程序设计基础及语言 —— C++ 大学教程

第 5 章 控制语句（第 II 部分）和逻辑运算符

邱 远

东南大学 网络空间安全学院
yuanqiu@seu.edu.cn

2026 年 11 月

目录

for 语句

do ... while 语句

switch 语句与枚举类型

逻辑运算符

定数循环

定数循环包括如下四个要素

1. 初始化：命名控制变量并赋初值
2. 循环（继续）条件：判断何时继续循环
3. 循环体
4. 更新：更新控制变量（通常为自增）

例如

```
#include <iostream>

int main() {
    unsigned int counter = 0; // declare and init control variable
    while (counter < 10) { // loop condition
        std::cout << counter << ' ';
        ++counter; // increment control variable
    }
    std::cout << std::endl;
}
```

for 循环

https://en.cppreference.com/cpp/language/for

for 语句常用于实现定数循环

for 循环语句

```
for (初始化; 循环条件; 更新) {
    循环体;
}
```

- 与while实现的定数循环 “几乎” 是等价的

```
1  // definite loop with for
2  for (unsigned int counter{}; counter < 10; ++counter) {
3      std::cout << counter << ' ';
4  }
```

```
1  // definite loop with while
2  unsigned int counter{};
3  while (counter < 10) {
4      std::cout << counter << ' ';
5      ++counter;
6  }
```

for 循环

https://en.cppreference.com/cpp/language/for

for 循环语句

```
for (初始化; 循环条件; 更新) {
    循环体;
}
```

- 执行顺序为：初始化 ⟶ 循环条件 ⟶ 循环体 ⟶ 更新



- 若初始值就不满足循环条件，可以一次也不执行（类似于`while`）
- 与`while`的区别在于初始值的作用域：`for`中初始化的变量在循环结束后就不可见了

```
for (int i = 0; i < 10; ++i) { ... }
std::cout << i; // error: 'i' is undefined
```

  - 若想在`for`外使用，需要将`i`定义在外部，或使用`while`
  - 定义在内部可以减少无关变量，并且可以反复使用同一名字

带初始化的 if（拓展）

https://en.cppreference.com/cpp/language/if

类似于**for**，从 C++17 起，**if**也支持定义仅内部可见的变量

```cpp
#include <iostream>

int main() {
    double weight, height;
    std::cin >> weight >> height; // kg, m

    if (double bmi = weight / (height * height); bmi >= 24.0) {
        std::cout << "Overweight! BMI: " << bmi << '\n';
    } else {
        std::cout << "Normal weight. BMI: " << bmi << '\n';
    }

    // bmi is not visible from this point
}
```

C++17 标准的代码需要g++ -std=c++17编译

逗号运算符

https://en.cppreference.com/cpp/language/operator_other

- a，b的效果是：先算 a，再算 b，然后返回 b
  - 优先级在所有运算符中最低
- 用于在只能填入一个表达式的地方执行额外语句
  - 常用于在for循环中有两个控制变量的情况[1]

打印 2^0 到 2^{10}

```
int power = 1;
for (int i = 0; i <= 10; ++i) {
    std::cout << power << ' ';
    power *= 2;
}
```

也可以写成

```
for (int i = 0, power = 1; i <= 10; ++i, power *= 2) {
    std::cout << power << ' ';
}
```

- 也可用于此前学到的通过标记值结束读入

  `while (std::cin >> grade, grade != -1) { total += grade; }`

[1] 当在初始化部分同时声明多个变量时，它们必须具有相同类型，赋值不受影响

省略表达式

for 循环语句

```
for (初始化; 循环条件; 更新) {
    循环体;
}
```

- for循环中的每个部分都可以为空，分别出现在以下场景
- 若变量在外部初始化，则初始化部分可以为空

```
int c{};
for (; c < 10; ++c) { ... }
std::cout << c;
```

- 循环体和更新本质是一先一后执行，可以完全将一者放入另一者中，此时类似while

```
int c{};
for (; c < 10;) { std::cout << c; ++c; } // equivalent to "while (c < 10)"
for (; c < 20; std::cout << c, ++c); // note usage of , instead of ;
```

- 若想要无限循环，可以令循环条件为空，等价于while (true)，但少打 4 个字

```
for (;;) { ... } // equivalent to "while (true)"
```

使用 for 的例子

- 依次输出 1 到 100 的所有整数

```
for (size_t i = 1; i <= 100; ++i)
    std::cout << i << ',';
```

- 依次输出 100 到 0 的所有整数

```
for (int i = 100; i >= 0; --i)
    std::cout << i << ',';
```

注意：当 i 为-1 时才会退出循环，因此 i 必须可以取负数

- 从小到大输出 100 以内 7 的倍数

```
for (unsigned i = 7; i < 100; i += 7)
    std::cout << i << ',';
```

对于这种每次增减固定量的情况，我们称 7 为**步长**

- 从小到大输出 100 以内 $\pi = 3.14$ 的正整数指数幂

```
for (double p = 1.; p < 100; p *= 3.14)
    std::cout << p << ',';
```

步长不为 1 时，慎用`i != 100`作为循环条件，容易导致死循环！

程序一：计算 2-20 所有偶数的和

```cpp
#include <iostream>

int main() {
    unsigned int total{};
    for (unsigned int number = 2; number <= 20; number += 2) {
        total += number;
    }

    std::cout << "Sum is " << total << std::endl;
}
```

程序二：计算复利

输入：本金 p、年利率 r、存款总年份 n
输出：每年年终的本息和，第 i 年计算方式为 $a_i = p(1+r)^i$

```
1  #include <iostream>
2  #include <iomanip>
3
4  int main() {
5      std::cout << "Enter principle, rate, term";
6      double principle, rate;
7      unsigned term;
8      std::cin >> principle >> rate >> term;
9
10     std::cout << "Year" << std::setw(21) << "Amount on deposit" << std::endl;
11
12     std::cout << std::fixed << std::setprecision(2);
13
14     double amount = principle;
15     for (unsigned year = 1; year <= term; ++ year) {
16         amount *= 1 + rate;
17         std::cout << std::setw(4) << year << std::setw(21) << amount << std::endl;
18     }
19 }
```

指数函数与设定宽度

https://en.cppreference.com/cpp/numeric/math/pow

- 除了可以用累乘，也可以用<cmath>库中的std::pow函数

```
#include <cmath>
    ...
    amount = principle * std::pow(1. + rate, year);
```

std::pow

```
double      pow ( double base, int exp );
double      pow ( double base, double exp );
```

- std::setw(int)用于设置**下一次**输出的宽度
  - 右对齐，超过宽度就按实际宽度输出
  - 由于是有参数的设置，需要#include <iomanip>
  - 想要更改左右对齐，可使用std::left, std::right, std::internal[2]
  - std::setw(int)是非粘性设定，只作用于下一次输出
  - 此前学过的std::fixed, std::setprecision(int)都是粘性(sticky)的

循环中不会发生改变的操作，尽可能放在循环外以优化效率

[2] https://en.cppreference.com/cpp/io/manip/left

练习：C++ 的 for 循环

小练习

使用 for 循环打印九九乘法表，要求对齐

练习：C++ 的 for 循环

小练习

使用 for 循环打印九九乘法表，要求对齐

```cpp
// example.cpp
#include <iomanip>
#include <iostream>

int main() {
    for (int i = 1; i <= 9; ++i) {
        for (int j = 1; j <= i; ++ j) {
            std::cout << j << "x" << i << "=" << std::setw(2) << i*j << ' ';
        }
        std::cout << std::endl;
    }
} // end function main
```

目录

for 语句

do ... while 语句

switch 语句与枚举类型

逻辑运算符

do ... while 循环

do ... while 循环语句

```
do {
    循环体;
} while (循环条件);
```

- do ... while语句和while语句类似，但会先将循环体执行一次
  - 对比：for循环和while循环都可以一次都不执行
- 有时也把while循环称为“当型循环”，do ... while循环称为“直到型循环”[3]
- 可以通过额外代码转为while循环，有时会更简介

[3] 某些语言为后者使用 until 关键字，C++ 设计者认为没有必要

程序三：通过泰勒展开计算 e^x

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \ldots$$

```
1  #include <cmath>
2  #include <iomanip>
3  #include <iostream>
4
5  int main() {
6      std::cout << "Please enter exponent and order > 0: " << std::endl;
7      double exponent, sum{1}, numerator{1.};
8      unsigned int order, denominator{1}, counter{0};
9      std::cin >> exponent >> order;
10     do {
11         numerator *= exponent;
12         denominator *= ++counter;
13         sum += numerator / denominator;
14     } while (counter < order);
15     std::cout << std::setprecision(20) << "Order-" << order << " taylor estimation: "
16               << sum << "\nActual: " << std::exp(exponent) << std::endl;
17 }
```

目录

for 语句

do ... while 语句

switch 语句与枚举类型

逻辑运算符

switch 多选

https://en.cppreference.com/cpp/language/switch

switch 多选语句

```
switch (表达式) {
case 常量表达式:
    语句;
case 常量表达式:
    语句;
...
default:
    语句;
}
```

- switch会计算括号内表达式，并从匹配的case入口进入大括号
  - 若都不匹配，有default从default进入，否则跳过大括号
- 括号内表达式的值必须是整型，字符型或者枚举型
  - 本质上，执行时需要进行==判断，而浮点无法判断相等
  - bool也是枚举型，但switch (bool)就是if
- 常量表达式必须在编译时能确定值，并且不能有相同值

程序四：统计 A、B、C、D、F 成绩（接口与实现）

头文件 (GradeBook.h)

```
1  /** @file GradeBook.h */
2  #pragma once
3  #include <string> // C++ standard string class
4  /** @class GradeBook */
5  class GradeBook {
6  public:
7      explicit GradeBook(std::string);
8      void setCourseName(std::string);
9      std::string getCourseName() const;
10     void displayMessage() const;
11     void inputGrades();
12     void displayGradeReport() const;
13 private:
14     std::string courseName;
15     unsigned int aCount{0}, bCount{0},
16         cCount{0}, dCount{0}, fCount{0};
17 }; // end class
```

实现源文件 (GradeBook.cpp)

```
1  /** @file GradeBook.cpp */
2  #include "GradeBook.h"
3  #include <iostream>
4
5  void GradeBook::inputGrades() {
6      int grade{};
7      std::cout << "Enter letter grades (EOF to end): " << std::endl;
8      while ((grade = std::cin.get()) != EOF) {
9          switch (grade) {
10         case 'a': case 'A': ++aCount; break;
11         case 'b': case 'B': ++bCount; break;
12         case 'c': case 'C': ++cCount; break;
13         case 'd': case 'D': ++dCount; break;
14         case 'f': case 'F': ++fCount; break;
15         case '\t': case '\n': case ' ': break;
16         default:
17             std::cout << "Incorrect letter! Enter a new one.\n";
18         }
19     }
20 }
21 void GradeBook::displayGradeReport() const {
22     std::cout << "\nNumber of students who received each letter grade:"
23         << "\nA: " << aCount << "\nB: " << bCount
24         << "\nC: " << cCount << "\nD: " << dCount
25         << "\nF: " << fCount << std::endl;
26 }
27 ...
```

程序四：统计 A、B、C、D、F 成绩（测试与结果）

```
1  /** @file main.cpp */
2  #include "GradeBook.h"
3
4  int main() {
5      GradeBook myGradeBook{"CS101 C++ Programming"};
6      myGradeBook.displayMessage();
7      myGradeBook.inputGrades();
8      myGradeBook.displayGradeReport();
9  }
```

程序输出

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

Enter letter grades (EOF to end):
aBcCAdfCE
Incorrect letter! Enter a new one.
D
A
b
^D

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

Line 9-12： break

https://en.cppreference.com/cpp/language/break

```
9         switch (grade) {
10        case 'a': case 'A': ++aCount; break;
11        case 'b': case 'B': ++bCount; break;
12        case 'c': case 'C': ++cCount; break;
13        case 'd': case 'D': ++dCount; break;
14        case 'f': case 'F': ++fCount; break;
15        case '\t': case '\n': case ' ': break;
16        default:
17            std::cout << "Incorrect letter! Enter a new one.\n";
18        }
```

- 若不加`break;`多选语句会从入口一直执行至尾
  - 尽管`case 'a':`与`case 'A':`之间没有代码，从`case 'a':`进来还是能执行到`++aCount;`
  - 因此`switch`语句同一个 case 的代码无需大括号
- `break;`的作用是跳出当前的循环或多选
- 注意对`if`无效，只用于`switch`，`for`，`while`，`do ... while`
  - `if`只要碰到右大括号就继续执行了
- 当有多层循环时，跳出最里侧的一层

break 与 continue

https://en.cppreference.com/cpp/language/continue

- 与break;类似的指令是continue;作用是直接进行下一次循环
  - 对于while与do ... while循环，跳过剩余循环体，判断循环条件
  - 对于for循环，跳过剩余循环体，先进行变量更新，再判断循环条件
- 二者都只作用于最内层[4]循环或多选

```
for (int i=1; i<=9; ++i) {
    for (int j=1; j<=9; ++j) {
        if (i == j) break;
        std::cout << i << j << ' ';
    }
}
```

所有个位数小于十位数的两位数

```
for (int i=1; i<=9; ++i) {
    for (int j=1; j<=9; ++j) {
        if (i == j) continue;
        std::cout << i << j << ' ';
    }
}
```

不是 11 倍数的两位数

- 如果while中的continue;跳过了循环准备，可能导致死循环

```
int i = 0;
while (i <= 10) {
    if (i == 5) continue; // infinite loop
    std::cout << i << ' ';
    ++i;
}
```

[4] 函数内可以通过return语句直接返回以跳过所有剩余语句

Line 8：C++ 字符

```
6       int grade{};
7       std::cout << "Enter letter grades (EOF to end): " << std::endl;
8       while ((grade = std::cin.get()) != EOF) {
```

- `int std::cin.get()`的作用是从输入流读取一个字符
  - 可以是空格、换行等
  - 对比：`std::cin >> s;`读一个空白分隔的词
  - 对比：`std::getline(std::cin, s);`读一整行
- 不同于字符串需要存在`std::string`类型中，单一字符可以存在`char`或`int`类型中，并用单引号表示
  - `char`是否有符号取决于系统，因此`std::cin.get()`返回`int`
- C++ 中，字符与整数可以互相转换，如`'A' == 65`为`true`
  - 字符同样可以进行运算，如大小写转换的最简单方法
    `char capital = little - 'a' + 'A';`
- `EOF`是`<iostream>`中定义的一个特殊字符，其值取决于系统
  - 输入流到文件末尾时返回 EOF
  - 终端中如何输入该值取决于系统和软件：Windows (Ctrl+Z), MacOS (Ctrl+D), Linux (Ctrl+D), Emacs (C-c C-d)

枚举类型

https://en.cppreference.com/cpp/language/enum

枚举类型 (章节 6.8)

```
enum 类型名 { 枚举值, 枚举值 };
enum 类型名: 整型 { 枚举值 = 1, 枚举值 = 2 };
enum class 类型名 { 枚举值 = 1, 枚举值 };
```

- 和class类似，枚举类型也是一种用户自定义类型
- 用于取值有限的变量，提高代码可读性，同时减少非法值

```
enum DayOfWeek { MON, TUE, WED, THU, FRI, SAT, SUN };
DayOfWeek day{THU};
```

- 枚举类型定义和变量定义可以合并[5]

```
enum Color { RED, GREEN, BLUE } c{RED};
```

- 可以认为枚举类型也是给整数起别名
  - 可以用冒号规定实际的类型 (C++11 起)
  - 枚举值不能重名

[5] 其实class也可以，但不常用，这也是为什么二者定义后都有分号

枚举值

https://en.cppreference.com/cpp/language/enum

枚举类型 (章节 6.8)

```
enum 类型名 { 枚举值, 枚举值 };
enum 类型名: 整型 { 枚举值 = 1, 枚举值 = 2 };
enum class 类型名 { 枚举值 = 1, 枚举值 };
```

- 若无定义值，枚举值从 0 开始依次递增
- 若有定义值，枚举值从定义开始递增至下个定义
- 定义值可以重复（容易造成歧义，谨慎使用）

```
enum State { ERROR = 0, PROBLEM = 0, OK = 1 } state{};
if (!state) { std::cerr << "Not OK!" << std::endl; }
```

- 枚举值可以转为int进行输出、比较、计算等，但 C++ 禁止将整数隐式转换为枚举类型，必须使用枚举值或显式转换[6]

```
state = OK; state = static_cast<State>(1);
```

- enum class可以避免污染全局命名，只通过类型名::枚举值使用

[6] 允许默认初始化，此时变量会被初始化至 0（不论有没有对应 0 的枚举值）

switch 用于枚举类型

```
#include <iostream>
int main() {
    enum class Color { RED, GREEN = 20, BLUE } c = Color::BLUE;
    switch (c) {
    case Color::RED: std::cout << "RED\n"; break;
    case Color::GREEN: std::cout << "GREEN\n"; break;
    case Color::BLUE : std::cout << "BLUE\n"; break;
    default: std::cerr << "Unknown color!\n"; break;
    }
}
```

静态反射（拓展，无需掌握）

C++26 拟引入反射，将简化枚举输出 (-std=c++26 -freflection)

```
#include <iostream>
#include <meta>
template <typename E>
    requires(std::meta::is_enum_type(^^E))
constexpr std::string_view enum_to_string(E value) {
    template for (constexpr auto enumerator : std::define_static_array(enumerators_of(^^E))) {
        if (value == [: enumerator :]) {
            return std::meta::identifier_of(enumerator);
        }
    }
    return "<unknown>";
}
int main() {
    enum class Color {RED, GREEN, BLUE} c{Color::BLUE};
    std::cout << enum_to_string(c) << std::endl;
}
```

目录

for 语句

do ... while 语句

switch 语句与枚举类型

逻辑运算符

与或非

https://en.cppreference.com/cpp/language/operator_logical

- 我们在上一章提到过，表达式`0 <= hour < 24`恒为`true`
  - 此前的解决方法是

```
if (hour >= 0) {
    if (hour < 24) {
        ...
    }
}
```

  - 但这样产生了额外的缩进，并且不便于理解
- 我们可以通过逻辑运算符（与或非）来进行逻辑运算
  - `if (0 <= hour && hour < 24)`
  - 与 (and, $\wedge$): 二者同为真才为真
  - 或 (or, $\vee$): 只要有真就为真
  - 非 (not, $\neg$): 真变假，假变真

C++ 逻辑运算符

- 与：`a && b`等价于`a and b`
- 或：`a || b`等价于`a or b`
- 非：`!a`等价于`not a`

真值表

$a \backslash b$	0	1
0	0	0
1	0	1

表 1: 与 (and)

$a \backslash b$	0	1
0	0	1
1	1	1

表 2: 或 (or)

a	0	1
$\neg a$	1	0

表 3: 非 (not)

$a \backslash b$	0	1
0	0	1
1	1	0

表 4: 异或 (xor)

- 逻辑运算是`bool`值之间的运算
- C++ 中也有位运算符 (`&`, `|`, `~`, `<<`, `>>`)，注意区分
  - 逻辑运算符两侧都会转成布尔型，并且没有逻辑异或

短路求值

- &&和||运算符在能确定计算结果后就不会执行后续表达式，该特性称为短路求值 (short-circuit evaluation)
  - 在&&中，若左侧为false，则右边不会计算

```
int a{2}, b{3}, c{3};
if (a && b++ >= 2 && c++ == 3) {
    c++;
}
std::cout << b << ' ' << c << std::endl; // 4 5
```

  - 在||中，若左侧为true，则右边不会计算

```
int a{2}, b{3}, c{3};
if (a || b++ >= 2 || c++ == 3) {
    c++;
}
std::cout << b << ' ' << c << std::endl; // 3 4
```

- 当后续表达式有副作用 (side-effect) 时需注意
  - 不建议代码依赖短路求值的副作用

运算符优先级

1. 小括号
2. 一元运算符（包括++，--，!，及正负号）
   - `!a * b`等价于`(!a) * b`
   - 因此一元运算符前后不留空格，以显示优先级高
3. 算术运算符（+，-，*，/，%）
4. 移位、流运算符（<<，>>）
   - `std::cout << a + b;`
   - `std::cout << (a < b);`
5. 关系比较（<，<=，>，>=，==，!=）
6. 按位与、异或、或（&，^，|）
7. 逻辑与、或（&&，||）
8. 条件（?:）
9. 赋值（=，+=，-=，*=，/=，%=）
10. 逗号

练习：C++ 的逻辑运算

小练习

写一个程序，不断读取用户输入的字母，并输出它是字母表中第几个。当用户输入非字母时，询问是否退出，若用户输入 Y 或者 y 则退出，N 则返回，否则重复询问。

练习：C++ 的逻辑运算

```cpp
// example.cpp
#include <iostream>
int main() {
    bool is_exit{false};
    do {
        std::cout << "Please enter a letter, or other character to exit: ";
        char c; std::cin >> c;
        bool is_lower{'a' <= c && c <= 'z'}, is_upper{'A' <= c && c <= 'Z'};
        if (is_lower) {
            std::cout << c << " is letter " << c - 'a' + 1 << '\n';
        } else if (is_upper) {
            std::cout << c << " is letter " << c - 'A' + 1 << '\n';
        } else {
            char answer;
            do {
                std::cout << "Do you want to exit? (Y/N): ";
                std::cin >> answer;
                switch (answer) {
                case 'Y': case 'y': is_exit = true; break;
                case 'N': case 'n': break;
                default: std::cout << "Invalid input!\n"; continue;
                }
                break;
            } while (true);
        }
    } while (!is_exit);
}
```

总结

- 结构化编程只需要用到三种结构
  - 顺序结构
  - 选择结构`if`
  - 循环结构`while`
- 选择结构的简便写法
  - 两个相反的条件`if (a) { ... } if (!a) { ... }`可以合并为 `if (a) { ... } else { ... }`
  - 多个等值`if (x == 1) { ... } else if (x == 2) { ... }`可以合并为 `switch (x) { case 1: ... case 2: ...}`
  - 根据条件取不同值`if (a) { x = 1; } else { x = 2;}`可以简写为 `x = (a) ? 1 : 2;`
- 循环结构的简便写法
  - 通过`for (int i = 1; i < 10; ++i) { ... }`写计数循环
  - 通过`do { ... } while (...);`写至少一次的循环
- 控制结构之间可以进行任意的堆叠、嵌套
- 枚举类型