# C++ Programming

Jingping Li
(Simon)
xxtjingping@buu.edu.cn

# Introduction

**Procedural, Structured, and Object-Oriented Programming**

- **Procedural or Structured**
  - A computer program can be thought of as consisting of a set of tasks.
  - Structured programming remains an enormously successful approach for dealing with complex problems.

# Introduction

  - First, it is natural to think of your data (employee records, for example) and what you can do with your data (sort, edit, and so on) as related ideas.
  - Second, programmers found themselves constantly reinventing new solutions to old problems.
- *Event-driven*
  - means that an event happens--the user presses a button or chooses from a menu--and the program must respond.

# Introduction

- object-oriented programming (OOP)is to treat data and the procedures that act upon the data as a single "object"--a self-contained entity with an identity and certain characteristics of its own.
- "data controlling access to code"

# C++ and Object-Oriented Programming

- C++ fully supports object-oriented programming, including the four pillars of object-oriented development: encapsulation, data hiding, inheritance, and polymorphism.
- With encapsulation, we can accomplish data hiding. Data hiding is the highly valued characteristic that an object can be used without the user knowing or caring how it works internally.

# C++ and Object-Oriented Programming

- C++ supports the idea of reuse through inheritance.
  - A new type, which is an extension of an existing type, can be declared. This new subclass is said to derive from the existing type and is sometimes called a derived type.
- different objects do "the right thing" through what is called function polymorphism and class polymorphism.

# Creating an Executable File

The steps to create an executable file are

**1.** Create a source code file, with a .CPP extension(txt file).

**2.** Compile the source code into a file with the .OBJ extension(binary file).

**3.** Link your OBJ file with any needed libraries to produce an executable program.

# First program

```
1: #include <iostream.h>
2:
3: int main()
4: {
5: cout << "Hello World!\n";
6: return 0;
7: }
```

Note: COMPILE---LINK---RUN

# Question and answer

**Q. Can a program run even if has a warning?**
**Q. Can I ignore warning messages from my compiler?**

A. Many books hedge on this one, but I'll stake myself to this position: No! Get into the habit, from day one, of treating warning messages as errors.

C++ uses the compiler to warn you when you are doing something you may not intend. Heed those warnings, and do what is required to make them go away.

# parts of a C++ program

```
1: #include <iostream.h>
2:
3: int main()
4: {
5: cout << "Hello World!\n";
6: return 0;
7: }
```

# A Brief Look at cout

```
3: #include <iostream.h>
4: int main()
5: {
6: cout << "Hello there.\n";
7: cout << "Here is 5: " << 5 << "\n";
8: cout << "The manipulator endl writes a new line to the screen." << endl;
9: cout << "Here is a very big number:\t" << 70000 << endl;
10: cout << "Here is the sum of 8 and 5:\t" << 8+5 << endl;
11: cout << "Here's a fraction:\t\t" << (float) 5/8 << endl;
12: cout << "And a very very big number:\t" << (double) 7000 * 7000 << endl;
13: cout << "Don't forget to replace Jesse Liberty with your name...\n";
14: cout << "Jesse Liberty is a C++ programmer!\n";
15: return 0;
16: }
```

To print a value to the screen, write the word cout typing the less-than character (<) twice

# Comments----before function

```
/*************************************************
Program: Hello World
File: Hello.cpp
Function: Main (complete program listing in this file)
Description: Prints the words "Hello world" to the screen
Author: Jesse Liberty (jl)
Environment: Turbo C++ version 4, 486/66 32mb RAM,
Windows 3.1
DOS 6.0. EasyWin module.
*************************************************/
```

# Variable

- How to declare and define variables and constants.
  - The role of a variable in programm.
- How to assign values to variables and manipulate those values.--- variable's name
- How to write the value of a variable to the screen.

# Enumerated Constants

- Enumerated constants enable you to create new types and then to define variables of those types whose values are restricted to a set of possible values.
- enum COLOR { RED, BLUE, GREEN, WHITE, BLACK };

# Enumerated Constants

This statement performs two tasks:

**1.** It makes COLOR the name of an enumeration, that is, a new type.

**2.** It makes RED a symbolic constant with the value 0, BLUE a symbolic constant with the value 1, GREEN a symbolic constant with the value 2, and so forth.

Every enumerated constant has an integer value.

# Enumerated Constants

- Any one of the constants can be initialized with a particular value, however, and those that are not initialized will count upward from the ones before them.

enum Color { RED=100, BLUE, GREEN=500, WHITE, BLACK=700 };

- then RED will have the value 100; BLUE, the value 101; GREEN, the value 500; WHITE, the value 501; and BLACK, the value 700.

```
#include <iostream.h>
int main()
{
	enum Days { Sunday, Monday, Tuesday, Wednesday,
Thursday, Friday, Saturday };
	Days DayOff;
	int x;
	cout << "What day would you like off (0-6)? ";
	cin >> x;
	DayOff = Days(x);
	if (DayOff == Sunday || DayOff == Saturday)
		cout << "\nYou're already off on weekends!\n";
	else
		cout << "\nOkay, I'll put in the vacation day.\n";
	return 0;
}
```

# Results

Output:

What day would you like off (0-6)? 1

Okay, I'll put in the vacation day.

What day would you like off (0-6)? 0

You're already off on weekends!

What day would you like off (0-6)? 3

?

What day would you like off (0-6)?6

# Expressions and Statements

- Statements
  - In C++ a statement controls the sequence of execution, evaluates an expression, or does nothing (the null statement).
  - All C++ statements end with a semicolon, even the null statement, which is just the semicolon and nothing else.

```
x = a + b;
```

# Blocks and Compound Statements

- Any place you can put a single statement, you can put a compound statement, also called a block.
- A block begins with an opening brace ({) and ends with a closing brace (}).
- Although every statement in the block must end with a semicolon, the block itself does not end with a semicolon.
- A block of code acts as one statement

# Expressions

- Expression is a legal set of operators and operands.
- Anything that can result to a value is an expression.
- In other word any expression has a value.
  - 3.14
  - x = a + b
  - y = x = a + b

# Operators

- **Assignment Operator =**
- **Mathematical Operators**

There are five mathematical operators: addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).

- **Combining the Assignment and Mathematical Operators**
- There are self-assigned subtraction (-=), division (/=), multiplication (*=), and modulus (%=) operators as well.

# Precedence

- x = 5 + 3 + 72 + 24
- TotalSeconds = NumMinutesToThink + NumMinutesToType * 60
- **Nesting Parentheses**
- complicated expression is read from the inside out

# 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`

# Relational Operators (continued)

**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

**TABLE 4-2** Evaluating Expressions Using Relational Operators and the ASCII Collating Sequence

| Expression | Value of Expression | Explanation |
|---|---|---|
| ' ' < 'a' | true | The ASCII value of ' ' is 32, and the ASCII value of 'a' is 97. Because 32 < 97 is true, it follows that ' ' < 'a' is true. |
| 'R' > 'T' | false | The ASCII value of 'R' is 82, and the ASCII value of 'T' is 84. Because 82 > 84 is false, it follows that 'R' > 'T' is false. |
| '+' < '*' | false | The ASCII value of '+' is 43, and the ASCII value of '*' is 42. Because 43 < 42 is false, it follows that '+' < '*' is false. |
| '6' <= '>' | true | The ASCII value of '6' is 54, and the ASCII value of '>' is 62. Because 54 <= 62 is true, it follows that '6' <= '>' is true. |

# Logical Operators

| *Operator* | *Symbol* | *Example* |
|---|---|---|
| AND | && | expression1 && expression2 |
| OR | \|\| | expression1 \|\| expression2 |
| NOT | ! | !expression |

Imagine a sophisticated alarm system that has this logic: "If the door alarm sounds AND it is after six p.m. AND it is NOT a holiday, OR if it is a weekend, then call the police."

# Increment and Decrement

- increment operator (++) and the decrement operator(--)
- **Prefix and Postfix**

The prefix operator is evaluated before the assignment, the postfix is evaluated after.

a = ++x

b = x++

# The Nature of Truth

- In C++, zero is considered false, and all other values (not zero)are considered true, although true is usually represented by 1.
- Especially some of the results of an expression

```
X>=y //if x is equal to y, the result is 1
X&&y //if x and y are true, the result is 1
X==5// if x has the value of 5, the result is 1
     //if x is not 5, the result is 0
```

# 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 (continued)

**TABLE 4-9** Precedence of Operators

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

# Order of Precedence (continued)

**EXAMPLE 4-5**

Suppose you have the following declarations:

```
bool found = true;
bool flag = false;
int num = 1;
double x = 5.2;
double y = 3.4;
int a = 5, b = 8;
int n = 20;
char ch = 'B';
```

# Order of Precedence (continued)

| Expression | Value | Explanation |
|---|---|---|
| `!found` | `false` | Because `found` is `true`, `!found` is `false`. |
| `x > 4.0` | `true` | Because `x` is `5.2` and `5.2 > 4.0` is `true`, the expression `x > 4.0` evaluates to `true`. |
| `!num` | `false` | Because `num` is `1`, which is nonzero, `num` is `true` and so `!num` is `false`. |
| `!found && (x >= 0)` | `false` | In this expression, `!found` is `false`. Also, because `x` is `5.2` and `5.2 >= 0` is `true`, `x >= 0` is `true`. Therefore, the value of the expression `!found && (x >= 0)` is `false && true`, which evaluates to `false`. |
| `!(found && (x >= 0))` | `false` | In this expression, `found && (x >= 0)` is `true && true`, which evaluates to `true`. Therefore, the value of the expression `!(found && (x >= 0))` is `!true`, which evaluates to `false`. |
| `x + y <= 20.5` | `true` | Because `x + y = 5.2 + 3.4 = 8.6` and `8.6 <= 20.5`, it follows that `x + y <= 20.5` evaluates to `true`. |

# Order of Precedence (continued)

| Expression | Value | Explanation |
|---|---|---|
| (n >= 0) && (n <= 100) | true | Here n is 20. Because 20 >= 0 is true, n >= 0 is true. Also, because 20 <= 100 is true, n <= 100 is true. Therefore, the value of the expression (n >= 0) && (n <= 100) is true && true, which evaluates to true. |
| ('A' <= ch && ch <= 'Z') | true | In this expression, the value of 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, the value of the expression ('A' <= ch && ch <= 'Z') is true && true, which evaluates to true. |
| (a + 2 <= b) && !flag | true | Now a + 2 = 5 + 2 = 7 and b is 8. Because 7 <= 8 is true, the expression a + 2 <= b evaluates to true. Also, because flag is false, !flag is true. Therefore, the value of the expression (a + 2 <= b) && !flag is true && true, which evaluates to true. |

# Short-Circuit Evaluation

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

```
(age >= 21) || ( x == 5)  //Line 1
(grade == 'A') && (x >= 7)  //Line 2
```

# Conditional (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`
- This line is read as "If expression1 is true, return the value of expression2; otherwise, return the value of expression3." Typically, this value would be assigned to a variable.

# `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
  - The value of a logical expression was stored in a variable of the data type `int`
- You can use the `int` data type to manipulate logical (Boolean) expressions

# The `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`

# Logical (Boolean) Expressions

- Logical expressions can be unpredictable
- The following expression appears to represent a comparison of `0`, `num`, and `10`:

```
0 <= num <= 10
```

- It always evaluates to `true` because `0 <= num` evaluates to either `0` or `1`, and `0 <= 10` is `true` and `1 <= 10` is `true`
- A correct way to write this expression is:

```
0 <= num && num <= 10
```

# Type Conversion in Expressions

- When constants and variables of different types are mixed in an expression, they are all converted to the same type. The compiler converts all operands up to the type of the largest operand, which is called *type promotion*.

IF an operand is a **long double**

THEN the second is converted to **long double**

ELSE IF an operand is a **double**

THEN the second is converted to **double**

ELSE IF an operand is a **float**

THEN the second is converted to **float**

ELSE IF an operand is an **unsigned long**

THEN the second is converted to **unsigned long**

ELSE IF an operand is **long**

THEN the second is converted to **long**

ELSE IF an operand is **unsigned int**

THEN the second is converted to **unsigned int**

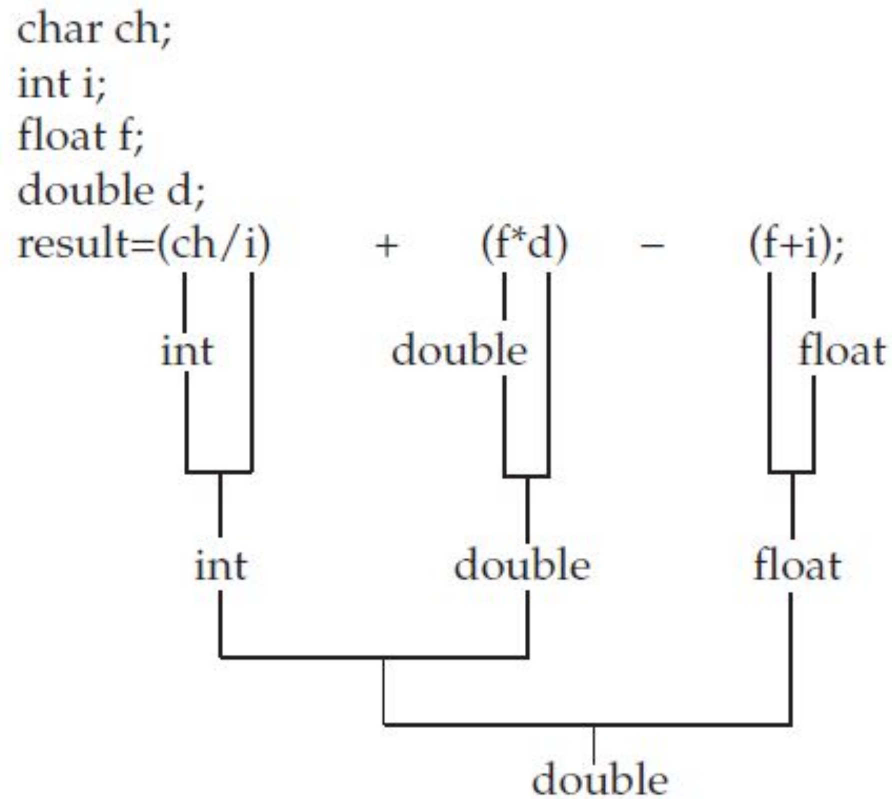



**Figure 2-2.** *A type conversion example*

# The Comma Operator

- The comma operator strings together several expressions. The left side of the comma operator is always evaluated as **void**. This means that the expression on the right side becomes the value of the total comma-separated expression.

```
x = (y=3, y+1);
```

What is the value of x?

# Bitwise Operators

- *Bitwise operation* refers to testing, setting, or shifting the actual bits in a byte or word, which correspond to the **char** and **int** data types and variants.
- The operations are applied to the individual bits of the operands.
- the bitwise operations can be used to mask off certain bits, such as parity.

<table>
<tr><th>Operator</th><th>Action</th></tr>
<tr><td>&</td><td>AND</td></tr>
<tr><td>|</td><td>OR</td></tr>
<tr><td>^</td><td>Exclusive OR (XOR)</td></tr>
<tr><td>~</td><td>One's complement (NOT)</td></tr>
<tr><td>>></td><td>Shift right</td></tr>
<tr><td><<</td><td>Shift left</td></tr>
</table>

**Table 2-6.** *Bitwise Operators*

```
char get_char_from_modem(void)
{
char ch;
ch = read_modem(); /* get a character from the
modem port */
return(ch & 127);
}
```

```
Parity bit
↓
1 1 0 0 0 0 0 1     ch containing an "A" with parity set
0 1 1 1 1 1 1 1     127 in binary
&_______________    bitwise AND
0 1 0 0 0 0 0 1     "A" without parity
```

- The bitwise OR, as the reverse of AND, can be used to set a bit.
- An exclusive OR, usually abbreviated XOR, will set a bit on if and only if the bits being compared are different.
- The bit-shift operators, >> and <<, move all bits in a value to the right or left as specified.

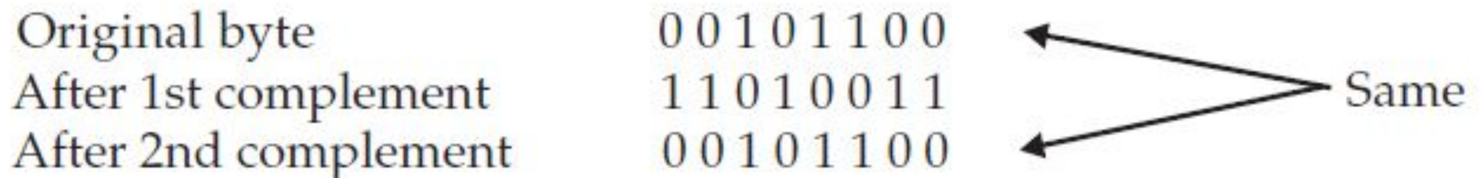
| unsigned char x; | x as each statement executes | value of x |
|---|---|---|
| x = 7; | 0 0 0 0 0 1 1 1 | 7 |
| x = x<<1; | 0 0 0 0 1 1 1 0 | 14 |
| x = x<<3; | 0 1 1 1 0 0 0 0 | 112 |
| x = x<<2; | 1 1 0 0 0 0 0 0 | 192 |
| x = x>>1; | 0 1 1 0 0 0 0 0 | 96 |
| x = x>>2; | 0 0 0 1 1 0 0 0 | 24 |

*Each left shift multiplies by 2. Notice that information has been lost after x<<2 because a bit was shifted off the end.

**Each right shift divides by 2. Notice that subsequent divisions do not bring back any lost bits.

**Table 2-7.** *Multiplication and Division with Shift Operators*

- The one's complement operator, ~, reverses the state of each bit in its operand. That is, all 1's are set to 0, and all 0's are set to 1.



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

- One-Way Selection
- Two-Way Selection
- Compound (Block of) Statements
- Multiple Selections: Nested `if`
- Comparing `if...else` Statements with a Series of `if` Statements

# Selection: `if` and `if...else` (continued)

- Using Pseudocode to Develop, Test, and Debug a Program
- Input Failure and the `if` Statement
- Confusion Between the Equality Operator (==) and the Assignment Operator (=)
- Conditional Operator (?:)

# 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 (continued)

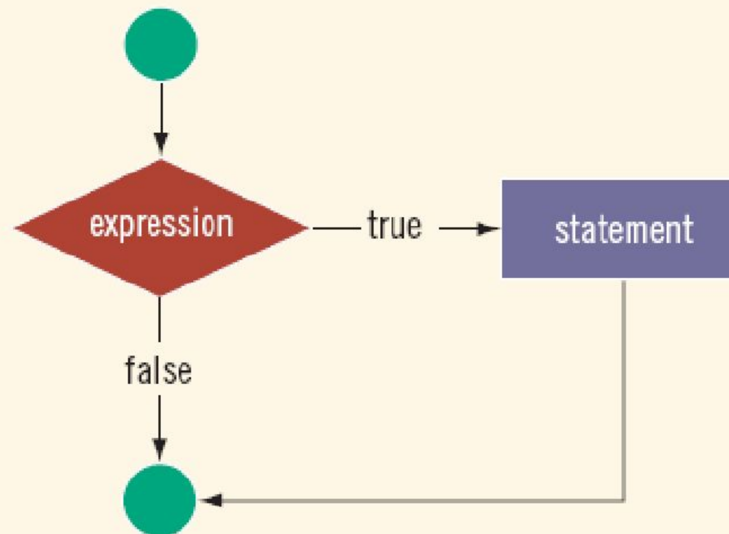



FIGURE 4-2 One-way selection

# One-Way Selection (continued)

**EXAMPLE 4-9**

```
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 4-10

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 (continued)

**EXAMPLE 4-11**

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 4-12**

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 (continued)

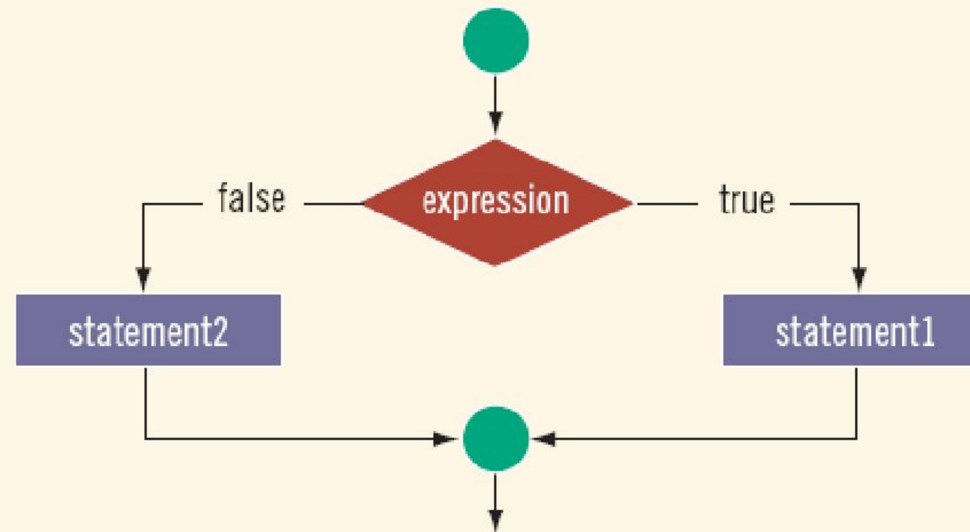



**FIGURE 4-3** Two-way selection

# Two-Way Selection (continued)

**EXAMPLE 4-13**

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`, then 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 (continued)

**EXAMPLE 4-14**

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.

# Compound (Block of) Statement

- Compound statement (block of statements):

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

- A compound statement is a single statement

# Compound (Block of) Statement (continued)

```
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;
}
```

# Multiple Selections: Nested `if`

- Nesting: one control statement in another
- An `else` is associated with the most recent `if` that has not been paired with an `else`

## EXAMPLE 4-18

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
```

To avoid excessive indentation, the code in Example 4-18 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` (continued)

**EXAMPLE 4-19**

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;
```

# Comparing `if...else` Statements with a Series of `if` Statements

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;
```

# Using Pseudocode to Develop, Test, and Debug a Program

- Pseudocode (pseudo): provides a useful means to outline and refine a program before putting it into formal C++ code
- You must first develop a program using paper and pencil
- On paper, it is easier to spot errors and improve the program
  - Especially with large programs

# 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

# Confusion Between == and =

- 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

# `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
}
```

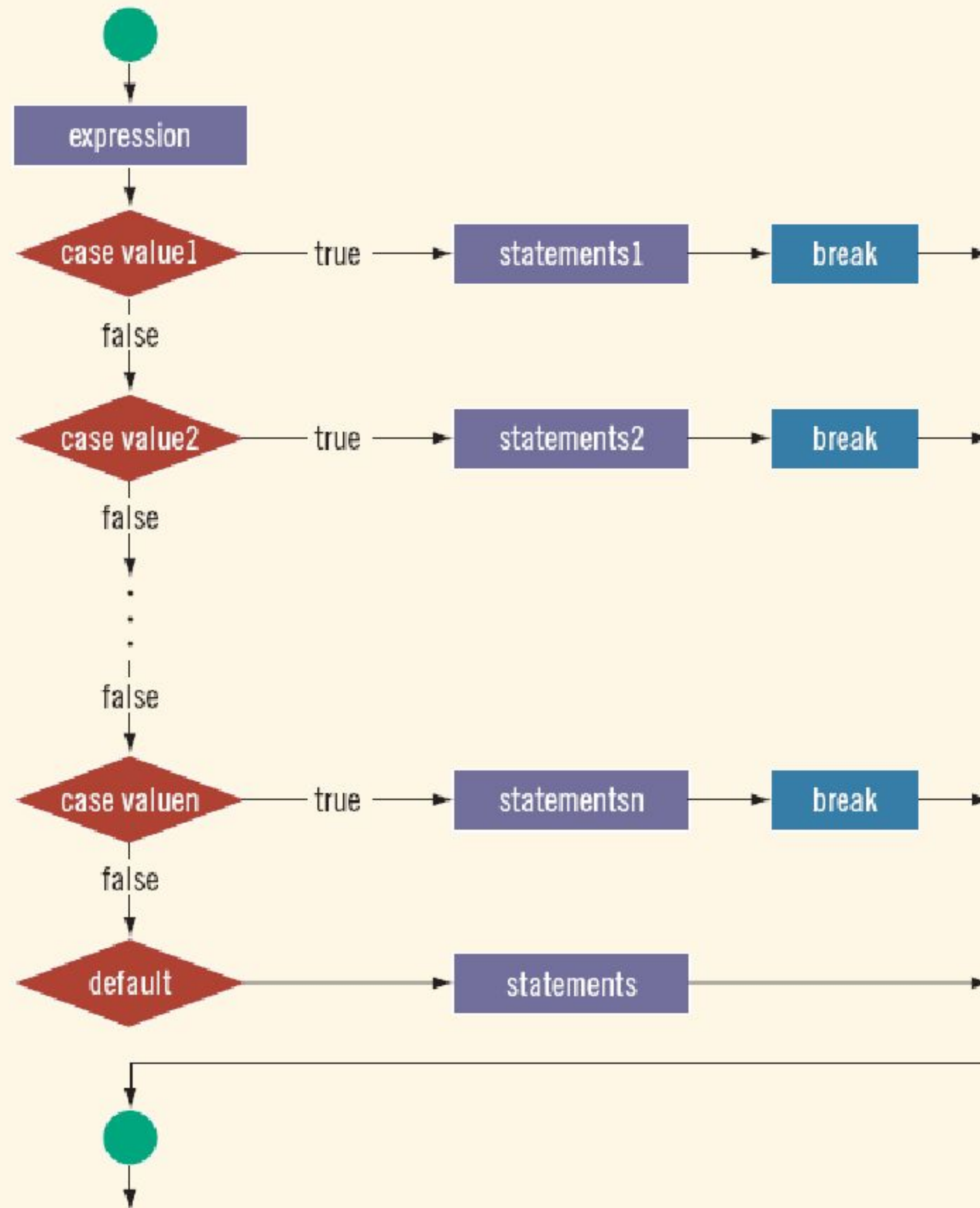



FIGURE 4-4 `switch` statement

# `switch` Structures (continued)

- 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 4-24

Consider the following statements, where grade is a variable of type `char`:

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

# 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 (continued)

- 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 (continued)

- `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

# Programming Example: Cable Company Billing

- This programming example calculates a customer's bill for a local cable company
- There are two types of customers:
  - Residential
  - Business
- Two rates for calculating a cable bill:
  - One for residential customers
  - One for business customers

# Programming Example: Rates

- For residential customer:
  - Bill processing fee: $4.50
  - Basic service fee: $20.50
  - Premium channel: $7.50 per channel
- For business customer:
  - Bill processing fee: $15.00
  - Basic service fee: $75.00 for first 10 connections and $5.00 for each additional connection
  - Premium channel cost: $50.00 per channel for any number of connections

# Programming Example: Requirements

- Ask user for account number and customer code
- Assume `R` or `r` stands for residential customer and `B` or `b` stands for business customer

# Programming Example: Input and Output

- Input:
  - Customer account number
  - Customer code
  - Number of premium channels
  - For business customers, number of basic service connections
- Output:
  - Customer’s account number
  - Billing amount

# Programming Example: Program Analysis

- Purpose: calculate and print billing amount
- Calculating billing amount requires:
  - Customer for whom the billing amount is calculated (residential or business)
  - Number of premium channels to which the customer subscribes
- For a business customer, you need:
  - Number of basic service connections
  - Number of premium channels

# Programming Example: Program Analysis (continued)

- Data needed to calculate the bill, such as bill processing fees and the cost of a premium channel, are known quantities
- The program should print the billing amount to two decimal places

# Programming Example: Algorithm Design

- Set precision to two decimal places
- Prompt user for account number and customer type
- If customer type is `R` or `r`
  - Prompt user for number of premium channels
  - Compute and print the bill
- If customer type is `B` or `b`
  - Prompt user for number of basic service connections and number of premium channels
  - Compute and print the bill

# Programming Example: Variables and Named Constants

```
int accountNumber;  //variable to store the customer's
                    //account number
char customerType;     //variable to store the customer code
int numOfPremChannels; //variable to store the number
                       //of premium channels to which the
                       //customer subscribes
int numOfBasicServConn; //variable to store the
                  //number of basic service connections
                  //to which the customer subscribes
double amountDue;  //variable to store the billing amount
```

```
      //Named constants - residential customers
const double RES_BILL_PROC_FEES = 4.50;
const double RES_BASIC_SERV_COST = 20.50;
const double RES_COST_PREM_CHANNEL = 7.50;

      //Named constants - business customers
const double BUS_BILL_PROC_FEES = 15.00;
const double BUS_BASIC_SERV_COST = 75.00;
const double BUS_BASIC_CONN_COST = 5.00;
const double BUS_COST_PREM_CHANNEL = 50.00;
```

# Programming Example: Formulas

Billing for residential customers:

```
amountDue = RES_BILL_PROC_FEES +
            RES_BASIC_SERV_COST
            + numOfPremChannels *
            RES_COST_PREM_CHANNEL;
```

# Programming Example: Formulas (continued)

Billing for business customers:

```
if (numOfBasicServConn <= 10)
    amountDue = BUS_BILL_PROC_FEES +
                BUS_BASIC_SERV_COST
                + numOfPremChannels *
                  BUS_COST_PREM_CHANNEL;
else
    amountDue = BUS_BILL_PROC_FEES +
                BUS_BASIC_SERV_COST
                + (numOfBasicServConn - 10)
                 * BUS_BASIC_CONN_COST
               + numOfPremChannels *
                 BUS_COST_PREM_CHANNEL;
```

# Programming Example: Main Algorithm

1. Output floating-point numbers in fixed decimal with decimal point and trailing zeros
   - Output floating-point numbers with two decimal places and set the precision to two decimal places
2. Prompt user to enter account number
3. Get customer account number
4. Prompt user to enter customer code
5. Get customer code

# Programming Example: Main Algorithm (continued)

6. If the customer code is `r` or `R`,
   - Prompt user to enter number of premium channels
   - Get the number of premium channels
   - Calculate the billing amount
   - Print account number and billing amount

# Programming Example: Main Algorithm (continued)

7. If customer code is `b` or `B`,
   - Prompt user to enter number of basic service connections
   - Get number of basic service connections
   - Prompt user to enter number of premium channels
   - Get number of premium channels
   - Calculate billing amount
   - Print account number and billing amount

# Programming Example: Main Algorithm (continued)

8. If customer code is other than `r`, `R`, `b`, or `B`, output an error message

# Looping

- **The while Statement**
- The syntax for the while statement is as follows:

```
while ( condition )
    statement;
```

- condition is any C++ expression, and statement is any valid C++ statement or block of statements.

# The do...while Statement

- The syntax for the do...while statement is as follows:

```
do
statement
while (condition);
// count to 10
int x = 0;
do
cout << "X: " << x++;
while (x < 10)
```

use `do...while` when you want to ensure the loop is executed at least once.

# for Loops

The syntax for the for statement is as follows:

for (initialization; test; action ) statement;

- for (counter = 0; counter < 5; counter++)

A for loop works in the following sequence:

- **1.** Performs the operations in the initialization.
- **2.** Evaluates the condition.
- **3.** If the condition is TRUE, executes the action statement and the loop.

example

```
for (int i = 0; i < 10; i++)
{
    cout << "Hello!" << endl;
    cout << "the value of i is: " << i << endl;
}
for (int i=0, j=0; i<3; i++, j++)
for( ; counter < 5; )
for (;;)
```

# Empty for Loops

```
1: //Listing 7.13
2: //Demonstrates null statement
3: // as body of for loop
4:
5: #include <iostream.h>
6: int main()
7: {
8: for (int i = 0; i<5; cout << "i: " << i++ << endl)
9: ;
10: return 0;
11: }
```

# continue and break

- The continue statement jumps back to the top of the loop.
- break; causes the immediate end of a while or for loop.

# example:continue

```
int values[10];
...
// Print the nonzero elements of the array.
for (int i = 0; i < 10; ++i) {
    if (values[i] == 0) {
// Skip over zero elements.
        continue;
    }
// Print the (nonzero) element.
    std::cout << values[i] << '\n';
}
```

# Example: break

example:

```
// Read integers from standard input until an
// error or end-of-file is encountered or a
// negative integer is read.
int x;
while (std::cin >> x) {
    if (x < 0) {
        break;
    }
    std::cout << x << '\n';
}
```

# Example: goto

```
int i = 0;
loop: // label for goto statement
do {
    if (i == 3) {
        ++i;
        goto loop;
    }
    std::cout << i << ’\n’;
    ++i;
} while (i < 10);
```

# Summary

- Control structures alter normal control flow
- Most common control structures are selection and repetition
- Relational operators: ==, <, <=, >, >=, !=
- Logical expressions evaluate to 1 (true) or 0 (false)
- Logical operators: ! (not), && (and), || (or)

# Summary (continued)

- Two selection structures: one-way selection and two-way selection
- The expression in an `if` or `if...else` structure is usually a logical expression
- No stand-alone `else` statement in C++
  - Every `else` has a related `if`
- A sequence of statements enclosed between braces, `{` and `}`, is called a compound statement or block of statements

# Summary (continued)

- Using assignment in place of the equality operator creates a semantic error
- `switch` structure handles multiway selection
- `break` statement ends `switch` statement
- Use `assert` to terminate a program if certain conditions are not met