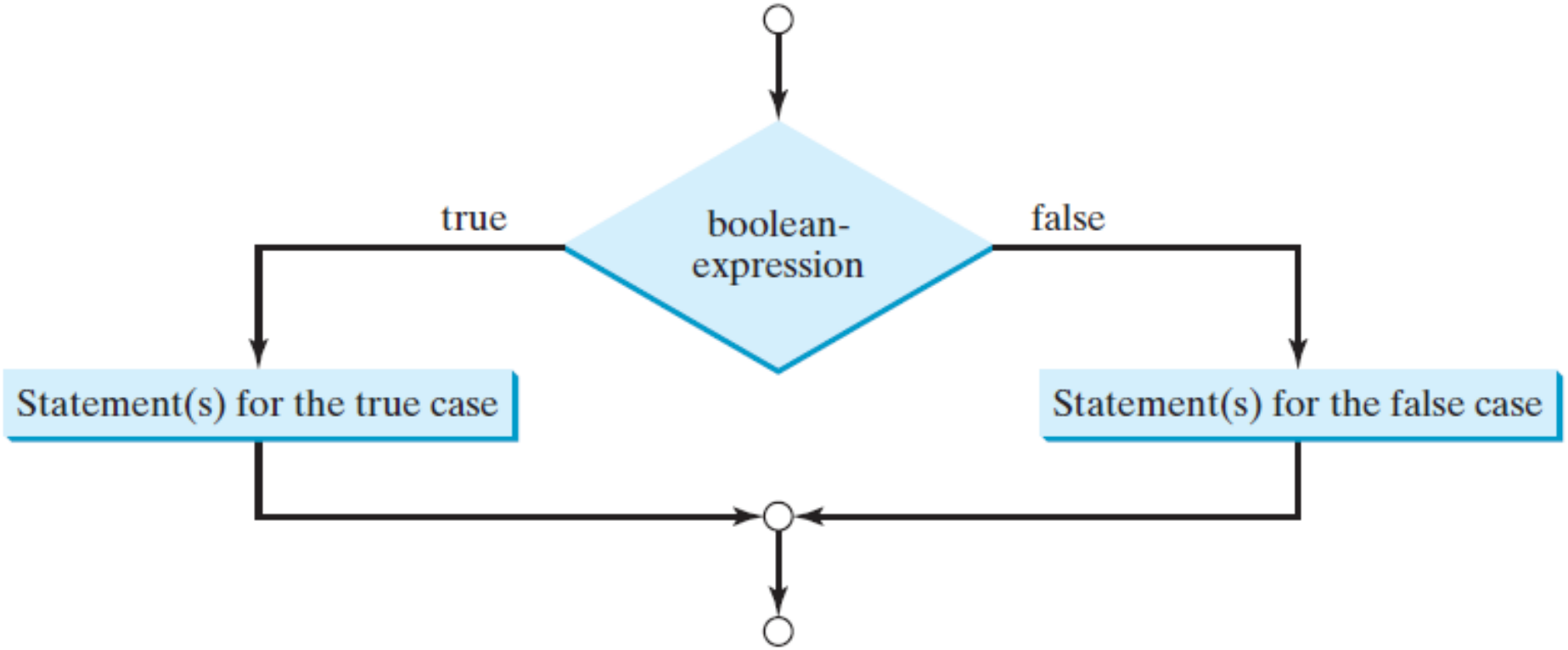
Two Way if-else Statement

- A one-way if statement performs an action if the specified condition is true. If the condition is false, nothing is done.
- But what if you want to take alternative actions when the condition is false? You can use a two-way if-else statement.
- The actions that a two-way if-else statement specifies differ based on whether the condition is true or false.

Syntax

```
if (boolean-expression) {
        statement(s)-for-the-true-case;
}
else {
        statement(s)-for-the-false-case;
}
```

Flowchart


true
boolean-
expression
false
Statement(s) for the true case
Statement(s) for the false case

Example

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area for the circle of radius " +
    radius + " is " + area);
}
else {
  System.out.println("Negative input");
}
```

Example 2

- Here are two example of using if and if-else statement. The examples checks whether a number is even or odd.

```
if (number % 2 == 0)
  System.out.println(number + " is even.");

System.out.println(number + " is odd.");
```

(a)

```
if (number % 2 == 0)
  System.out.println(number + " is even.");
else
  System.out.println(number + " is odd.");
```

(b)

Example 2

- Java code (a) with one way if-statement works fine when the introduced number is odd. However, two messages will be displayed (even and odd) when an even number is introduced.
- The Java code (b) with two way if-else statement resolves the issue.

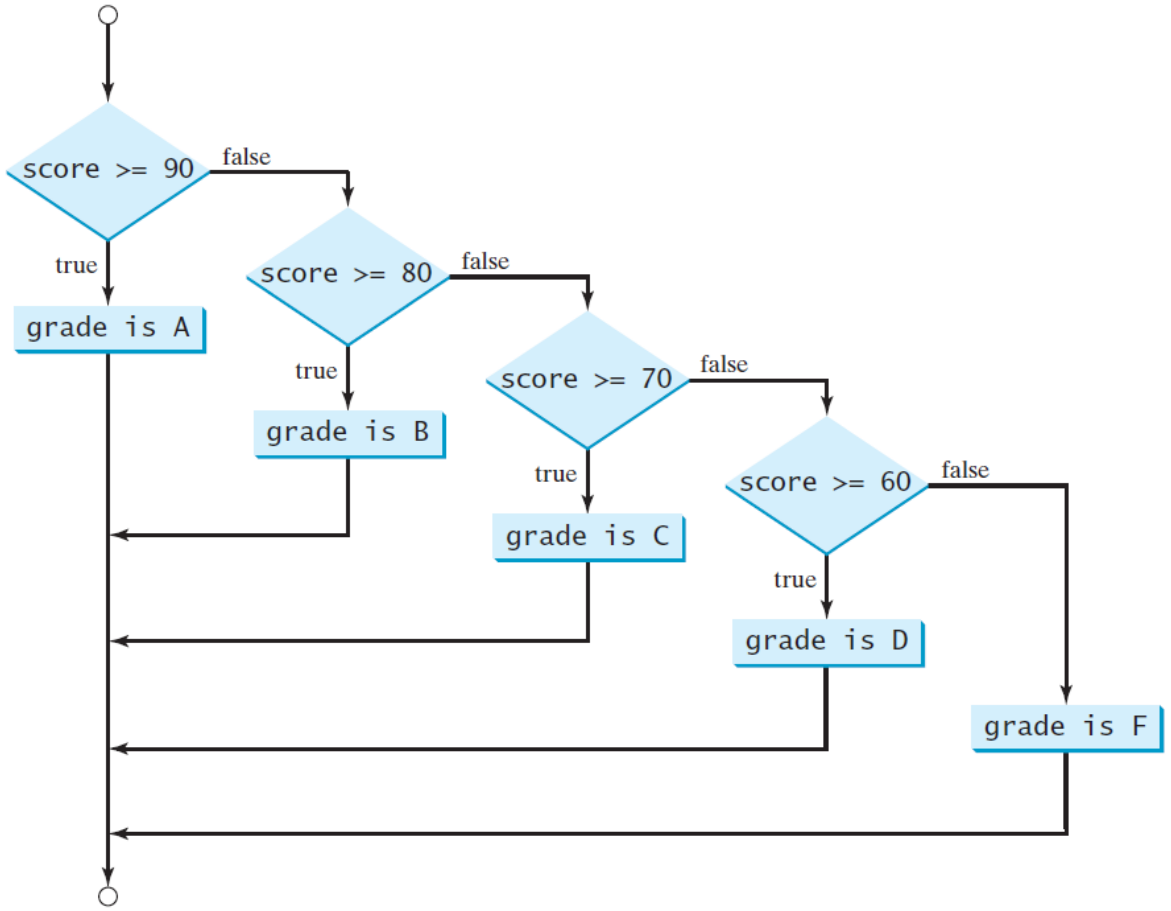
Nested if statements

- An if statement can be inside another if statement to form a nested if statement.

```
if (i > k) {
  if (j > k)
    System.out.println("i and j are greater than k");
}
else
  System.out.println("i is less than or equal to k");
```

Multi Way if-else Statements

- The nested if statement can be used to implement multiple alternatives.



Multi Way if-else Statements (with indentations)

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

(a)

Multi Way if-else Statements (preferred format)

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

(b)

Logical Operators

- Sometimes, whether a statement is executed is determined by a combination of several conditions.
- You can use logical operators to combine these conditions to form a compound Boolean expression.
- Logical operators, also known as Boolean operators, operate on Boolean values to create a new Boolean value.

Logical Operators

TABLE 3.3 Boolean Operators

Operator	*Name*	*Description*
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

Operator Not (!)

TABLE 3.4 Truth Table for Operator !

p	!p	*Example (assume* `age = 24, weight = 140`*)*
`true`	`false`	`!(age > 18)` is `false`, because `(age > 18)` is `true`.
`false`	`true`	`!(weight == 150)` is `true`, because `(weight == 150)` is `false`.

Operator And (&&)

TABLE 3.5 Truth Table for Operator &&

p_1	p_2	p_1 && p_2	*Example (assume* `age = 24, weight = 140`*)*
`false`	`false`	`false`	
`false`	`true`	`false`	`(age > 28) && (weight <= 140)` is `false`, because `(age > 28)` is `false`.
`true`	`false`	`false`	
`true`	`true`	`true`	`(age > 18) && (weight >= 140)` is `true`, because `(age > 18)` and `(weight >= 140)` are both `true`.

Operator Or (| |)

TABLE 3.6 Truth Table for Operator | |

p_1	p_2	p_1 || p_2	*Example (assume* `age = 24, weight = 140`*)*
`false`	`false`	`false`	`(age > 34) || (weight >= 150)` is `false`, because `(age > 34)` and `(weight >= 150)` are both `false`.
`false`	`true`	`true`	
`true`	`false`	`true`	`(age > 18) || (weight < 140)` is `true`, because `(age > 18)` is `true`.
`true`	`true`	`true`	

Operator Exclusive Or (^)

TABLE 3.7 Truth Table for Operator ^

p_1	p_2	p_1 ^ p_2	*Example (assume* `age = 24, weight = 140`*)*
`false`	`false`	`false`	`(age > 34) ^ (weight > 140)` is `false`, because `(age > 34)` and `(weight > 140)` are both `false`.
`false`	`true`	`true`	`(age > 34) ^ (weight >= 140)` is `true`, because `(age > 34)` is `false` but `(weight >= 140)` is `true`.
`true`	`false`	`true`	
`true`	`true`	`false`	

Switch Statements

- A switch statement executes statements based on the value of a variable or an expression.

```
switch (switch-expression) {
  case value1: statement(s)1;
               break;

  case value2: statement(s)2;
               break;
  ...
  case valueN: statement(s)N;
               break;
  default:     statement(s)-for-default;
}
```

Switch Statements

- The switch statement observes the following rules:
  - The switch-expression must yield a value of char, byte, short, int, or String type and must always be enclosed in parentheses.
  - The value1, . . ., and valueN must be constant and have the same data type as the value of the switch-expression.

Switch Statements

- When the value in a case statement matches the value of the switch-expression, the statements starting from this case are executed until either a break statement or the end of the switch statement is reached.
- The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.
- The keyword break is optional. The break statement immediately ends the switch statement.

Example

- What is the corresponding output for different integer values of day ?

```
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5: System.out.println("Weekday"); break;
    case 0:
    case 6: System.out.println("Weekend");
}
```

Example

year % 12 =

0: monkey
1: rooster
2: dog
3: pig
4: rat
5: ox
6: tiger
7: rabbit
8: dragon
9: snake
10: horse
11: sheep

FIGURE 3.6 The Chinese Zodiac is based on a twelve-year cycle.

LISTING 3.9 ChineseZodiac.java

```
1  import java.util.Scanner;
2
3  public class ChineseZodiac {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6
7      System.out.print("Enter a year: ");
8      int year = input.nextInt();
9
10     switch (year % 12) {
11       case 0: System.out.println("monkey"); break;
12       case 1: System.out.println("rooster"); break;
13       case 2: System.out.println("dog"); break;
14       case 3: System.out.println("pig"); break;
15       case 4: System.out.println("rat"); break;
16       case 5: System.out.println("ox"); break;
17       case 6: System.out.println("tiger"); break;
18       case 7: System.out.println("rabbit"); break;
19       case 8: System.out.println("dragon"); break;
20       case 9: System.out.println("snake"); break;
21       case 10: System.out.println("horse"); break;
22       case 11: System.out.println("sheep");
23     }
24   }
25 }
```

```
Enter a year:  1963  Enter
rabbit
```

```
Enter a year:  1877  Enter
ox
```

Conditional Expressions

- A conditional expression evaluates an expression based on a condition.

  boolean-expression ? expression1 : expression2;

- The result of this conditional expression is expression1 if boolean-expression is true; otherwise the result is expression2.

Examples

```
max = (num1 > num2) ? num1 : num2;
```

- It assigns the larger number of variable num1 and num2 to max.

```
System.out.println((num % 2 == 0) ? "num is even" : "num is odd");
```

- It displays the message “num is even” if num is even, and otherwise displays “num is odd.”

Operator Precedence and Associativity

- Operator precedence and associativity determine the order in which operators are evaluated.
- The precedence rule defines precedence for operators.
- The logical operators have lower precedence than the relational operators and the relational operators have lower precedence than the arithmetic operators.

Operator Precedence and Associativity

TABLE 3.8 Operator Precedence Chart

Precedence	*Operator*
↓	+, – (Unary plus and minus), ++var and --var (Prefix)
	(type) (Casting)
	! (Not)
	*, /, % (Multiplication, division, and remainder)
	+, – (Binary addition and subtraction)
	<, <=, >, >= (Relational)
	==, != (Equality)
	^ (Exclusive OR)
	&& (AND)
	|| (OR)
	=, +=, –=, *=, /=, %= (Assignment operator)
	var++ and var-- (Postfix)

Operator Precedence and Associativity

- If operators with the same precedence are next to each other, their associativity determines the order of evaluation.
- All binary operators except assignment operators are left associative.

```
a - b + c - d   is equivalent to   ((a - b) + c) - d
```

Operator Precedence and Associativity

- Assignment operators are right associative.

```
a = b += c = 5   is equivalent to   a = (b += (c = 5))
```