PROGRAMMING WITH C++

DST102TC

JIANJIA WANG

JIANJIA.WANG@XJTLU.EDU.CN

SCHOOL OF AI AND ADVANCED COMPUTING

Xi'an Jiaotong-Liverpool University

西交利物浦大学

CHAPTER 3

SELECTION

Xi'an Jiaotong-Liverpool University
西交利物浦大學

MOTIVATIONS

If you assigned a negative value for radius in Listing 2.1, ComputeArea.cpp, the program would print an invalid result. If the radius is negative, you don't want the program to compute the area. How can you deal with this situation?

OBJECTIVES

To declare bool variables and write Boolean expressions using relational operators (§3.2).

To implement selection control using one-way if statements (§3.3).

To program using one-way if statements (GuessBirthday) (§3.4).

To implement selection control using two-way if statements (§3.5).

To implement selection control using nested if and multi-way if-else statements (§3.6).

To avoid common errors and pitfalls in if statements (§3.7).

To generate random numbers using the rand function and set a seed using the srand function (§3.9).

To combine conditions using logical operators (&&, ||, and !) (§3.10).

To program using selection statements with combined conditions (LeapYear, Lottery) (§§3.11–3.12).

To implement selection control using switch statements (§3.13).

To write expressions using the conditional operator (§3.14).

To examine the rules governing operator precedence and operator associativity (§3.15).

THE BOOL TYPE AND OPERATORS

Often in a program you need to compare two values, such as whether i is greater than j.

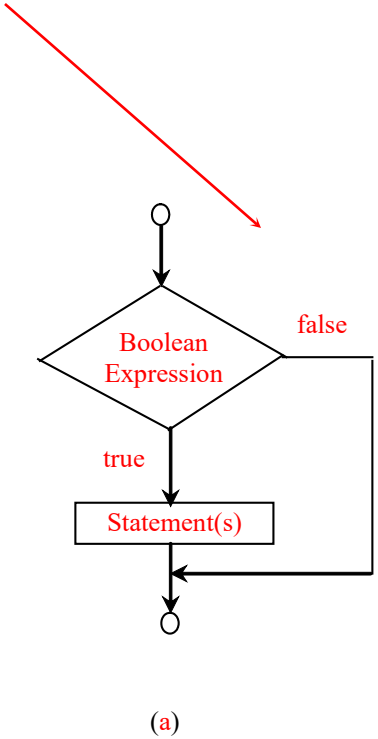
C++ provides six *relational operators* (also known as *comparison operators*) in Table 3.1 that can be used to compare two values.

RELATIONAL OPERATORS

Operator	Name	Example	Result
<	less than	1 < 2	true
<=	less than or equal to	1 <= 2	true
>	greater than	1 > 2	false
>=	greater than or equal to	1 >= 2	false
==	equal to	1 == 2	false
!=	not equal to	1 != 2	true

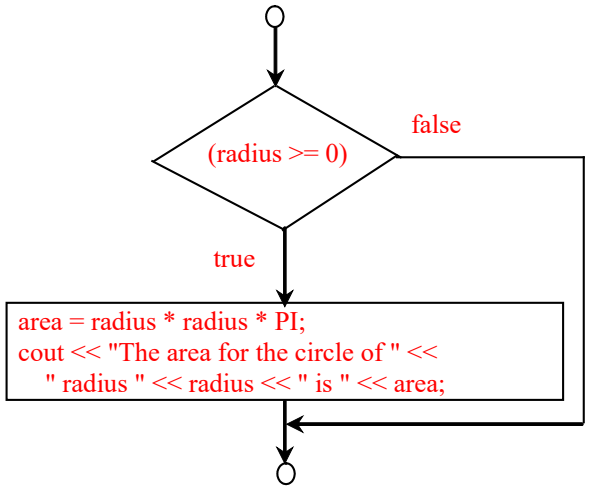
ONE-WAY IF STATEMENTS

```
if (booleanExpression)
{
  statement(s);
}
```



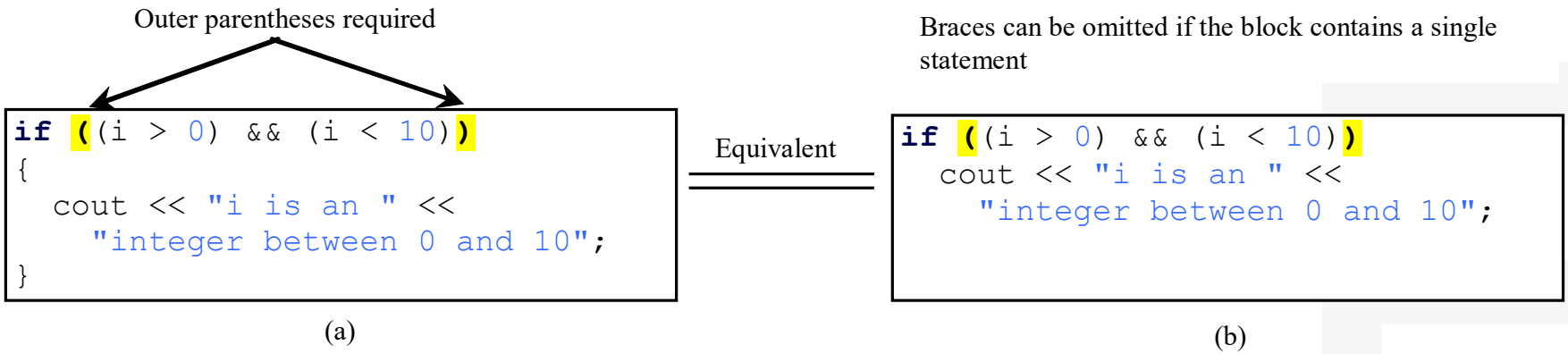
(a)

```
if (radius >= 0)
{
  area = radius * radius * PI;
  cout << "The area for the circle of " <<
    " radius " << radius << " is " << area;
}
```



(b)

NOTE


Outer parentheses required
Braces can be omitted if the block contains a single statement
if ((i > 0) && (i < 10))
{
cout << "i is an " <<
"integer between 0 and 10";
}
Equivalent
if ((i > 0) && (i < 10))
cout << "i is an " <<
"integer between 0 and 10";
(a)
(b)

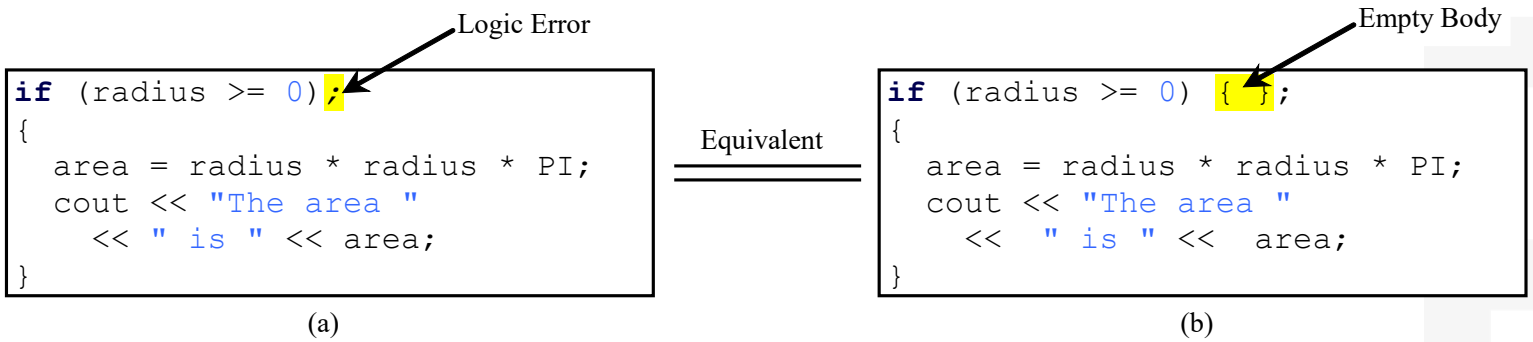
EXAMPLES

Listing 3.1 gives a program that prompts the user to enter an integer. If the number is a multiple of **5**, display **HiFive**. If the number is even, display **HiEven**.

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

int main()
{
  // Prompt the user to enter an integer
  int number;
  cout << "Enter an integer: ";
  cin >> number;
  if (number % 5 == 0)
    cout << "HiFive" << endl;
  if (number % 2 == 0 )
    cout << "HiEven" << endl;
  return 0;
}
```

CAUTION

Adding a semicolon at the end of an if clause is a common mistake.

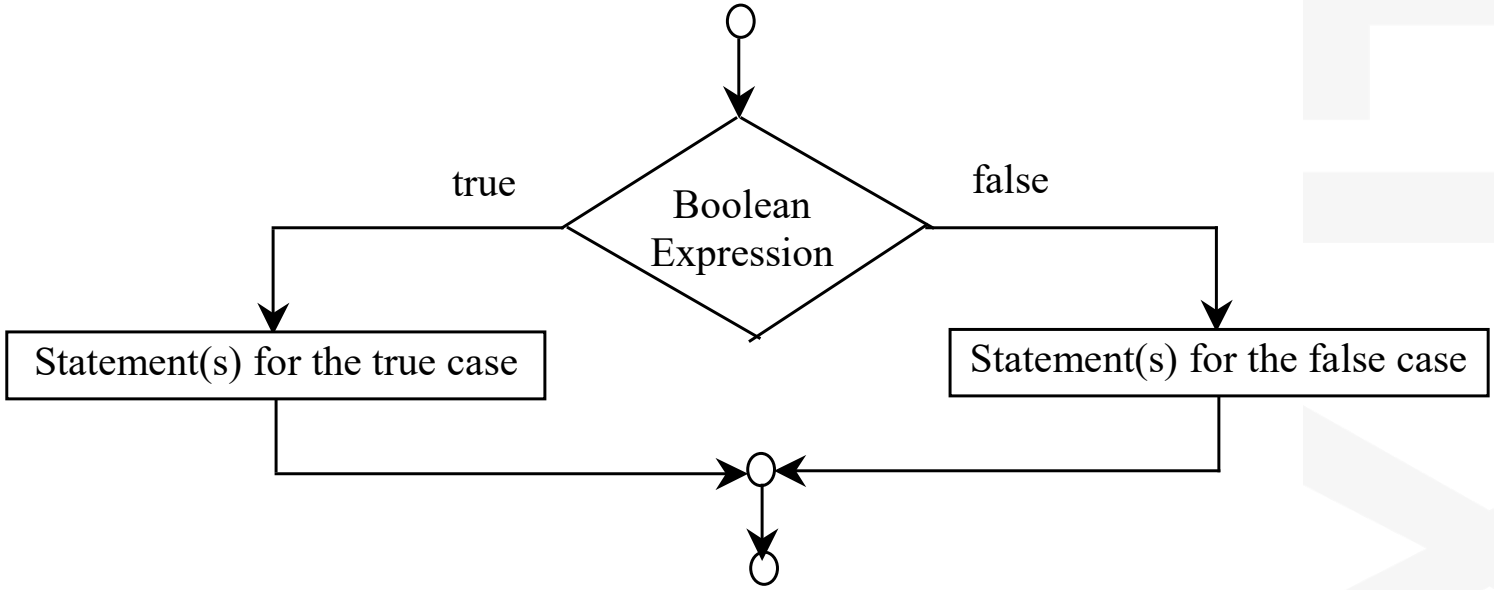


This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.

THE IF...ELSE STATEMENT

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



NESTED IF STATEMENTS

```
if (i > k)
{
  if (j > k)
    cout << "i and j are greater than k";
}
else
  cout << "i is less than or equal to k";
```

MULTIPLE ALTERNATIVE IF STATEMENTS

```
if (score >= 90.0)
  cout << "Grade is A";
else
  if (score >= 80.0)
    cout << "Grade is B";
  else
    if (score >= 70.0)
      cout << "Grade is C";
    else
      if (score >= 60.0)
        cout << "Grade is D";
      else
        cout << "Grade is F";
```

(a)

Equivalent

This is better

```
if (score >= 90.0)
  cout << "Grade is A";
else if (score >= 80.0)
  cout << "Grade is B";
else if (score >= 70.0)
  cout << "Grade is C";
else if (score >= 60.0)
  cout << "Grade is D";
else
  cout << "Grade is F";
```

(b)

TRACE IF-ELSE STATEMENT

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
  cout << “Grade is A";
else if (score >= 80.0)
  cout << “Grade is B";
else if (score >= 70.0)
  cout << “Grade is C";
else if (score >= 60.0)
  cout << “Grade is D";
else
  cout << “Grade is F";
```

TRACE IF-ELSE STATEMENT

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
  cout << “Grade is A";
else if (score >= 80.0)
  cout << “Grade is B";
else if (score >= 70.0)
  cout << “Grade is C";
else if (score >= 60.0)
  cout << “Grade is D";
else
  cout << “Grade is F";
```

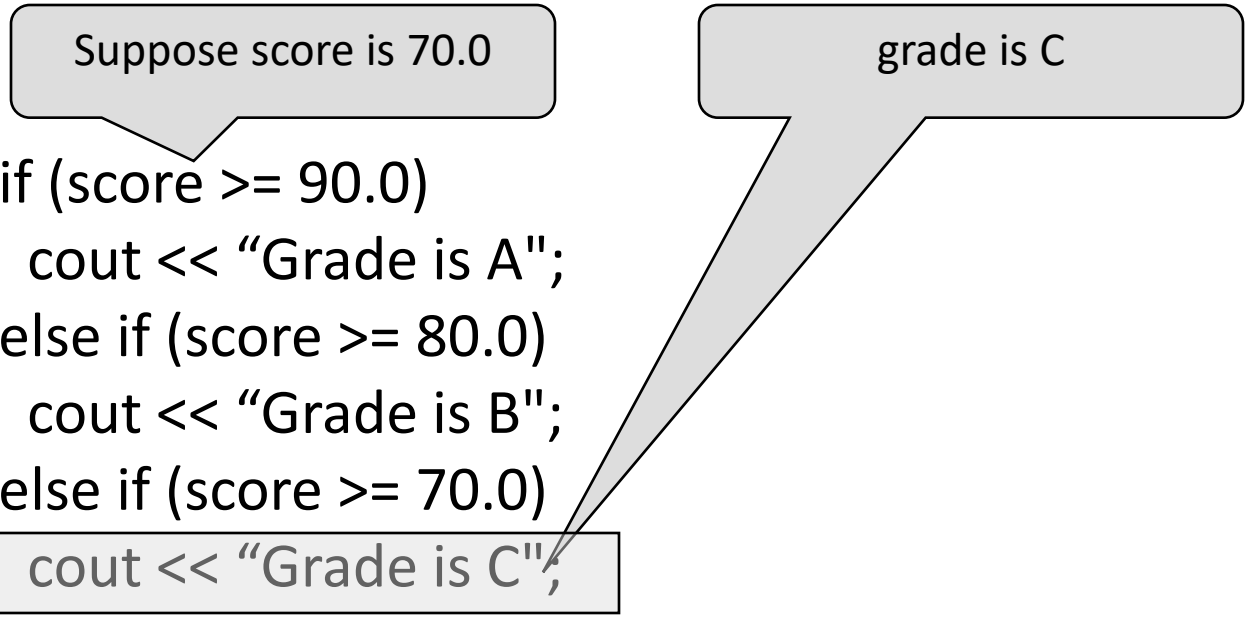
TRACE IF-ELSE STATEMENT

Suppose score is 70.0

The condition is true

```
if (score >= 90.0)
  cout << “Grade is A";
else if (score >= 80.0)
  cout << “Grade is B";
else if (score >= 70.0)
  cout << “Grade is C";
else if (score >= 60.0)
  cout << “Grade is D";
else
  cout << “Grade is F";
```

TRACE IF-ELSE STATEMENT




```
if (score >= 90.0)
  cout << “Grade is A";
else if (score >= 80.0)
  cout << “Grade is B";
else if (score >= 70.0)
  cout << “Grade is C";
else if (score >= 60.0)
  cout << “Grade is D";
else
  cout << “Grade is F";
```

TRACE IF-ELSE STATEMENT

Suppose score is 70.0

Exit the if statement

```
if (score >= 90.0)
  cout << “Grade is A";
else if (score >= 80.0)
  cout << “Grade is B";
else if (score >= 70.0)
  cout << “Grade is C";
else if (score >= 60.0)
  cout << “Grade is D";
else
  cout << “Grade is F";
```

NOTE

The else clause matches the most recent if clause in the same block.

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    cout << "A";
else
    cout << "B";
```

(a)

Equivalent

This is better with correct indentation

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    cout << "A";
  else
    cout << "B";
```

(b)

NOTE, CONT.

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1; int j = 2; int k = 3;

if (i > j)
{
  if (i > k)
    cout << "A";
}
else
  cout << "B";
```

This statement prints B.

TIP

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

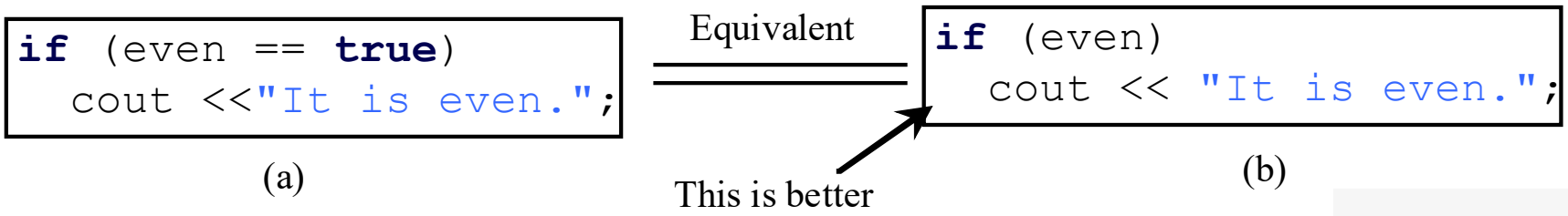
Equivalent

This is better

```
bool even
  = number % 2 == 0;
```

(b)

CAUTION


if (even == true)
cout <<"It is even.";
Equivalent
if (even)
cout << "It is even.";
(a)
This is better
(b)

COMMON ERRORS IN SELECTION STATEMENTS

Common Error 1: Forgetting Necessary Braces

```
if (radius >= 0)
  area = radius * radius * PI;
  cout << "The area "
    << " is " << area;
```

(a) Wrong

```
if (radius >= 0)
{
  area = radius * radius * PI;
  cout << "The area "
    <<  " is " <<  area;
}
```

(b) Correct

COMMON ERRORS IN SELECTION STATEMENTS

Common Error 2: Wrong Semicolon at the if Line

Logic Error

```
if (radius >= 0);
{
  area = radius * radius * PI;
  cout << "The area "
    << " is " << area;
}
```

(a)

Equivalent

Empty Body

```
if (radius >= 0) { };
{
  area = radius * radius * PI;
  cout << "The area "
    <<  " is " <<  area;
}
```

(b)

COMMON ERRORS IN SELECTION STATEMENTS

Common Error 3: Mistakenly Using = for ==

```
if (count = 1)
  cout << "count is zero" << endl;
else
  cout << "count is not zero" << endl;
```

COMMON ERRORS IN SELECTION STATEMENTS

Common Error 4: Redundant Testing of Boolean Values

```
if (even == true)
  cout <<"It is even.";
```

(a)

Equivalent

```
if (even)
  cout << "It is even.";
```

(b)

This is better

EXAMPLE: BODY MASS INDEX

Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

BMI	Interpretation
below 16	serious underweight
16-18	underweight
18-24	normal weight
24-29	overweight
29-35	seriously overweight
above 35	gravely overweight

```
#include <iostream>
using namespace std;
int main()
{
  // Prompt the user to enter weight in pounds
  cout << "Enter weight in pounds: ";
  double weight;
  cin >> weight;
  // Prompt the user to enter height in inches
  cout << "Enter height in inches: ";
  double height;
  cin >> height;
  const double KILOGRAMS_PER_POUND = 0.45359237; // Constant
  const double METERS_PER_INCH = 0.0254; // Constant
  // Compute BMI
  double weightInKilograms = weight * KILOGRAMS_PER_POUND;
  double heightInMeters = height * METERS_PER_INCH;
  double bmi = weightInKilograms /
    (heightInMeters * heightInMeters);
  // Display result
  cout << "BMI is " << bmi << endl;
  if (bmi < 18.5)
    cout << "Underweight" << endl;
  else if (bmi < 25)
    cout << "Normal" << endl;
  else if (bmi < 30)
    cout << "Overweight" << endl;
  else
    cout << "Obese" << endl;
  return 0;
}
```

EXAMPLE: A SIMPLE MATH LEARNING TOOL

This example creates a program for a first grader to practice subtractions.

The program randomly generates two single-digit integers number1 and number2 with number1 >= number2 and displays a question such as “What is 9 – 2?” to the student, as shown in the sample output.

After the student types the answer, the program displays a message to indicate whether the answer is correct.

```
#include <iostream>
#include <ctime> // for time function #include <cstdlib> // for rand and srand functions
using namespace std;
int main()
{  // 1. Generate two random single-digit integers
  srand(time(0));
  int number1 = rand() % 10;
int number2 = rand() % 10;
  // 2. If number1 < number2, swap number1 with number2
  if (number1 < number2)
  {    int temp = number1;
    number1 = number2;
    number2 = temp; }
  // 3. Prompt the student to answer  “what is number1 – number2?”
  cout << "What is " << number1 << " - " << number2 << "? ";
  int answer;
  cin >> answer;
  // 4. Grade the answer and display the result
  if (number1 - number2 == answer)
    cout << "You are correct!";
  else
    cout << "Your answer is wrong." << endl << number1 << " - "
        << number2 << " should be " << (number1 - number2) << endl;
  return 0; }
```

LOGICAL OPERATORS

Operator	Name	Description
`!`	`not`	`logical negation`
`&&`	`and`	`logical conjunction`
`||`	`or`	`logical disjunction`

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 fal

TRUTH TABLE FOR OPERATOR &&

p1	p2	p1 && p2	Example (assume age = 24, weight = 140)
false	false	false	(age > 28) && (weight < 140) is false, because (age > 28) and (weight < 140) are both false.
false	true	false	
true	false	false	
true	true	true	(age > 18) && (weight >= 140) is true, because both (age > 18) and (weight >= 140) are true.

TRUTH TABLE FOR OPERATOR ||

p1	p2	p1 || p2	Example (assume age = 24, weight = 140)
false	false	false	(age > 34) || (weight < 140) is false, because both (age > 34) and (weight < 140) are false.
false	true	true	
true	false	true	(age > 18) || (weight >= 150) is false, because (age > 18) is true.
true	true	true	

EXAMPLES

Listing 3.3 gives a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3, and whether a number is divisible by 2 or 3 but not both:

```
#include <iostream>
using namespace std;
int main()
{
  int number;
  cout << "Enter an integer: ";
  cin >> number;
  if (number % 2 == 0 && number % 3 == 0)
    cout << number << " is divisible by 2 and 3." << endl;
  if (number % 2 == 0 || number % 3 == 0)
    cout << number << " is divisible by 2 or 3." << endl;
  if ((number % 2 == 0 || number % 3 == 0) &&
       !(number % 2 == 0 && number % 3 == 0))
    cout << number << " divisible by 2 or 3, but not both." << endl;
  return(0);
}
```

SHORT-CIRCUIT OPERATOR

When evaluating p1 && p2, C++ first evaluates p1 and then evaluates p2 if p1 is true; if p1 is false, it does not evaluate p2. When evaluating p1 || p2, C++ first evaluates p1 and then evaluates p2 if p1 is false; if p1 is true, it does not evaluate p2. Therefore, && is referred to as the *conditional* or *short-circuit AND* operator, and || is referred to as the *conditional* or *short-circuit OR* operator.

EXAMPLES

Write a program that lets the user enter a year and checks whether it is a leap year.

A year is a *leap year* if it is divisible by 4 but not by 100 or if it is divisible by 400. So you can use the following Boolean expression to check whether a year is a leap year:

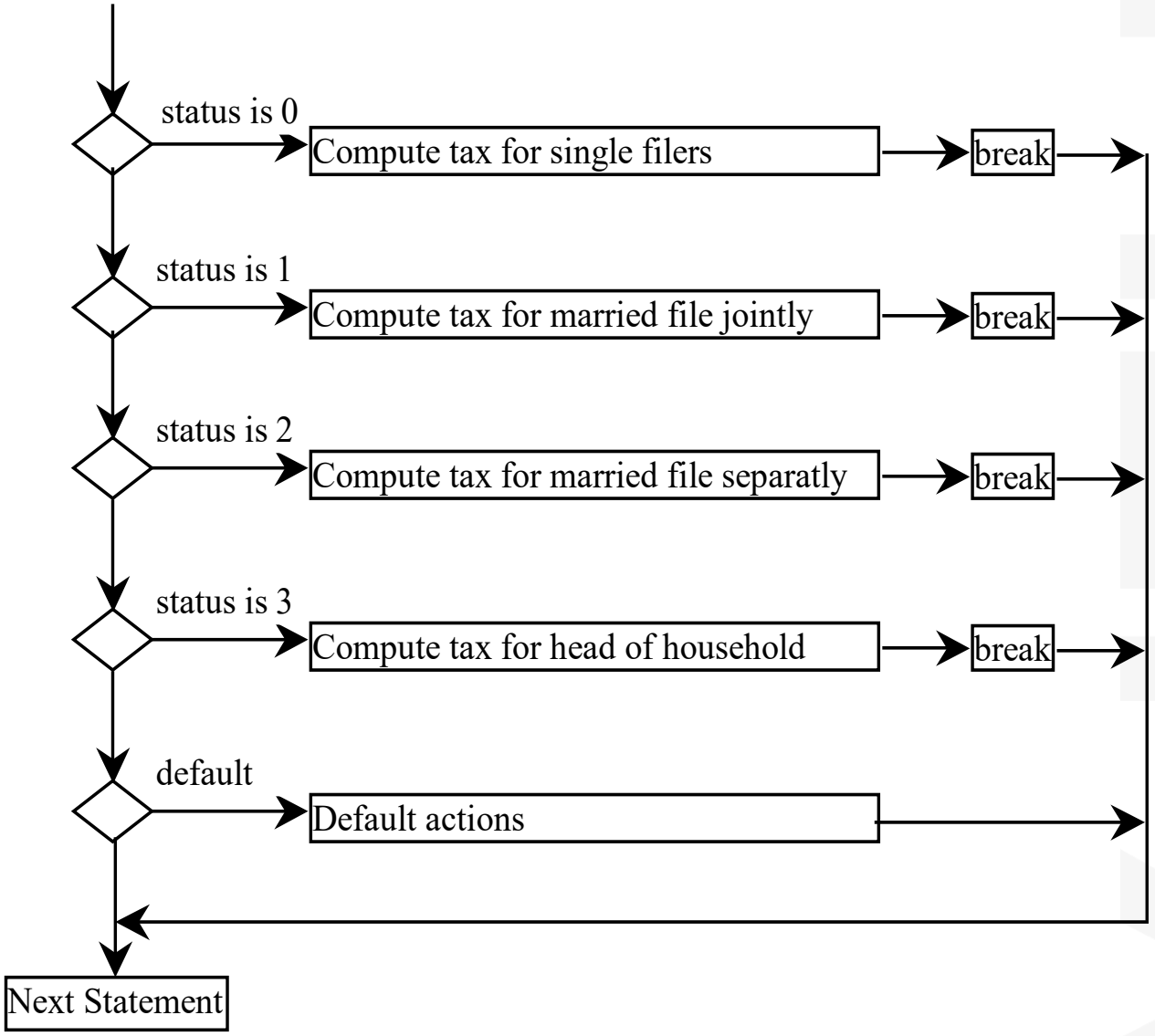
```
(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)
```

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

int main()
{
  cout << "Enter a year: ";
  int year;
  cin >> year;

  // Check if the year is a leap year
  bool isLeapYear =
    (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

  // Display the result in a message dialog box
  if (isLeapYear)
    cout << year << " is a leap year" << endl;
  else
    cout << year << " is a not leap year" << endl;

  return 0;
}
```

SWITCH STATEMENTS

```
switch (status)
{
  case 0:  compute taxes for single filers;
           break;
  case 1:  compute taxes for married file jointly;
           break;
  case 2:  compute taxes for married file separately;
           break;
  case 3:  compute taxes for head of household;
           break;
  default: cout << "Errors: invalid status“ << endl;
}
```

SWITCH STATEMENT FLOW CHART



SWITCH STATEMENT RULES

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.

```
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 STATEMENT RULES

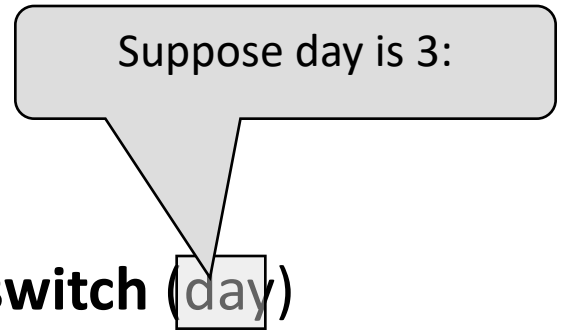
The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the break statement is not present, the next case statement will be executed.

The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-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;
}
```

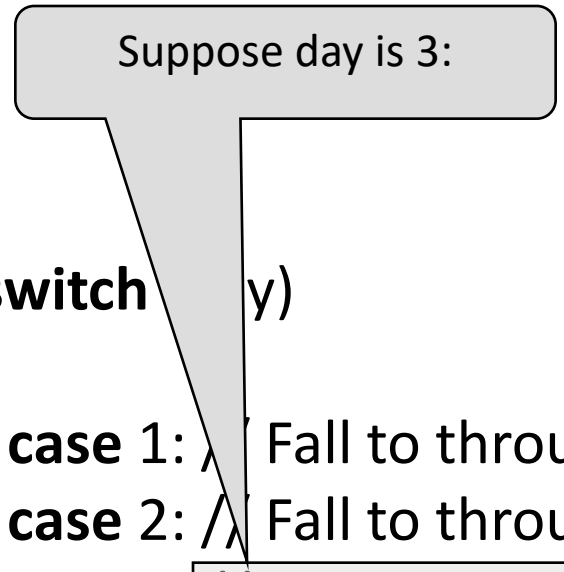
When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached.

TRACE SWITCH STATEMENT



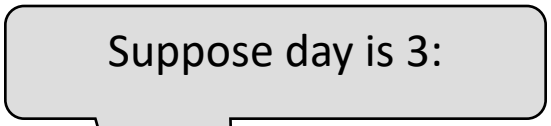

```
switch (day)
{
  case 1: // Fall to through to the next case
  case 2: // Fall to through to the next case
  case 3: // Fall to through to the next case
  case 4: // Fall to through to the next case
  case 5: cout << "Weekday"; break;
  case 0: // Fall to through to the next case
  case 6: cout << "Weekend";
}
```

TRACE SWITCH STATEMENT



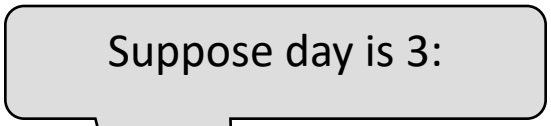

```
switch (day)
{
  case 1: // Fall to through to the next case
  case 2: // Fall to through to the next case
  case 3: // Fall to through to the next case
  case 4: // Fall to through to the next case
  case 5: cout << "Weekday"; break;
  case 0: // Fall to through to the next case
  case 6: cout << "Weekend";
}
```

TRACE SWITCH STATEMENT




```
switch      y)
{
  case 1:   Fall to through to the next case
  case 2: / Fall to through to the next case
  case 3: // Fall to through to the next case
  case 4: // Fall to through to the next case
  case 5: cout << "Weekday"; break;
  case 0: // Fall to through to the next case
  case 6: cout << "Weekend";
}
```

TRACE SWITCH STATEMENT

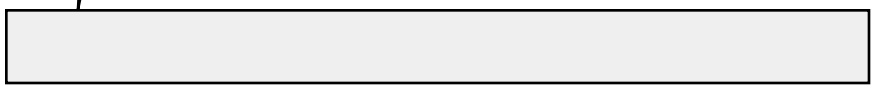



```
switch      y)
{
  case 1:     Fall to through to the next case
  case 2: /   Fall to through to the next case
  case 3: /   Fall to through to the next case
  case 4: //  Fall to through to the next case
  case 5: cout << "Weekday"; break;
  case 0: // Fall to through to the next case
  case 6: cout << "Weekend";
}
```

TRACE SWITCH STATEMENT

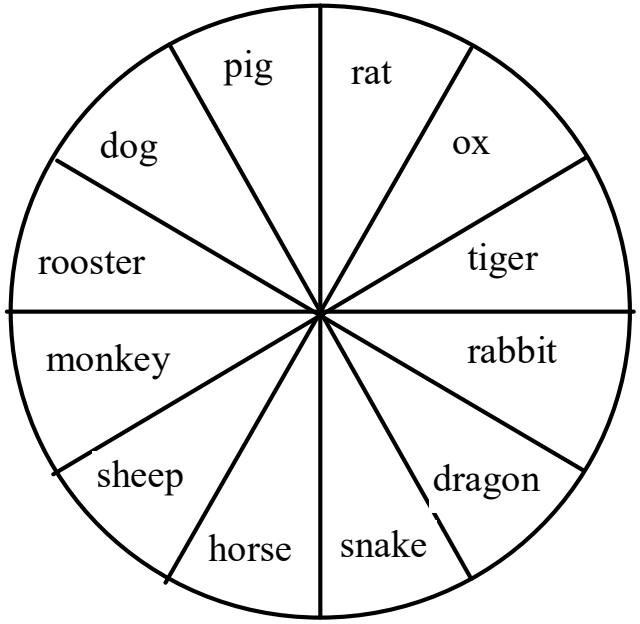
Suppose day is 3:

```
switch (day)
{
  case 1: // Fall to through to the next case
  case 2: // Fall to through to the next case
  case 3: // Fall to through to the next case
  case 4: // Fall to through to the next case
  case 5: cout << "Weekday"; break;
  case 0: // Fall to through to the next case
  case 6: cout << "Weekend";
}
```



PROBLEM: CHINESE ZODIAC

Write a program that prompts the user to enter a year and displays the animal for the year.



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

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Enter a year: ";
  int year;
  cin >> year;
  switch (year % 12)
  {
    case 0: cout << "monkey" << endl; break;
    case 1: cout << "rooster" << endl; break;
    case 2: cout << "dog" << endl; break;
    case 3: cout << "pig" << endl; break;
    case 4: cout << "rat" << endl; break;
    case 5: cout << "ox" << endl; break;
    case 6: cout << "tiger" << endl; break;
    case 7: cout << "rabbit" << endl; break;
    case 8: cout << "dragon" << endl; break;
    case 9: cout << "snake" << endl; break;
    case 10: cout << "horse" << endl; break;
    case 11: cout << "sheep" << endl; break;
  }
  return 0;
}
```

CONDITIONAL OPERATOR

```
if (x > 0)
  y = 1
else
  y = -1;
```

is equivalent to

```
y = (x > 0) ? 1 : -1;
```

(booleanExpression) ? expression1 : expression2

Ternary operator
Binary operator
Unary operator

CONDITIONAL OPERATOR

```
cout << ((num % 2 == 0) ? "num is even" :
   "num is odd");
```

CONDITIONAL OPERATOR, CONT.

```
(booleanExp) ? exp1 : exp2
```

OPERATOR PRECEDENCE

How to evaluate $3 + 4 * 4 > 5 * (4 + 3) - 1$?

OPERATOR PRECEDENCE

- var++, var--
- +, - (Unary plus and minus), ++var,--var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Comparison)
- ==, !=; (Equality)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR
- =, +=, -=, *=, /=, %= (Assignment operator)

ENUMERATED TYPES

enum Day {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY};

Once a type is defined, you can declare a variable of that type:

```
Day day;
```

The variable day can hold one of the values defined in the enumerated type. For example, the following statement assigns enumerated value MONDAY to variable day:

```
day = MONDAY;
```

ENUMERATED TYPES

As with any other type, you can declare and initialize a variable in one statement:

```
Day day = MONDAY;
```

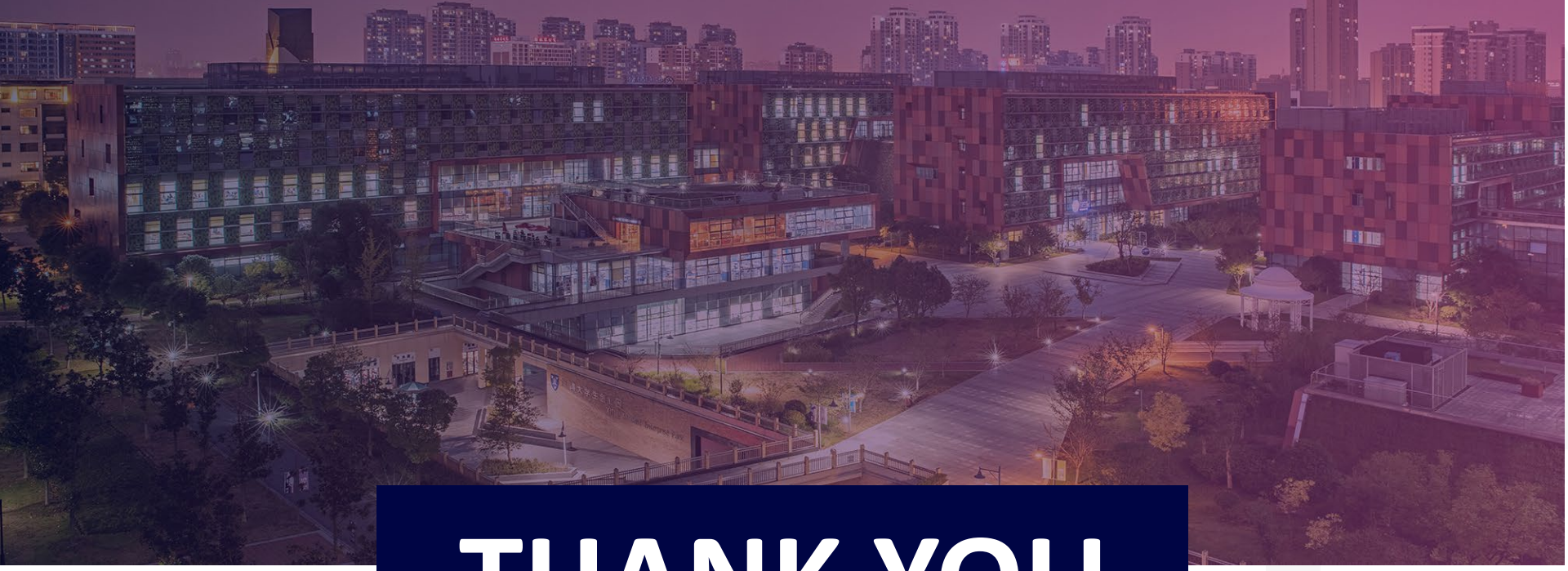
Furthermore, C++ allows you to declare an enumerated type and variable in one statement. For example,

```
enum Day {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY} day = MONDAY;
```

```
#include <iostream>
using namespace std;
int main()
{
  enum Day {MONDAY = 1, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY} day;
  cout << "Enter a day (1 for Monday, 2 for Tuesday, etc): ";
  int dayNumber;
  cin >> dayNumber;
  switch (dayNumber) {
    case MONDAY:
      cout << "Play soccer" << endl;
      break;
    case TUESDAY:
      cout << "Piano lesson" << endl;
      break;
    case WEDNESDAY:
      cout << "Math team" << endl;
      break;
    default:
      cout << "Go home" << endl;
  }
  return 0;
}
```

SUMMARIZATION

- input from the keyboard
- numeric data types
- write integer literals, floating-point literals, and literals in scientific notation
- write and evaluate expressions
- software development process



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

Xi'an Jiaotong-Liverpool University
西交利物浦大学