5

Control Statements: Part 2

东南大学软件学院

OBJECTIVES

In this chapter you'll learn:

- To use the `for` and `do…while` repetition statements to execute statements in a program repeatedly.
- To understand multiple selection using the `switch` selection statement.
- To use the `break` and `continue` program control statements to alter the flow of control.
- To use the logical operators to form complex conditional expressions in control statements.

5.2 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires:
  1. Name of a control variable (loop counter)
  2. Initial value of the control variable
  3. Loop-continuation condition that tests for the final value of the control variable
  4. Increment/decrement of control variable at each iteration

3

```
int main()
{
  int counter = 1; // 1. and 2. control variable

  while ( counter <= 10 ) // 3. loop-
continuationcondition
  {
      cout << counter << " ";
      counter++; // 4. increment control variable by 1
  }

  cout << endl;
  return 0;
}
```

4

Common Programming Error 5.1

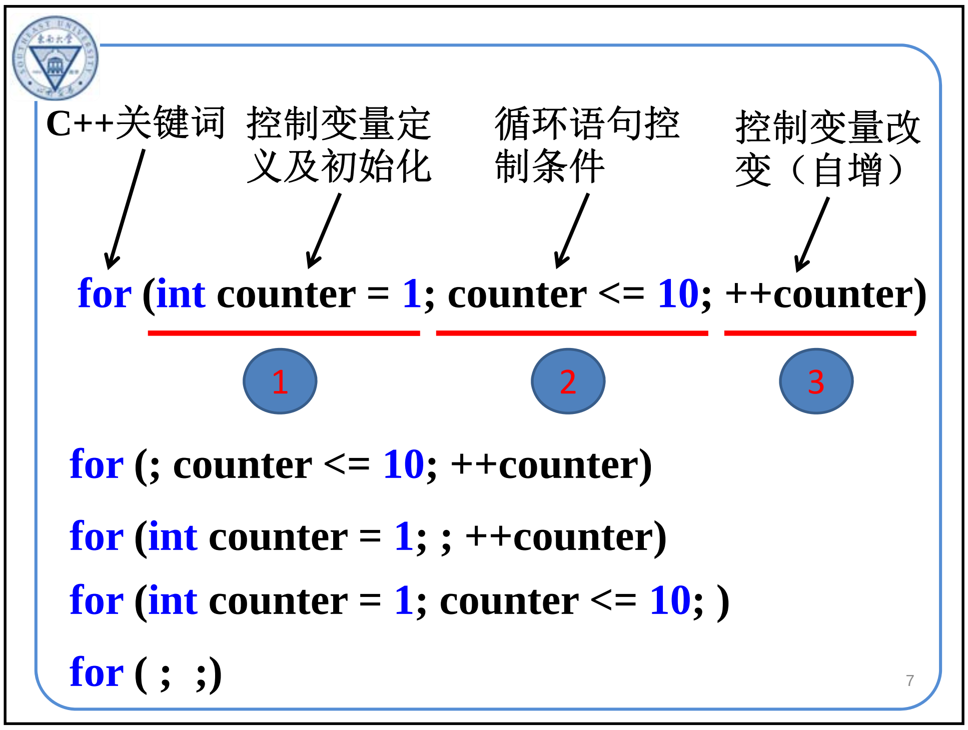
Floating-point values are approximate, so controlling counting loops with floating-point variables can result in imprecise counter values and inaccurate tests for termination. 循环控制变量不能使用浮点数。

5.3 for Repetition Statement

for repetition statement

– Specifies counter-controlled repetition details in a single line of code

```
int main()
{
   for ( int counter = 1; counter <= 10; counter++ )
      cout << counter << " ";

   cout << endl; // output a newline
   return 0; // indicate successful termination
}
```



C++关键词　控制变量定义及初始化　循环语句控制条件　控制变量改变（自增）

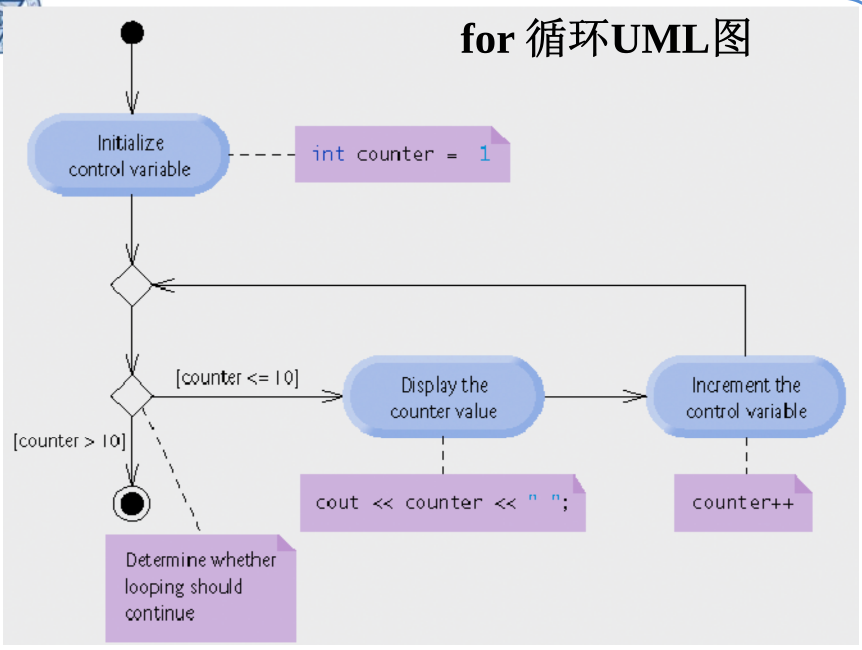
```
for (int counter = 1; counter <= 10; ++counter)
```

1　2　3

```
for (; counter <= 10; ++counter)

for (int counter = 1; ; ++counter)

for (int counter = 1; counter <= 10; )

for ( ;  ;)
```

7

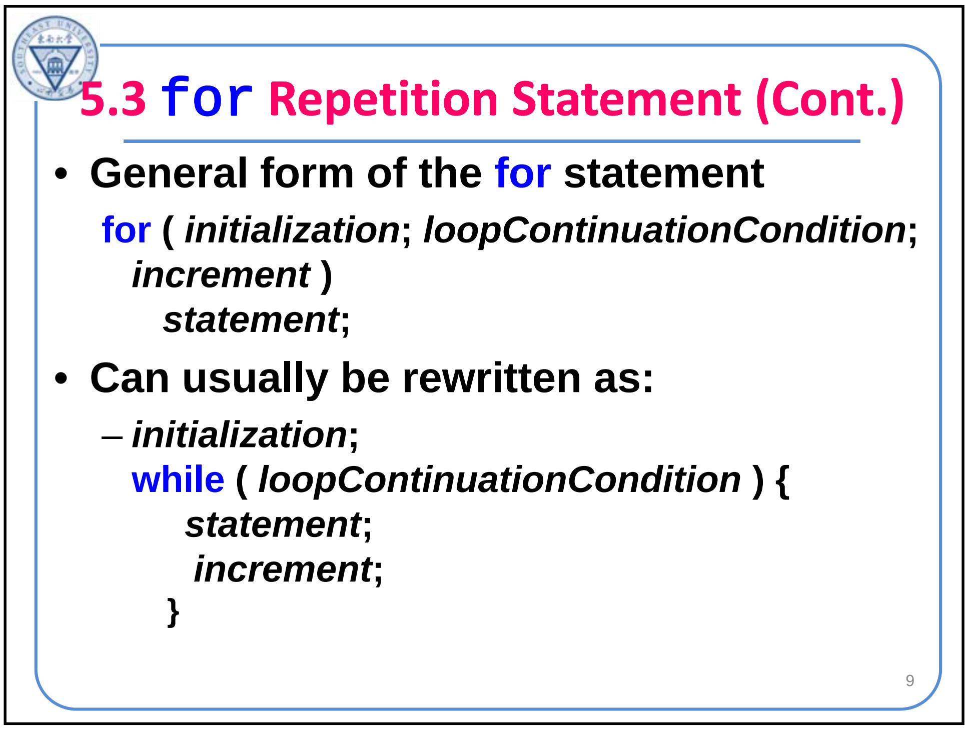
for 循环UML图


Initialize control variable
[counter <= 10]
Display the counter value
Increment the control variable
[counter > 10]
Determine whether looping should continue

```
int counter = 1
```

```
cout << counter << " ";
```

```
counter++
```

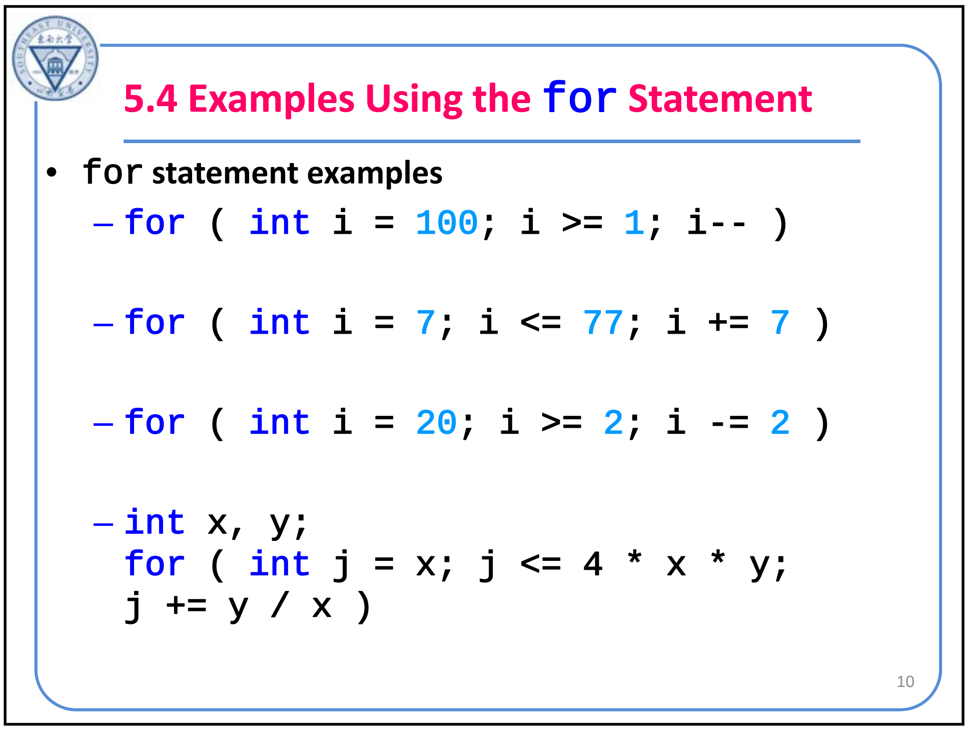


5.3 `for` Repetition Statement (Cont.)

- **General form of the for statement**

```
for ( initialization; loopContinuationCondition;
   increment )
      statement;
```

- **Can usually be rewritten as:**
  - initialization;
```
initialization;
while ( loopContinuationCondition ) {
     statement;
      increment;
   }
```



5.4 Examples Using the `for` Statement

- `for` statement examples
  - `for ( int i = 100; i >= 1; i-- )`
  - `for ( int i = 7; i <= 77; i += 7 )`
  - `for ( int i = 20; i >= 2; i -= 2 )`
  -
```
int x, y;
for ( int j = x; j <= 4 * x * y;
j += y / x )
```

Common Programming Error 5.3

When the control variable of a for statement is declared in the initialization section of the for statement header, using the control variable after the body of the statement is a compilation error.

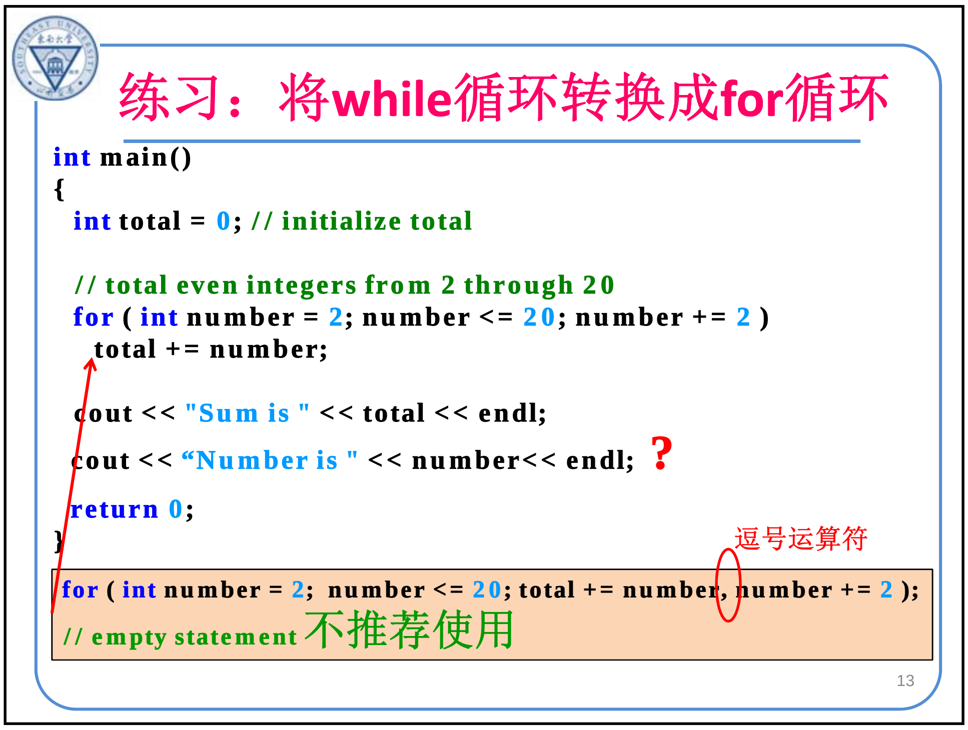
•当控制变量的声明是放在for语句头部的初始化部分，在for语句体之后再使用该控制变量是一个编译错误。

11

练习：将while循环转换成for循环

```
int main()
{
  int total = 0;
   int number = 2;

  // total even integers from 2 through 20
  while(number <= 20)
   {
    total += number;
     number += 2 ;
    }

  cout << "Sum is " << total << endl; // display results
  return 0;
}
```

12



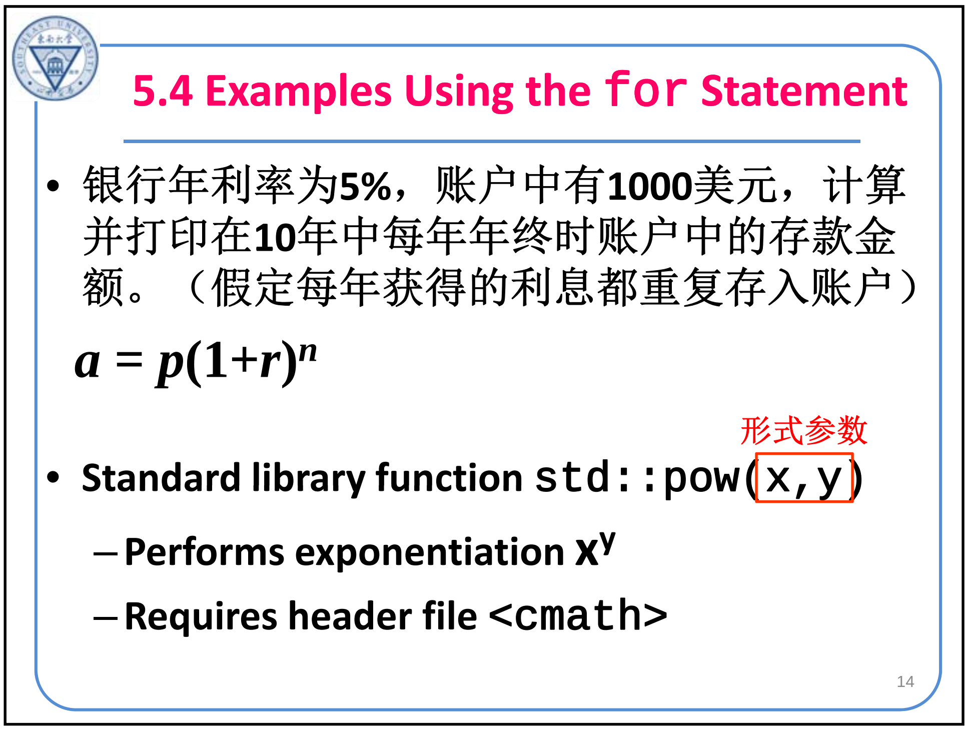
练习：将while循环转换成for循环

```
int main()
{
  int total = 0; // initialize total

  // total even integers from 2 through 20
  for ( int number = 2; number <= 20; number += 2 )
    total += number;

  cout << "Sum is " << total << endl;

  cout << “Number is " << number<< endl;  ?

  return 0;
}
```

逗号运算符

```
for ( int number = 2;  number <= 20; total += number, number += 2 );
// empty statement
```
不推荐使用

13



5.4 Examples Using the `for` Statement

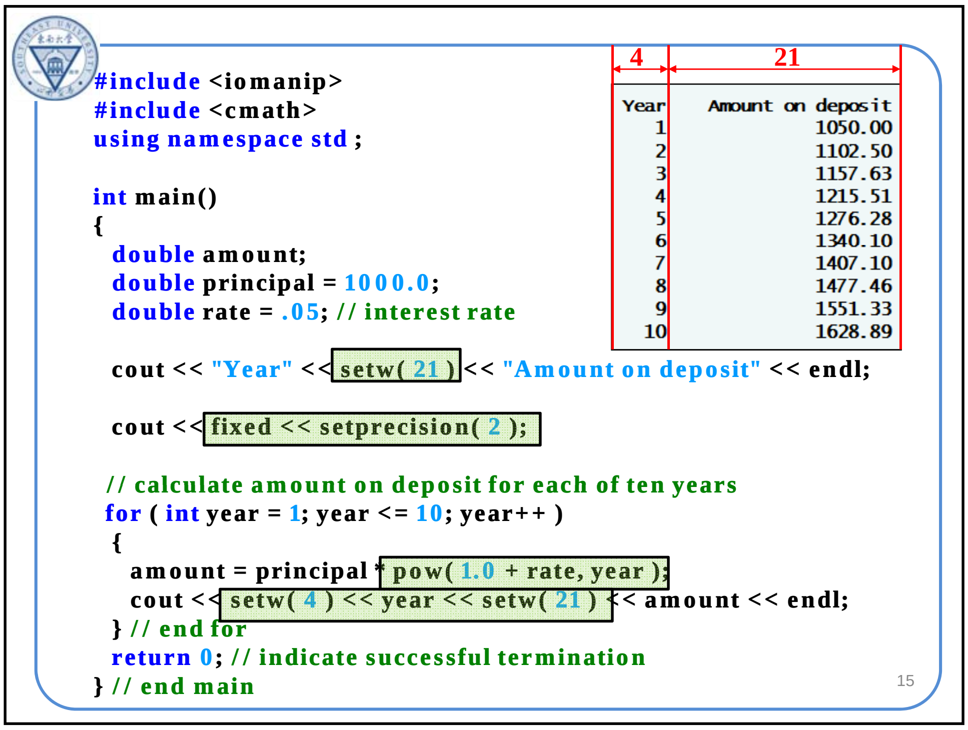
- 银行年利率为**5%**，账户中有**1000**美元，计算并打印在**10**年中每年年终时账户中的存款金额。（假定每年获得的利息都重复存入账户）

$$a = p(1+r)^n$$

形式参数

- **Standard library function `std::pow(x,y)`**
  - **Performs exponentiation x^y**
  - **Requires header file `<cmath>`**

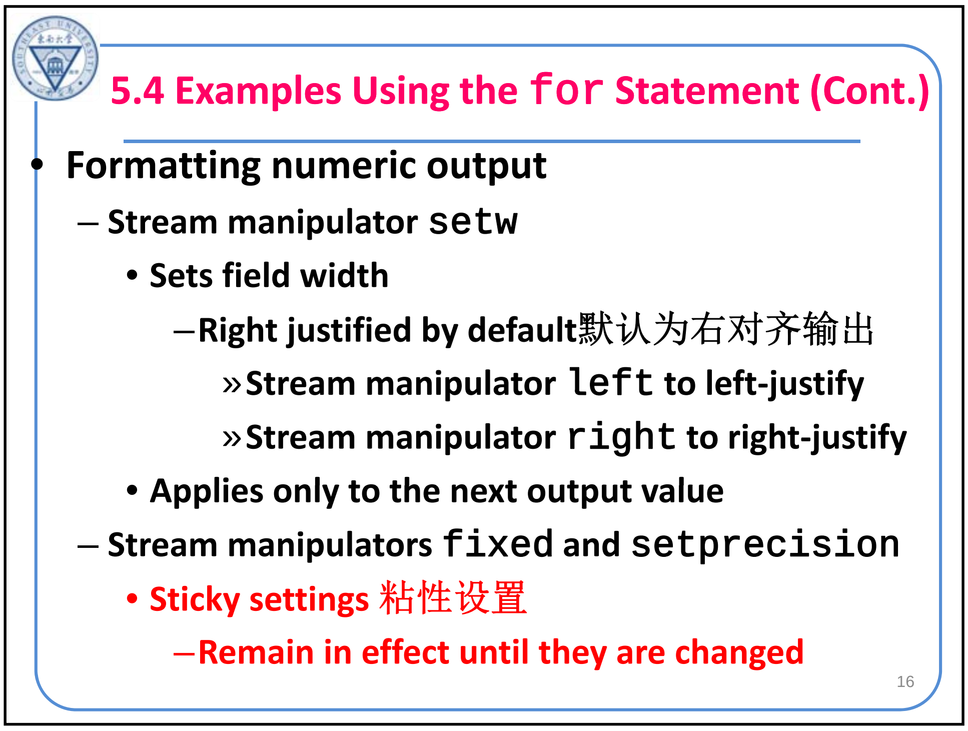
14



```
#include <iomanip>
#include <cmath>
using namespace std ;

int main()
{
  double amount;
  double principal = 1000.0;
  double rate = .05; // interest rate

  cout << "Year" << setw( 21 ) << "Amount on deposit" << endl;

  cout << fixed << setprecision( 2 );

 // calculate amount on deposit for each of ten years
 for ( int year = 1; year <= 10; year++ )
 {
   amount = principal * pow( 1.0 + rate, year );
   cout << setw( 4 ) << year << setw( 21 ) << amount << endl;
 } // end for
 return 0; // indicate successful termination
} // end main
```

```
  4           21
Year    Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89
```

15



5.4 Examples Using the `for` Statement (Cont.)

- Formatting numeric output
  - Stream manipulator `setw`
    - Sets field width
      - Right justified by default默认为右对齐输出
        - » Stream manipulator `left` to left-justify
        - » Stream manipulator `right` to right-justify
    - Applies only to the next output value
  - Stream manipulators `fixed` and `setprecision`
    - Sticky settings 粘性设置
      - Remain in effect until they are changed

16

Performance Tip 5.1

•Avoid placing expressions whose values do not change inside. （But…）

•避免循环体内部放置那些不会发生改变的表达式。

•例如：pow(1.0+rate, year)

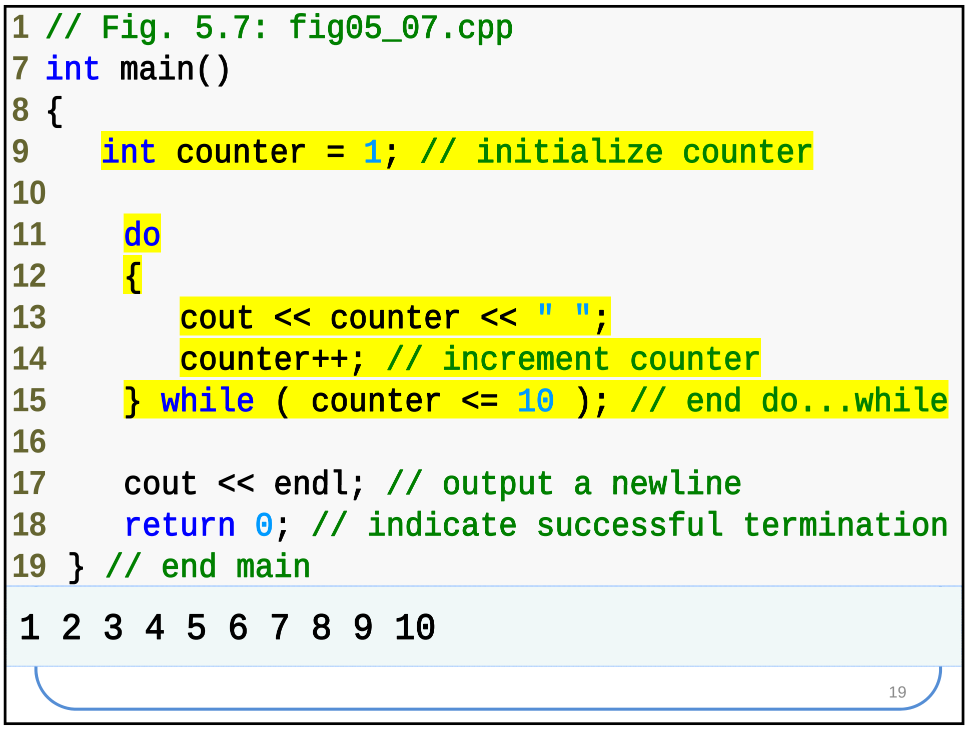
17

5.5 do…while Repetition Statement

- do…while statement
  - Similar to while statement
  - Tests loop-continuation *after* performing body of loop
    - Loop body always executes at least once
  - do

```
do
{
    ……
}while(循环控制条件);
```

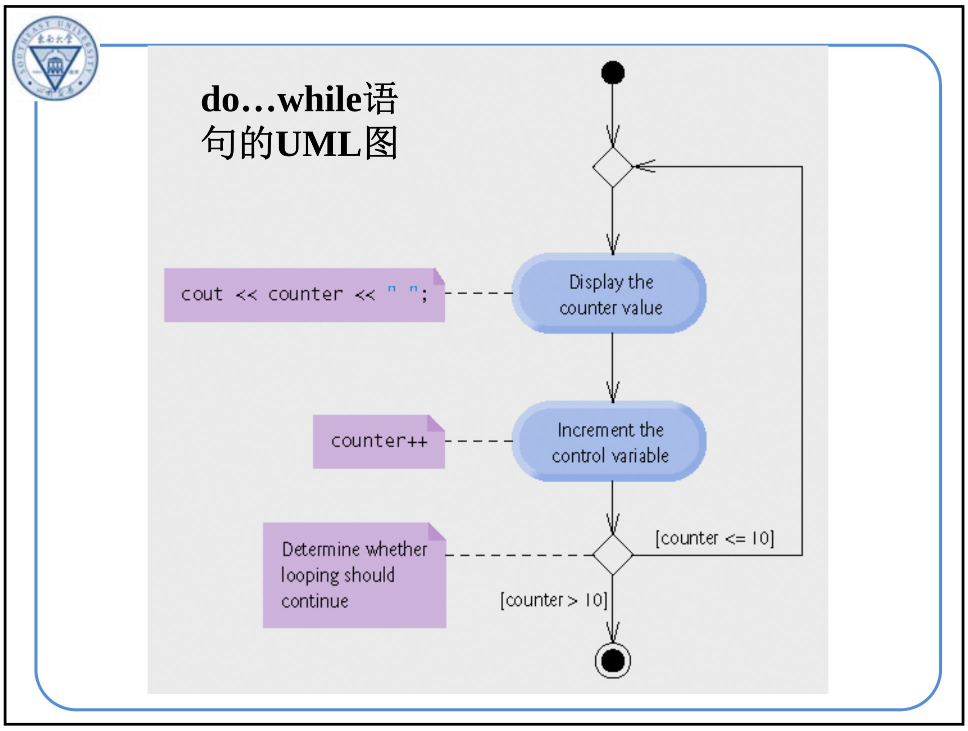
18



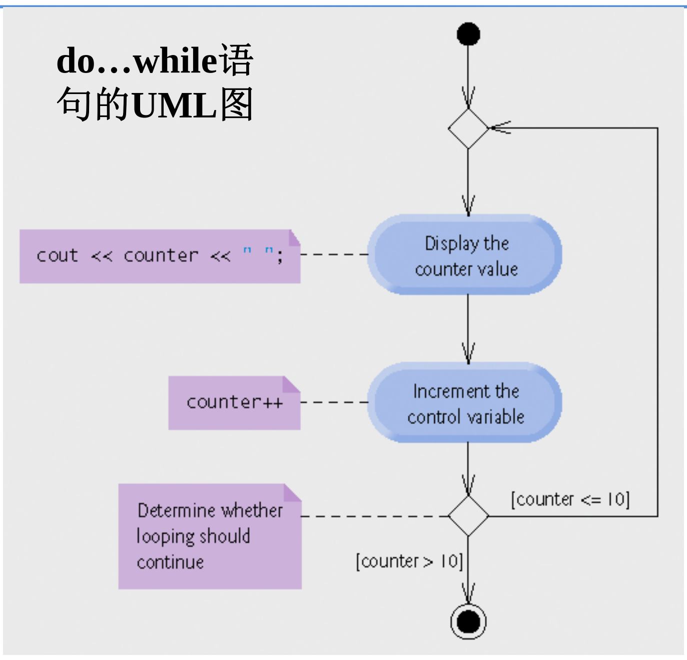
```cpp
// Fig. 5.7: fig05_07.cpp
int main()
{
   int counter = 1; // initialize counter

    do
    {
       cout << counter << " ";
       counter++; // increment counter
    } while ( counter <= 10 ); // end do...while

    cout << endl; // output a newline
    return 0; // indicate successful termination
} // end main
```

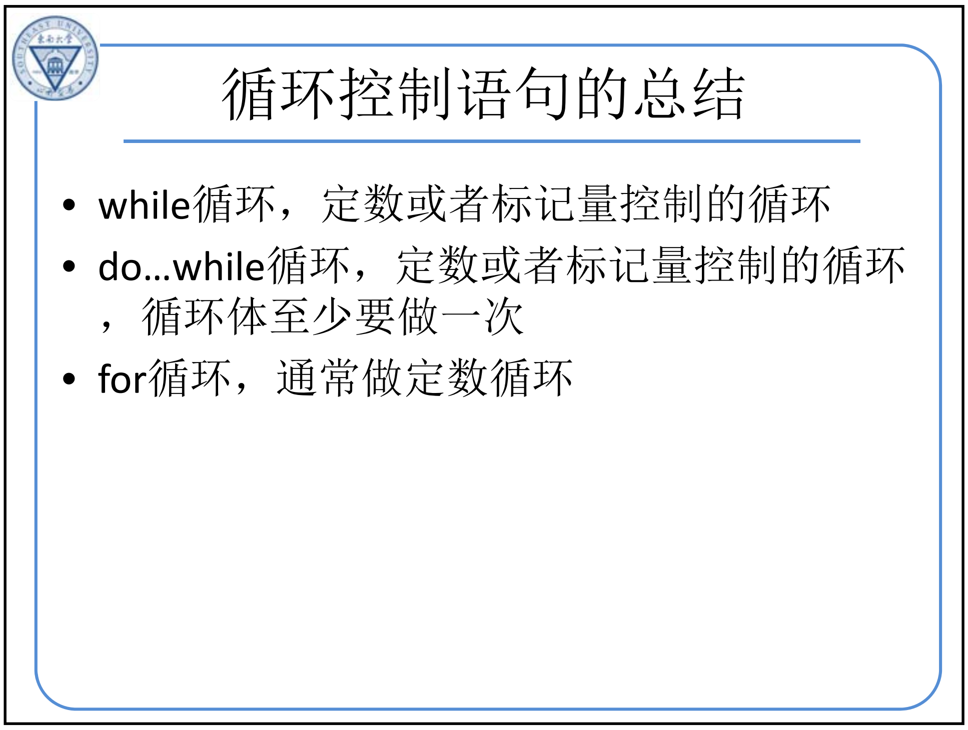
```
1 2 3 4 5 6 7 8 9 10
```

19



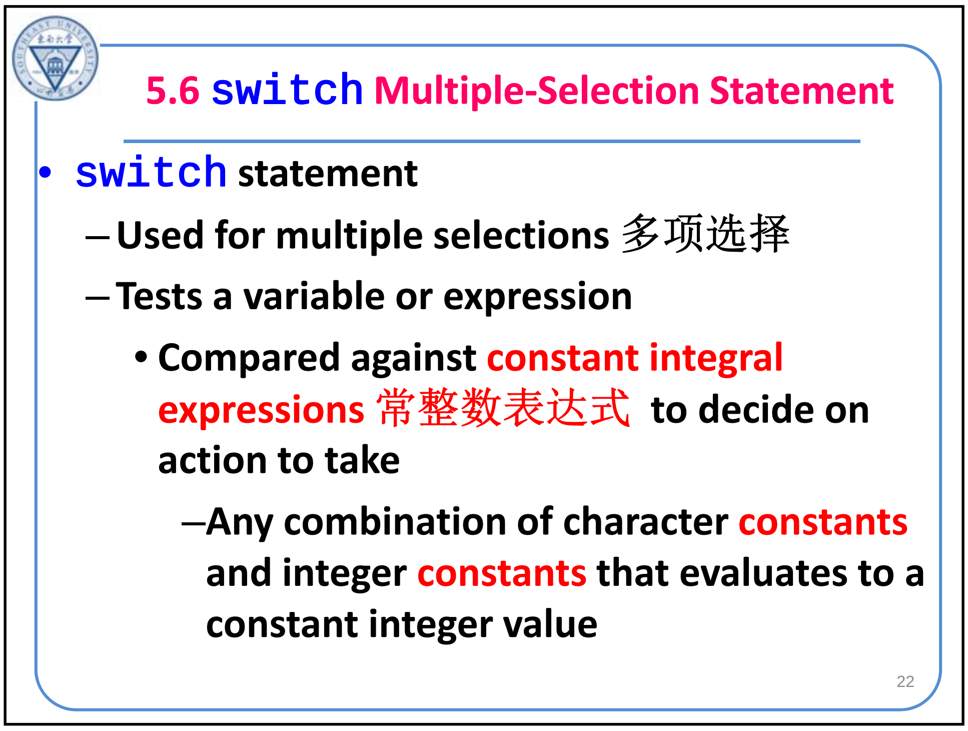
do…while语句的UML图


cout << counter << " ";
Display the counter value
counter++
Increment the control variable
Determine whether looping should continue
[counter <= 10]
[counter > 10]



循环控制语句的总结

- while循环，定数或者标记量控制的循环
- do…while循环，定数或者标记量控制的循环，循环体至少要做一次
- for循环，通常做定数循环



5.6 switch Multiple-Selection Statement

- switch statement
  - Used for multiple selections 多项选择
  - Tests a variable or expression
    - Compared against constant integral expressions 常整数表达式 to decide on action to take
      - Any combination of character constants and integer constants that evaluates to a constant integer value

22

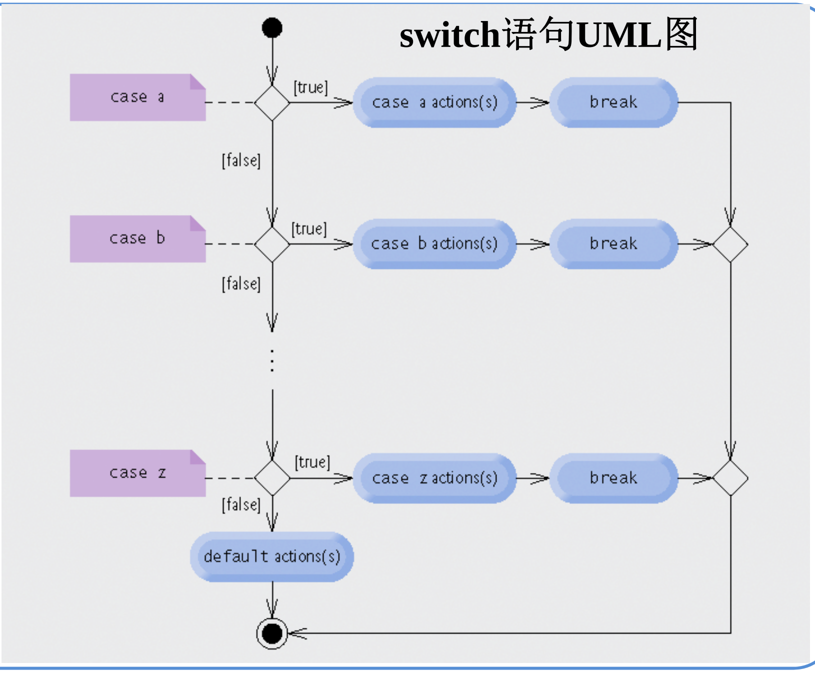
5.6 switch Multiple-Selection Statement

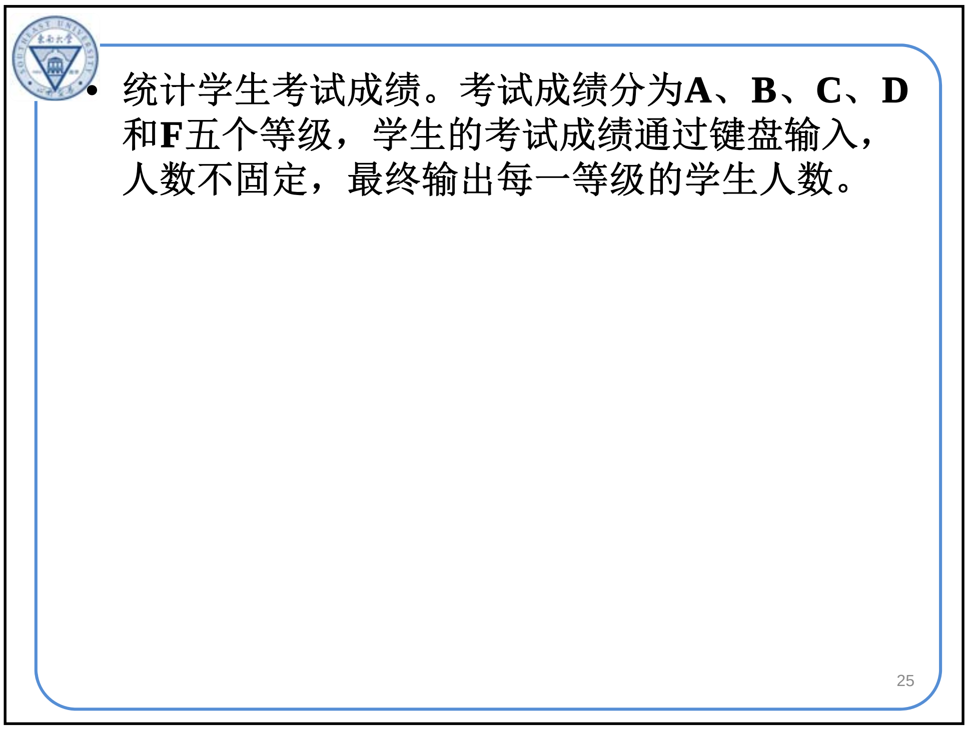
- switch语句的格式

```
switch (整型表达式){
  case 常整型表达式1: <语句序列1> break;
  case 常整型表达式2: <语句序列2> break;
  ……
  case 常整型表达式n: <语句序列n> break;
  default: <语句序列n+1> （可选项）
}
```

23

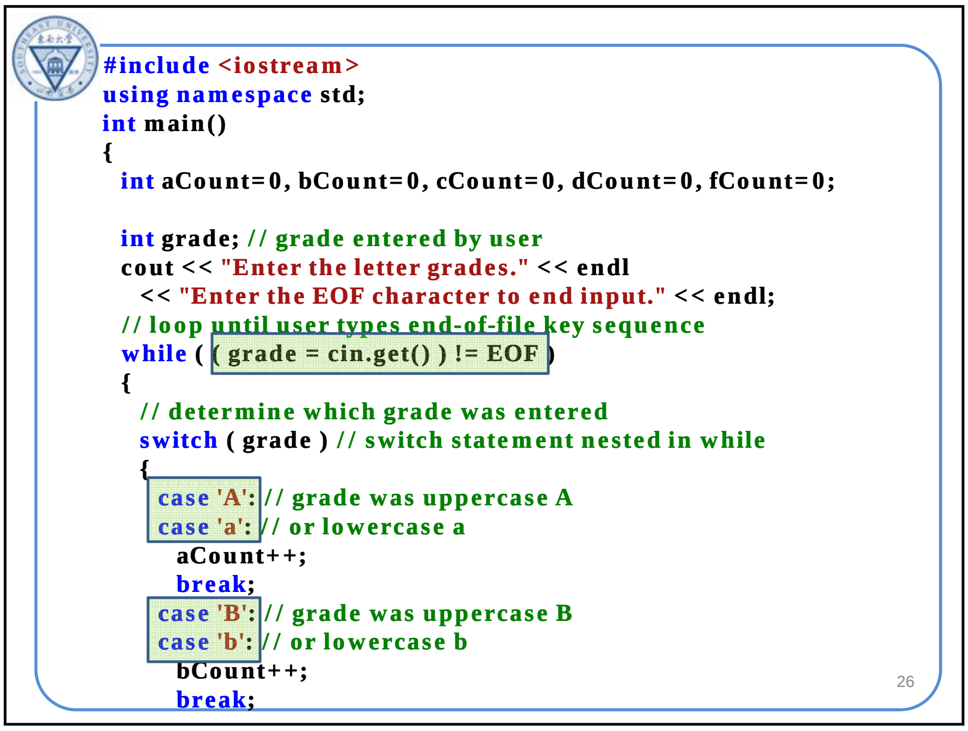
switch语句UML图





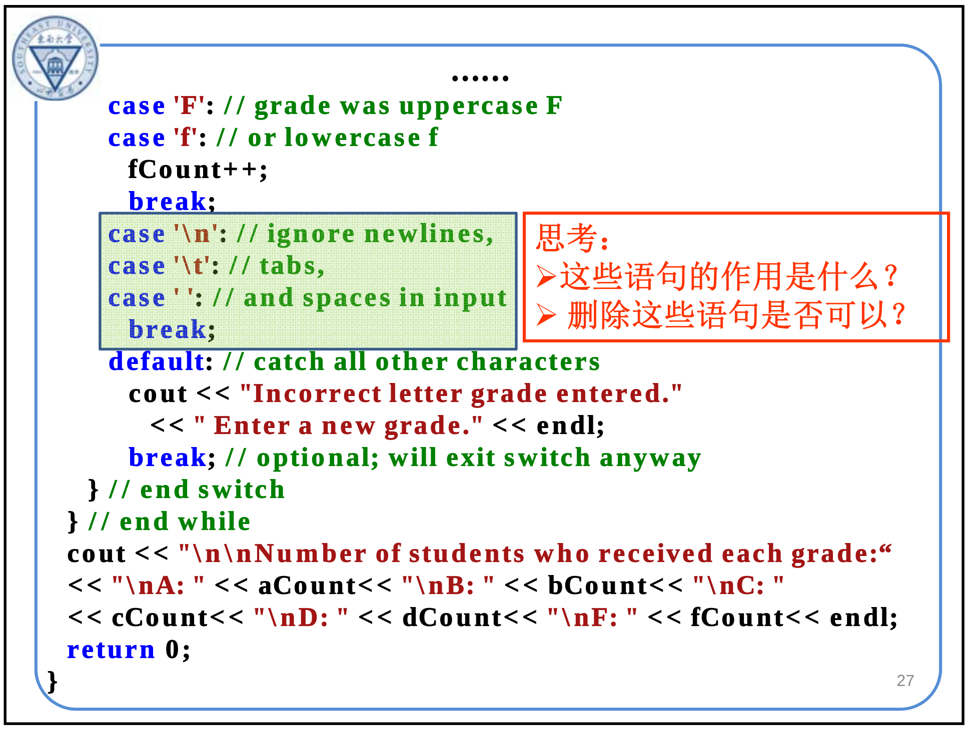
- 统计学生考试成绩。考试成绩分为**A**、**B**、**C**、**D**和**F**五个等级，学生的考试成绩通过键盘输入，人数不固定，最终输出每一等级的学生人数。

25



```
#include <iostream>
using namespace std;
int main()
{
  int aCount=0, bCount=0, cCount=0, dCount=0, fCount=0;

  int grade; // grade entered by user
  cout << "Enter the letter grades." << endl
    << "Enter the EOF character to end input." << endl;
  // loop until user types end-of-file key sequence
  while ( ( grade = cin.get() ) != EOF )
  {
    // determine which grade was entered
    switch ( grade ) // switch statement nested in while
    {
      case 'A': // grade was uppercase A
      case 'a': // or lowercase a
        aCount++;
        break;
      case 'B': // grade was uppercase B
      case 'b': // or lowercase b
        bCount++;
        break;
```

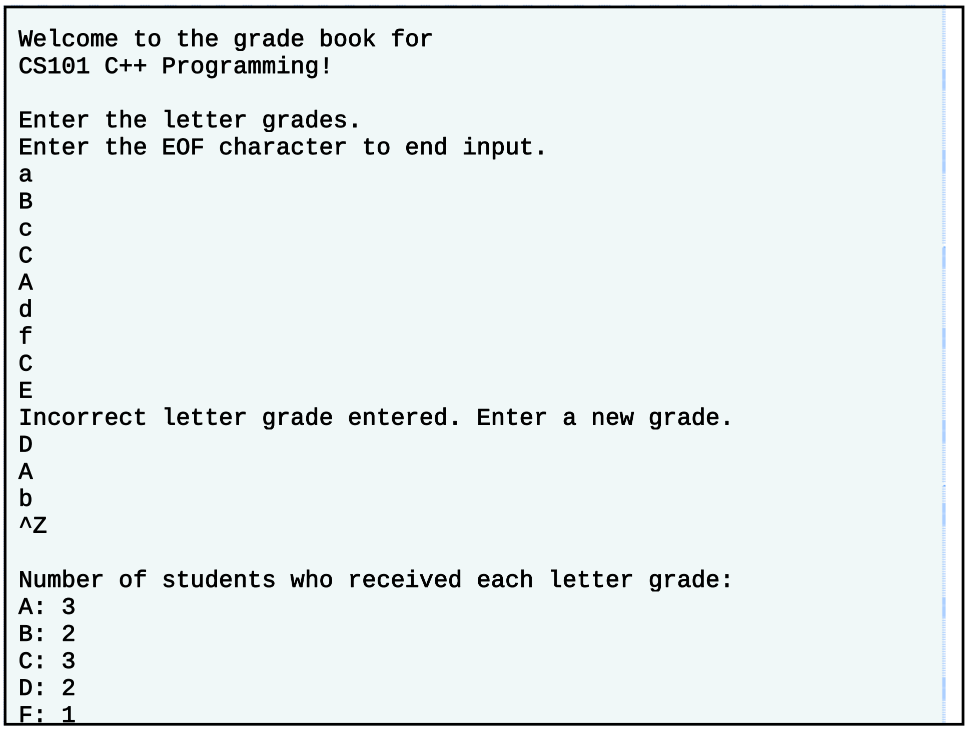
26



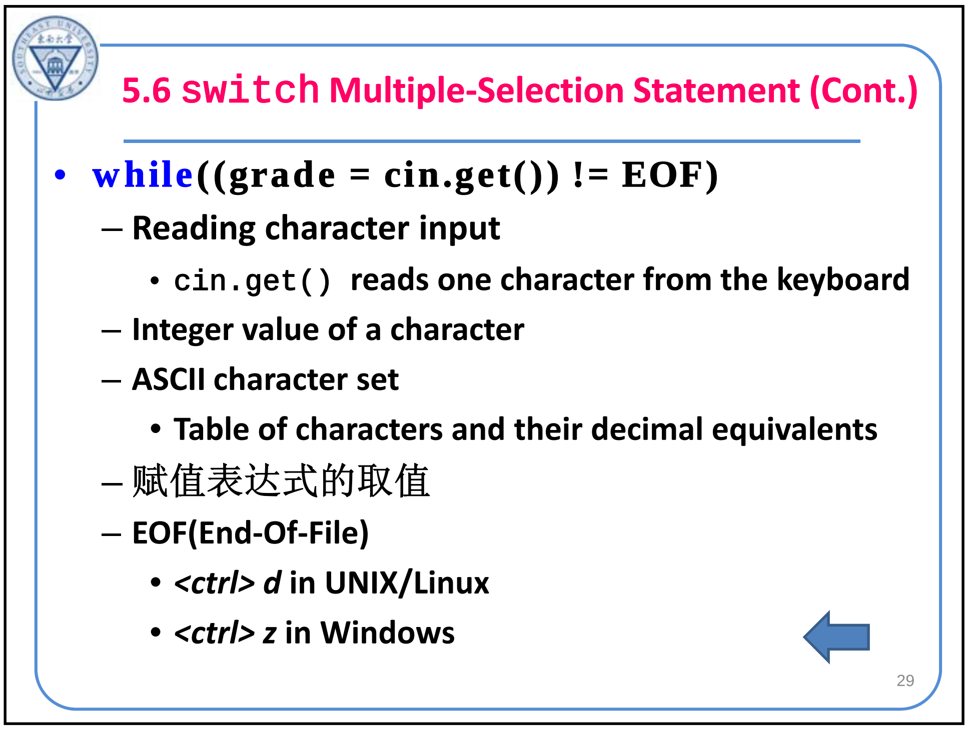
```
……
    case 'F': // grade was uppercase F
    case 'f': // or lowercase f
      fCount++;
      break;
    case '\n': // ignore newlines,
    case '\t': // tabs,
    case ' ': // and spaces in input
      break;
    default: // catch all other characters
      cout << "Incorrect letter grade entered."
        << " Enter a new grade." << endl;
      break; // optional; will exit switch anyway
  } // end switch
} // end while
cout << "\n\nNumber of students who received each grade:“
<< "\nA: " << aCount<< "\nB: " << bCount<< "\nC: "
<< cCount<< "\nD: " << dCount<< "\nF: " << fCount<< endl;
return 0;
}
```

思考：
➢这些语句的作用是什么？
➢ 删除这些语句是否可以？

27

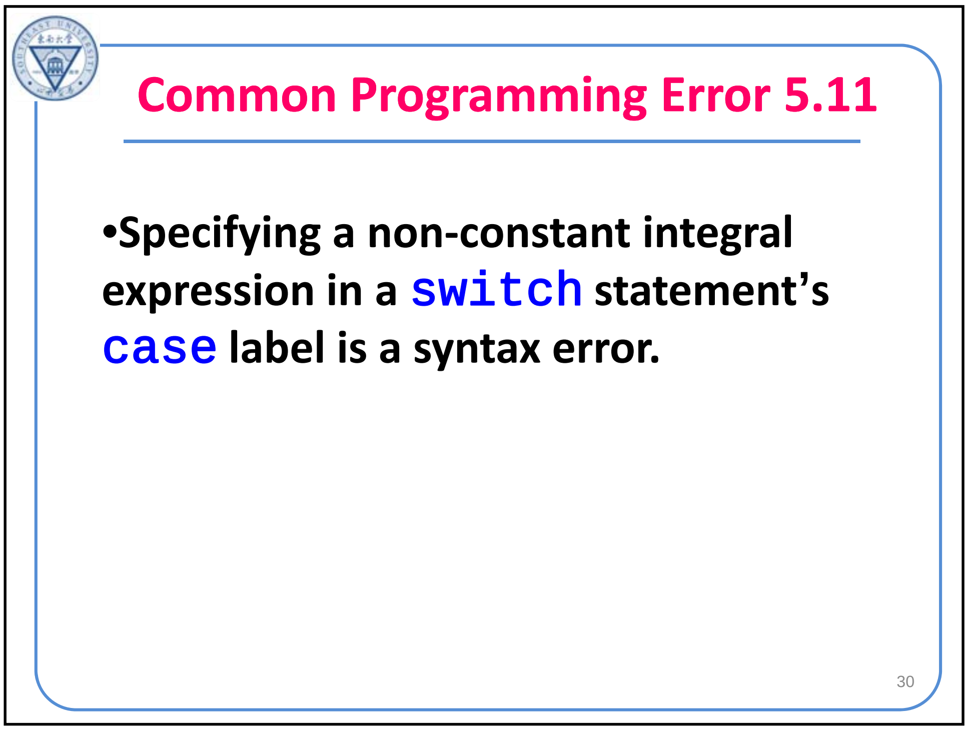


```
Welcome to the grade book for
CS101 C++ Programming!

Enter the letter grades.
Enter the EOF character to end input.
a
B
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z

Number of students who received each letter grade:
A: 3
B: 2
C: 3
D: 2
F: 1
```



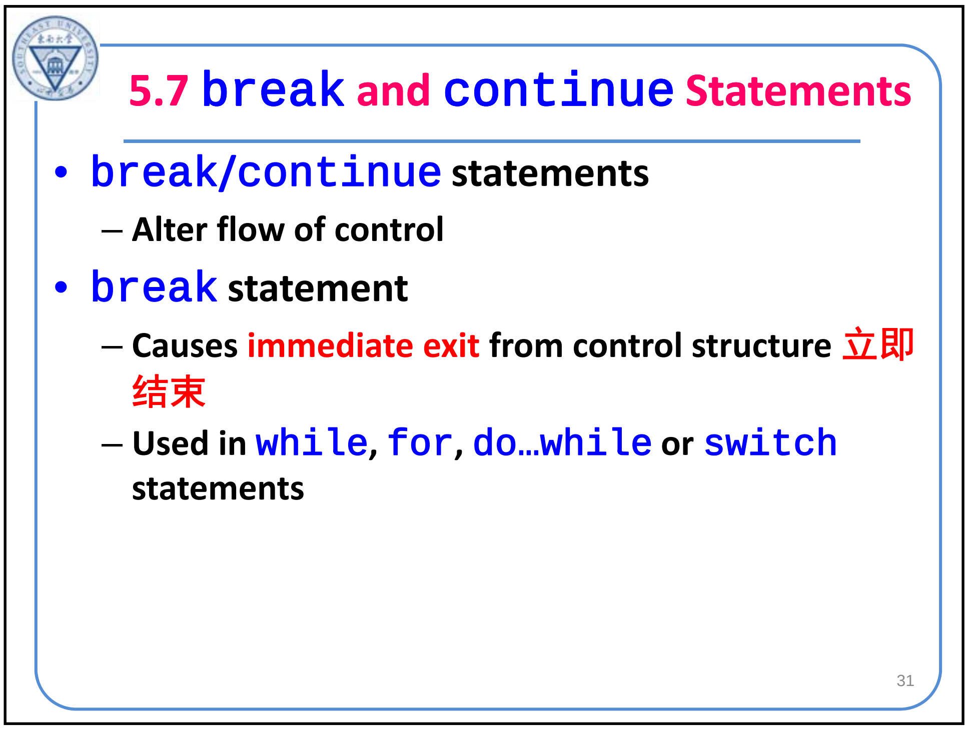
5.6 switch Multiple-Selection Statement (Cont.)

- `while((grade = cin.get()) != EOF)`
  - Reading character input
    - `cin.get()` reads one character from the keyboard
  - Integer value of a character
  - ASCII character set
    - Table of characters and their decimal equivalents
  - 赋值表达式的取值
  - EOF(End-Of-File)
    - *<ctrl> d* in UNIX/Linux
    - *<ctrl> z* in Windows



Common Programming Error 5.11

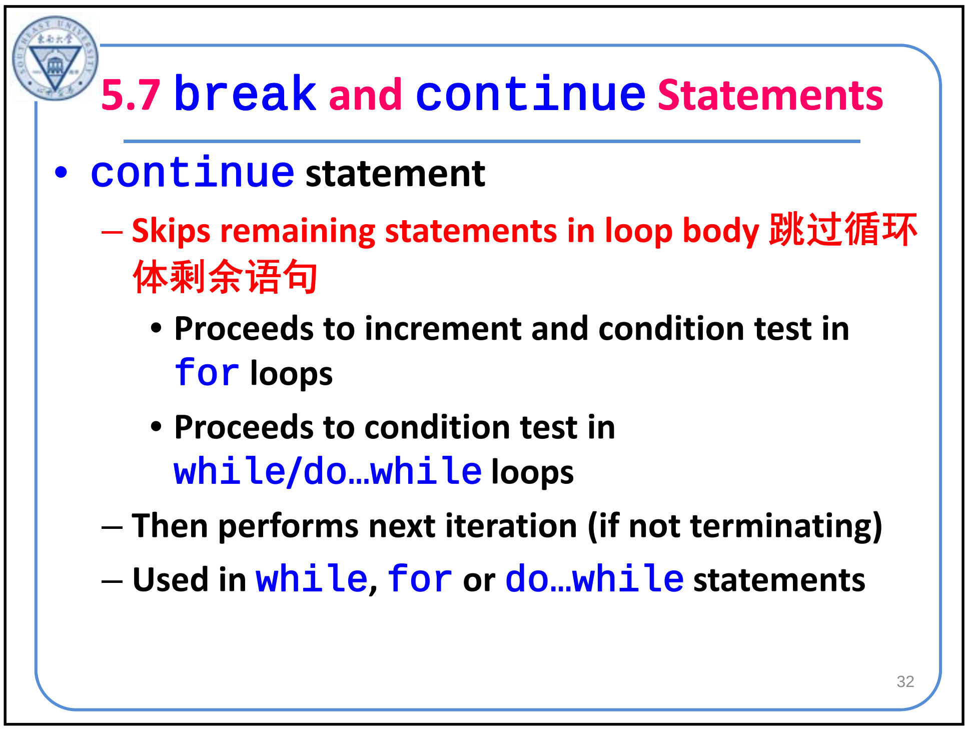
•Specifying a non-constant integral expression in a `switch` statement’s `case` label is a syntax error.



5.7 break and continue Statements

- break/continue statements
  - Alter flow of control
- break statement
  - Causes immediate exit from control structure 立即结束
  - Used in while, for, do…while or switch statements

31



5.7 break and continue Statements

- continue statement
  - Skips remaining statements in loop body 跳过循环体剩余语句
    - Proceeds to increment and condition test in for loops
    - Proceeds to condition test in while/do…while loops
  - Then performs next iteration (if not terminating)
  - Used in while, for or do…while statements

32

```
int main()
{
  int count;

  for ( count = 1; count <= 10; count++ ) // loop 10 times
  {
    if ( count == 5 )
      break; // break loop only if x is 5

    cout << count << " ";
  } // end for

  cout << "\nBroke out of loop at count = " << count << endl;
        return 0; // indicate successful termination
}
```

Output:

```
1 2 3 4
Broke out of loop at count = 5
```

```
int main()
{
  for ( int count = 1; count <= 10; count++ )
  {
    if ( count == 5 ) // if count is 5,
      continue;     // skip remaining code in loop

    cout << count << " ";
  } // end for

  cout << "\nUsed continue to skip printing 5" << endl;
        return 0; // indicate successful termination
} // end main
```

Output:

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

5.8 Logical Operators 逻辑运算符

- Logical operators
  - Allows for more complex conditions
    - Combines simple conditions into complex conditions
- C++ logical operators

  && (logical AND) 逻辑与

  || (logical OR) 逻辑或

  ! (logical NOT) 逻辑非

5.8 Logical Operators (Cont.)

- Logical AND (&&) Operator 逻辑与运算符
  - Consider the following `if` statement

```
if ( gender == 1 && age >= 65 )
   seniorFemales++;
```

  - Combined condition is `true`
    - If and only if both simple conditions are `true`
  - Combined condition is `false`
    - If either or both of the simple conditions are `false`

Common Programming Error 5.13

•Although 3 < x < 7 is a mathematically correct condition, it does not evaluate as you might expect in C++. Use (3 < x && x < 7) to get the proper evaluation in C++.

expression1	expression2	expression1 && expression2
false	false	false
false	true	false
true	false	false
true	true	true

Fig. 5.15 | && (logical AND) operator truth table.

5.8 Logical Operators (Cont.)

- **Logical OR (||) Operator 逻辑或运算符**
  - Consider the following `if` statement

```
if ( ( semesterAverage >= 90 ) ||
   ( finalExam >= 90 )
      cout << "Student grade is A" <<
    endl;
```

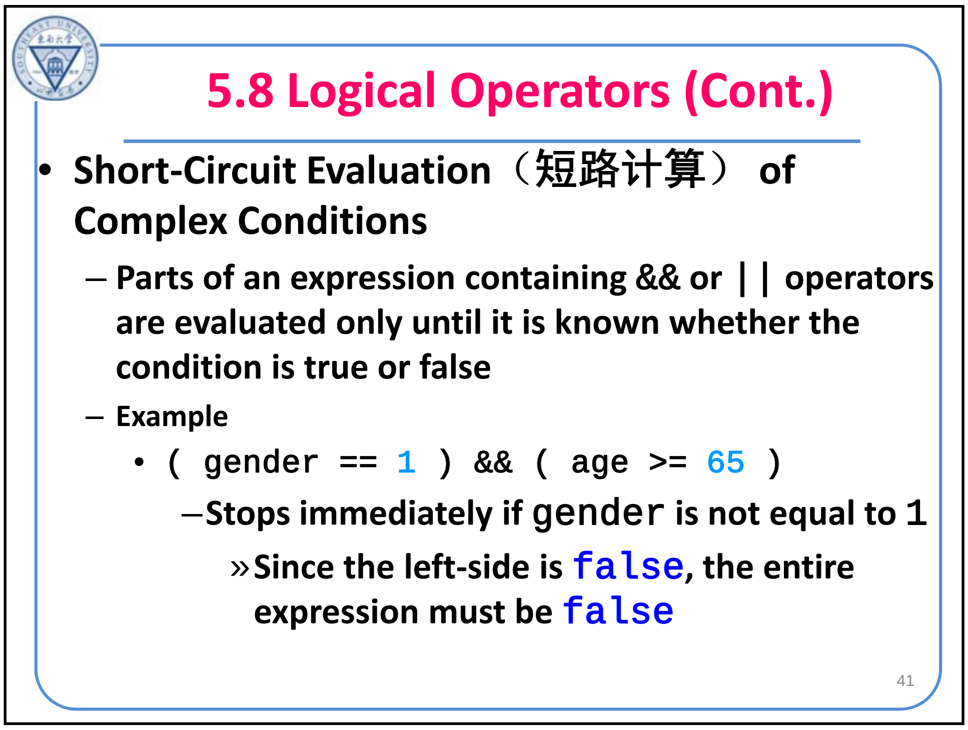
  - Combined condition is `true`
    - If either or both of the simple conditions are `true`
  - Combined condition is `false`
    - If both of the simple conditions are `false`

39

expression1	expression2	expression1 || expression2
`false`	`false`	`false`
`false`	`true`	`true`
`true`	`false`	`true`
`true`	`true`	`true`

Fig. 5.16 | || (logical OR) operator truth table.

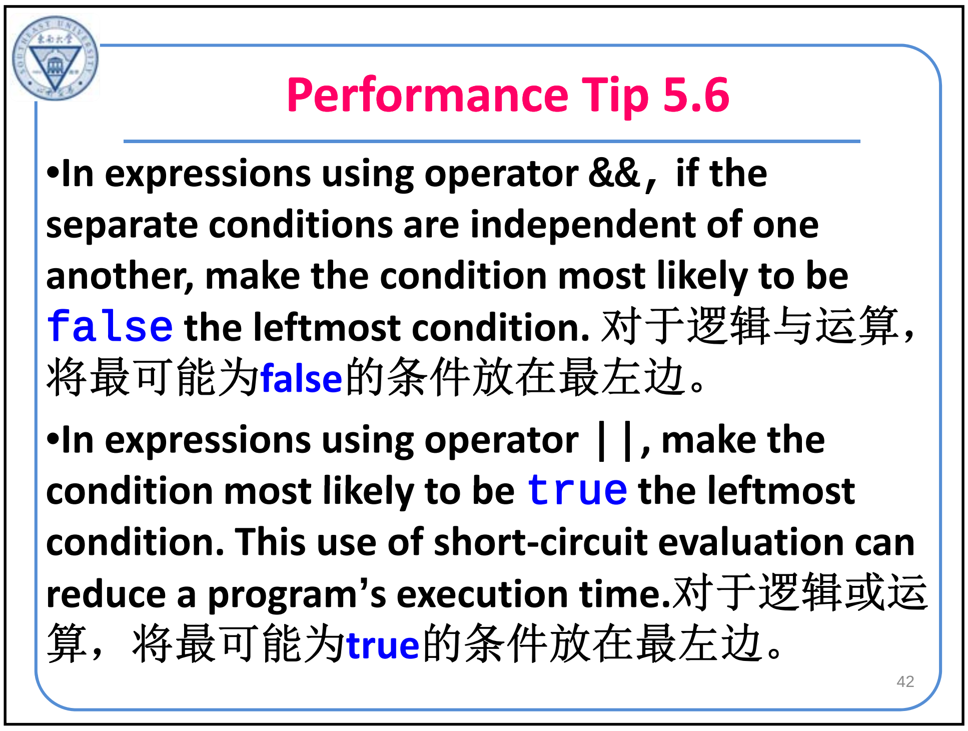
40



5.8 Logical Operators (Cont.)

- **Short-Circuit Evaluation（短路计算） of Complex Conditions**
  - Parts of an expression containing && or || operators are evaluated only until it is known whether the condition is true or false
  - Example
    - `( gender == 1 ) && ( age >= 65 )`
      - Stops immediately if `gender` is not equal to `1`
        - » Since the left-side is `false`, the entire expression must be `false`

41



Performance Tip 5.6

•In expressions using operator &&， if the separate conditions are independent of one another, make the condition most likely to be `false` the leftmost condition. 对于逻辑与运算，将最可能为false的条件放在最左边。

•In expressions using operator ||, make the condition most likely to be `true` the leftmost condition. This use of short-circuit evaluation can reduce a program’s execution time.对于逻辑或运算，将最可能为true的条件放在最左边。

42

5.8 Logical Operators (Cont.)

- **Logical Negation (!) Operator 逻辑非运算符**
  - **Unary operator 一元运算符**
  - **Returns `true` when its operand is `false`, and vice versa**
  -
```
if ( !( grade == sentinelValue ) )
    cout << “The next grade is ” << grade
<< endl;
```

||

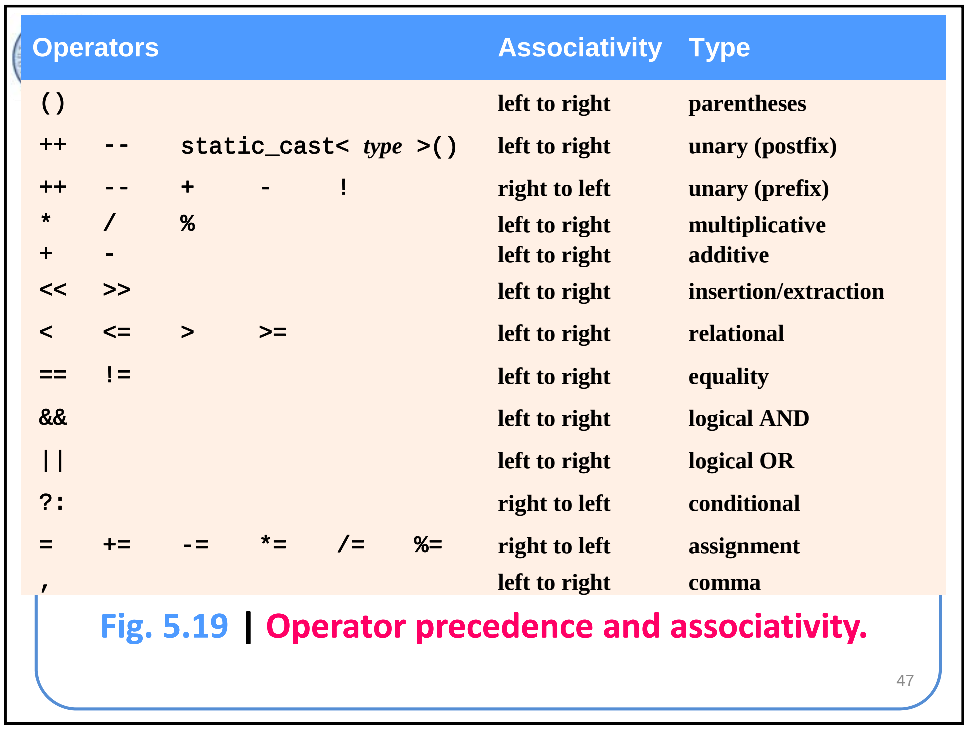
  -
```
if ( grade != sentinelValue )
    cout << "The next grade is " << grade
<< endl;
```

Expression	! expression
`false`	`true`
`true`	`false`

Fig. 5.17 | ! (logical negation) operator truth table.

```cpp
// Fig. 5.18: fig05_18.cpp
// Logical operators.
#include <iostream>
using std::cout;
using std::endl;
using std::boolalpha; // causes bool values to print as "true" or "false"

int main()
{
   // create truth table for && (logical AND) operator
   cout << boolalpha << "Logical AND (&&)"
      << "\nfalse && false: " << ( false && false )
      << "\nfalse && true: " << ( false && true )
      << "\ntrue && false: " << ( true && false )
      << "\ntrue && true: " << ( true && true ) << "\n\n";
......
} // end main
```

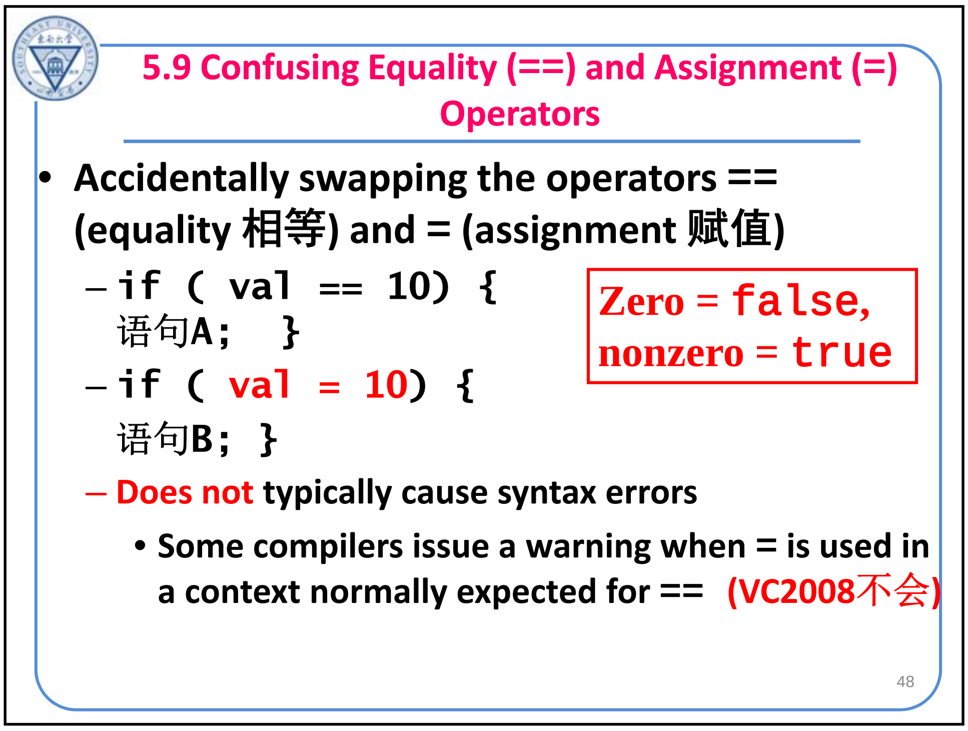
```
Logical AND (&&)
false && false: false
false && true: false
true && false: false
true && true: true

Logical OR (||)
false || false: false
false || true: true
true || false: true
true || true: true

Logical NOT (!)
!false: true
!true: false
```



Operators	Associativity	Type
`()`	left to right	parentheses
`++ -- static_cast<` *type* `>()`	left to right	unary (postfix)
`++ -- + - !`	right to left	unary (prefix)
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`<< >>`	left to right	insertion/extraction
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment
`,`	left to right	comma

Fig. 5.19 | Operator precedence and associativity.

47



5.9 Confusing Equality (==) and Assignment (=) Operators

- Accidentally swapping the operators == (equality 相等) and = (assignment 赋值)
  - `if ( val == 10) {` 语句A; `}`
  - `if ( val = 10) {` 语句B; `}`
  - Does not typically cause syntax errors
    - Some compilers issue a warning when = is used in a context normally expected for == (VC2008不会)

Zero = `false`,
nonzero = `true`

48

5.9 Confusing Equality (==) and Assignment (=) Operators (Cont.)

- ***Lvalues*** **左值**
  - Expressions that can appear on left side of assignment
  - Can be changed (i.e., variables)

    `x = 4;`
- ***Rvalues*** **右值**
  - Only appear on right side of equation
  - Constants, such as numbers (i.e. cannot write `4 = x;`)
- ***Lvalues*** **can be used as** ***rvalues*****, but not vice versa**（左值可以被用于右值，反之则不行）

49

Error-Prevention Tip 5.3

- `if (x == 7) …`
- 用`if (7 == x) …` 替换上述的条件

•**This will prevent the potential devastation of a runtime logic error.**

50

5.10 Structured Programming Summary

- Structured programming
  - Produces programs that are easier to understand, test, debug and modify
- Rules for structured programming
  - Only use single-entry/single-exit control structures
  - Rules (Fig. 5.21)
    - Rule 2 is the stacking rule
    - Rule 3 is the nesting rule

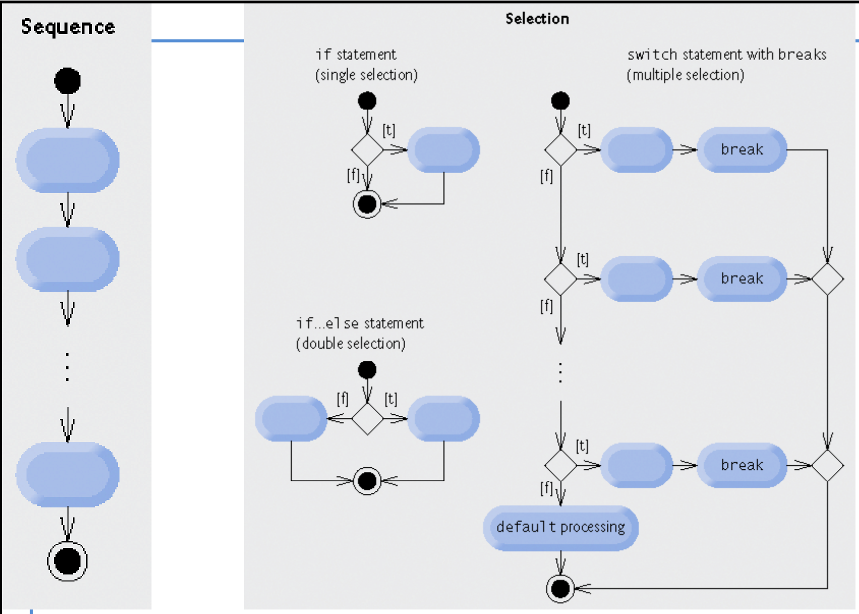
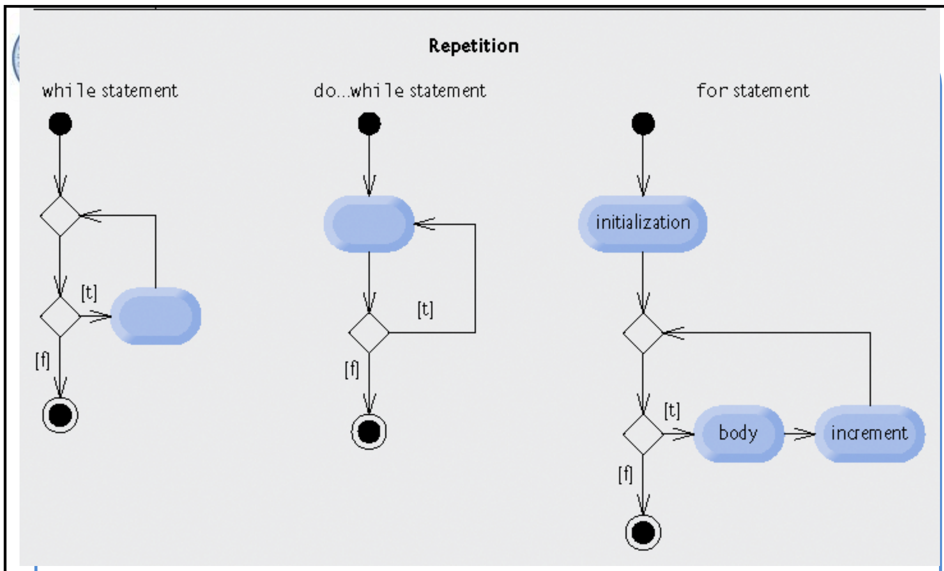


Fig. 5.20 | C++'s single-entry/single-exit sequence, selection and repetition statements.



53

Rules for Forming Structured Programs

1)	Begin with the "simplest activity diagram" (Fig. 5.22).
2)	Any action state can be replaced by two action states in sequence.
3)	Any action state can be replaced by any control statement (sequence, `if`, `if...else`, `switch`, `while`, `do...while` or `for`).
4)	Rules 2 and 3 can be applied as often as you like and in any order.

Fig. 5.21 | Rules for forming structured programs.

54

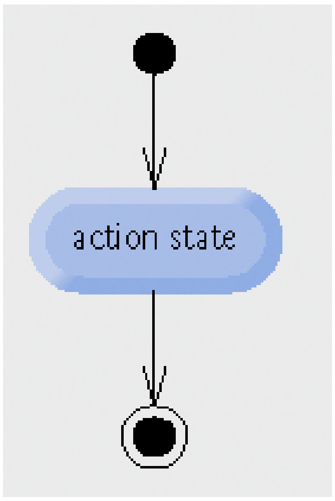


Fig. 5.22 | Simplest activity diagram.

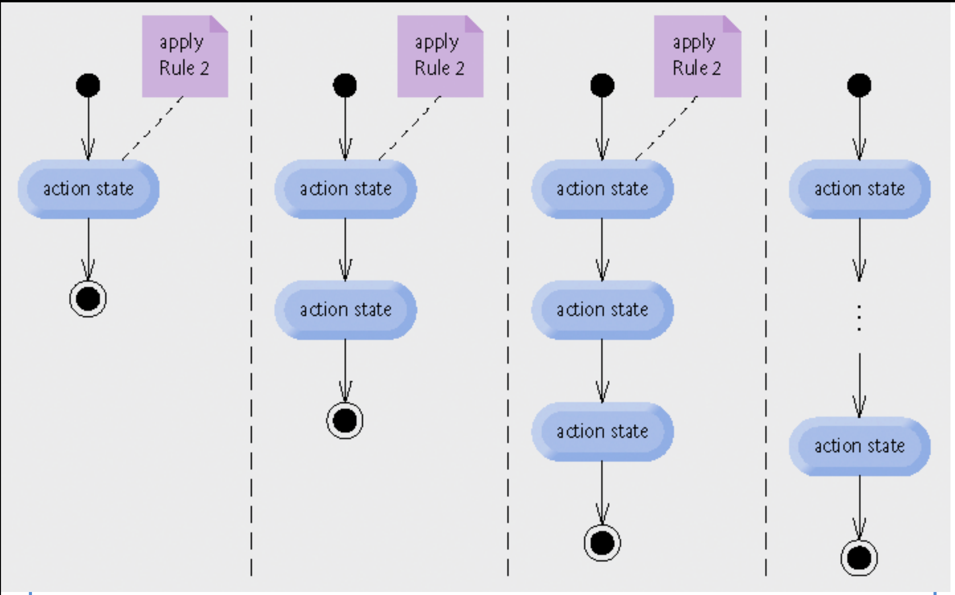


Fig. 5.23 | Repeatedly applying Rule 2 of Fig. 5.21 to the simplest activity diagram.

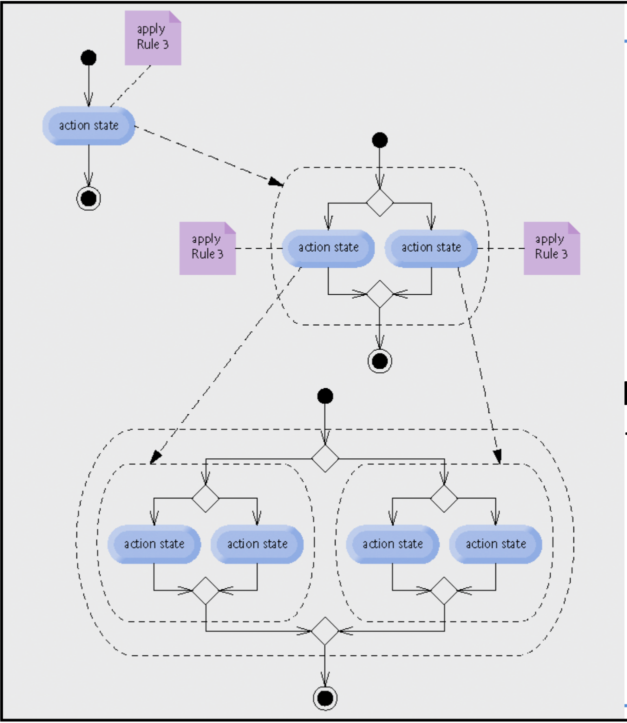


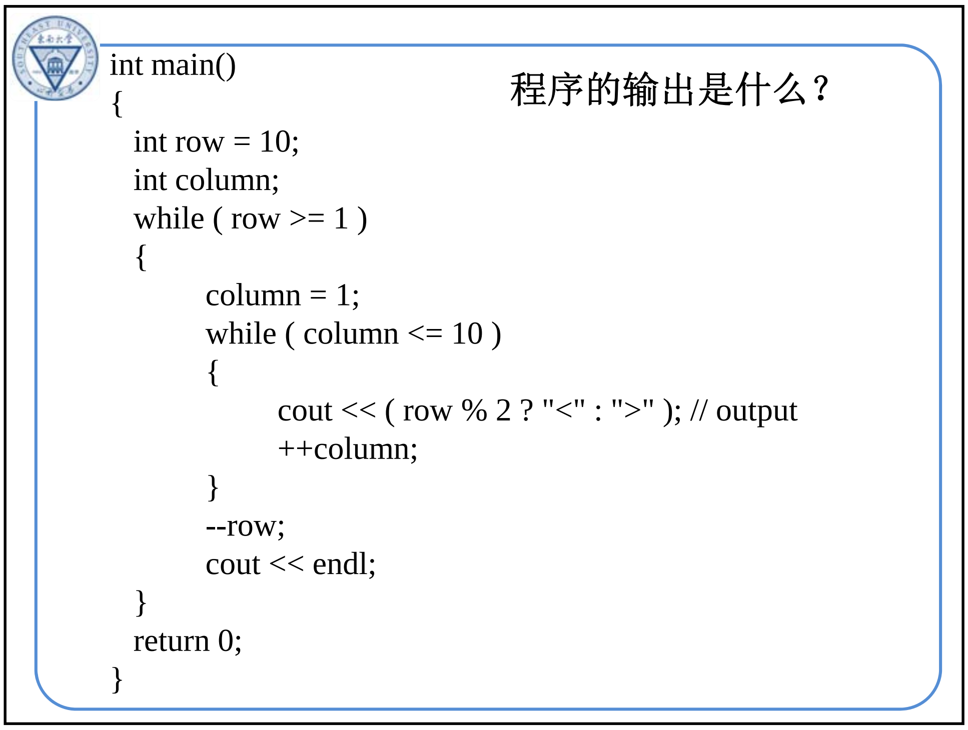
Fig. 5.24 | Applying Rule 3 of Fig. 5.21 to the simplest activity diagram several times.

57

如果删除了switch中的对'\n'等转义字符判断的语句，将得到下列输出结果：

```
Enter the letter grades.
Enter the EOF character to end input.
a
Incorrect letter grade entered. Enter a new grade.
b
Incorrect letter grade entered. Enter a new grade.
C
Incorrect letter grade entered. Enter a new grade.
^Z

Number of students who received each letter grade:
A: 1
B: 1
C: 1
D: 0
F: 0
请按任意键继续. . .
```

58



程序的输出是什么？

```
int main()
{
  int row = 10;
  int column;
  while ( row >= 1 )
  {
        column = 1;
        while ( column <= 10 )
        {
              cout << ( row % 2 ? "<" : ">" ); // output
              ++column;
        }
        --row;
        cout << endl;
  }
  return 0;
}
```