



程序设计基础及语言 —— C++ 大学教程

第 4 章 控制语句（第 I 部分）、赋值、自增和自减运算符

邱远

东南大学 网络空间安全学院
yuanqiu@seu.edu.cn

2026 年 10 月



目录

算法与编程

if 语句

while 语句

缩写运算符



算法

算法 (Algorithm)

解决问题的步骤（包括执行的动作与执行顺序）

- 问题：猜一个 1-100 之间（包含）的整数
- 算法一：从 1 猜到 100（最差情况 100 次）
- 算法二：二分查找（最差情况 $\lceil \log_2(100) \rceil = 7$ 次）

先设计算法，再编写程序，好的算法设计能让编程实现事半功倍

案例：《雷神之锤 III》中的快速反平方根 ($\frac{1}{\sqrt{i}}$) 算法¹

```
i = 0x5f3759df - ( i >> 1 );
```

¹<https://zhuanlan.zhihu.com/p/445813662>



结构化编程

- 程序按照书写顺序依次执行，改变程序的执行顺序称为**控制转移**
- 传统编程依赖跳转 (Goto) 进行控制转移，容易导致代码混乱
 - C++ 兼容 C，因此也支持 **goto**

```
int abs(int x) {  
    if (x > 0) goto exit;  
    x = -x;  
exit: return x;  
}
```
 - Dijkstra 认为过度使用 goto 是有害的，建议采用控制结构 [2]
- Böhm 与 Jacopini 证明，任何程序都只需要用到三种控制结构 [1]
 - 顺序结构 (sequence structure)
 - 选择结构 (selection structure)
 - 循环结构 (repetition structure)

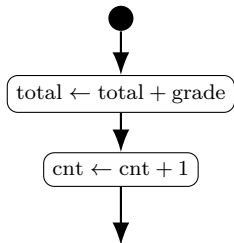


伪代码与 UML 活动图

- 算法与控制结构可以用伪代码 (Pseudocode) 或 UML 活动图 (Activity diagram) 表示, 二者都与具体编程语言无关

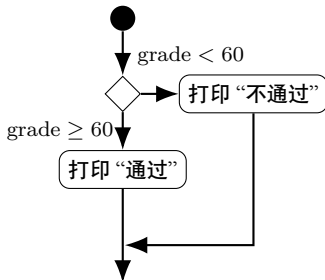
Algorithm 1 顺序结构

```
total ← total + grade
cnt ← cnt + 1
```



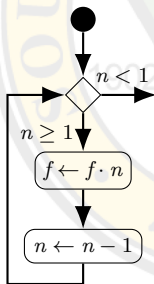
Algorithm 2 选择结构

```
if grade ≥ 60 then
  | 打印“通过”
else
  | 打印“不通过”
```



Algorithm 3 循环结构

```
while n ≥ 1 do
  | f ← f · n
  | n ← n - 1
```





目录

算法与编程

if 语句

while 语句

缩写运算符





回顾：第 2 章中的 if 语句

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

16

```
if (num1 == num2) cout << num1 << " == " << num2 << endl;
```

if 选择语句

```
if (布尔表达式) {  
    语句;  
} else {  
    语句;  
}
```

- 当布尔表达式为true时，执行if大括号内的语句；否则执行else大括号内的语句（若有），最后执行后续语句
- 布尔表达式外的小括号不能省略，如if (true)
- 语句应相对 if 缩进；若语句只有一句，则大括号可省略
 - 按照编程规范，即使只有一句，大括号也不应省略
 - 是否有空格空行不影响编译运行
- 若表达式非布尔型，会被转为bool，此时非 0 都为true



if ... else ...

- 此前当我们面临两个相反的情况时写两个 if

```
if (grade >= 60) std::cout << "Pass" << std::endl;
if (grade < 60) std::cout << "Fail" << std::endl;
```

- 将后者放入 else 中，不仅代码更清晰，也能让程序节约一次判断

```
if (grade >= 60) {
    std::cout << "Pass" << std::endl;
} else {
    std::cout << "Fail" << std::endl;
}
```

- if-else 可以嵌套²，例如

```
if (delta >= 0) {
    if (delta > 0) {
        std::cout << "Two Real Roots" << std::endl;
    } else {
        std::cout << "One Real Root" << std::endl;
    }
} else {
    std::cout << "No Real Root" << std::endl;
}
```

²好的编程习惯应该避免嵌套过多层 if-else



复合语句

- 复合语句 (Compound Statements) 又称代码块 (Code Block), 用于给多条语句一样的层级关系

```
if (delta >= 0)
    int x = std::sqrt(delta);
    std::cout << x << std::endl; // error: 'x' is undefined
```

- 任何能写单条语句的地方都能用大括号, 且大括号可以嵌套
 - 大括号可以为空, 但不要写 `if (a == 0) {} else { ... }`

- 由于 `else` 存在歧义可能, 即使只有一条语句也不建议省略大括号

```
if (delta >= 0)
    if (delta > 0)
        std::cout << "Two Real Roots" << std::endl;
    else // else corresponds to delta > 0
        std::cout << "One Real Root" << std::endl;
```

- 为免歧义, C++ 规定 dangling else 匹配最近的 if, 但可读性差
- 另一个好处是后续一行变多行时无需修改 `if` 所在行
- 例外情况: 若 `else` 后面只有一个 `if ... else ...`, 此时习惯连写

```
if (grade >= 90) {
} else if (grade >= 75) {
} else if (grade >= 60) {
} else {
}
```



避免多重嵌套（拓展）

考虑一个 setTime(int hour, int minute, int second); 函数

```

void setTime(int hour, int minute, int second) {
    if (hour >= 0) {
        if (hour < 24) {
            if (minute >= 0) {
                if (minute < 60) {
                    if (second >= 0) {
                        if (second < 60) {
                            time = hour * 3600 + minute * 60 + second;
                        } else
                            cerr << "second too large!" << endl;
                    } else
                        cerr << "second too small!" << endl;
                } else
                    cerr << "minute too large!" << endl;
            } else
                cerr << "minute too small!" << endl;
        } else
            cerr << "hour too large!" << endl;
    } else
        cerr << "hour too small!" << endl;
}

```

思考

为什么不能写 `if (0 <= hour < 24) ?`

```

void setTime(int hour, int minute, int second) {
    if (hour < 0) {
        cerr << "hour too small!" << endl; return;
    }

    if (hour >= 24) {
        cerr << "hour too large!" << endl; return;
    }

    if (minute < 0) {
        cerr << "minute too small!" << endl; return;
    }

    if (minute >= 60) {
        cerr << "minute too large!" << endl; return;
    }

    if (second < 0) {
        cerr << "second too small!" << endl; return;
    }

    if (second >= 60) {
        cerr << "second too large!" << endl; return;
    }

    time = hour * 3600 + minute * 60 + second;
}

```

通过 early return 代替层层嵌套，让主逻辑位于同一层级



区分赋值与比较

- 由于赋值运算符 `=` 有返回值，将 `==` 写成 `=` 并不会触发编译错误

```
if (x = 10) {  
    ...  
}
```

- 上述 if 永远成立，因为 `x = 10` 赋值会返回 `x`
- 避免该问题的方法之一是将常量写在判等的左边

```
if (10 == x) {  
    ...  
}
```

- 因为 `10 = x` 是不合法的表达（只能赋值给变量），会被编译器检查出来



条件运算符

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

- 我们想将学生的成绩存在 `std::string` 变量中

```
std::string s;
if (grade >= 60) s = "Pass";
else s = "Fail";
```

- 对这种常见的场景，C++ 提供了一种简便写法

条件运算符 (Conditional operator)

(条件) ? 表达式 1 : 表达式 2

- `std::string s{(grade >= 60) ? "Pass" : "Fail"};`
- **operator?**: 是 C++ 中唯一的三元 (ternary) 运算符，与其他运算符一样，计算后返回一个值

- 计算顺序为先条件，若为真则计算表达式 1，否则计算表达式 2
- 为了确定结果类型，表达式 1 与 2 的结果必须能转为相同类型
- 条件运算符优先级很低，仅高于赋值，低于比较等运算，因此除了赋值以外，外部一般都要加括号，内部一般不加括号
- 结合律为从右向左，相当于 else-if 嵌套

```
s = (grade >= 90) ? "A" : (grade >= 75) ? "B" : (grade >= 60) ? "C" : "D";
```



练习：C++ 的选择语句

小练习

请写程序读入三个整数 x, y, z ，输出其中的最大值。



练习：C++ 的选择语句

小练习

请写程序读入三个整数 x, y, z ，输出其中的最大值。

```
1 // example.cpp
2 #include <iostream>
3 int main() {
4     std::cout << "Enter three integers: " << std::endl;
5     int x, y, z;
6     std::cin >> x >> y >> z;
7     int max{x};
8     max = (max > y) ? max : y;
9     max = (max > z) ? max : z;
10    std::cout << "Max is: " << max << std::endl;
11 }
```



目录

算法与编程

if 语句

while 语句

缩写运算符





while 循环

<https://en.cppreference.com/cpp/language/while>

循环 (Loop) 用于重复执行一段程序，直到满足结束条件

- 依次输入全班 30 位同学的成绩（固定次数）
- 猜 1-100 的一个数（终止条件）

在 C++ 中，二者都可以通过 `while` 语句来实现

while 循环语句

```
while (循环条件) {  
    语句;  
}
```

- 与 `if` 语句除了关键词不同，整体格式类似
- 如果循环条件为 `true`，则执行大括号内的内容，然后再次判断循环条件，直到表达式为 `false`，执行后续语句
- 循环体内应当有能改变循环条件的语句，否则会形成“死循环”



程序一：计算 10 人的平均分（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 determineClassAverage() const;
12 private:
13     std::string courseName;
14 }; // end class

```

实现源文件 (GradeBook.cpp)

```

1  /** @file GradeBook.cpp */
2  #include "GradeBook.h"
3  #include <iostream>
4  GradeBook::GradeBook(std::string name) { setCourseName(name); }
5  void GradeBook::setCourseName(std::string name) {
6      if (name.size() <= 25) courseName = name;
7      else courseName = name.substr(0, 25);
8  }
9  std::string GradeBook::getCourseName() const { return courseName; }
10 void GradeBook::displayMessage() const {
11     std::cout << "Welcome to the grade book for\n"
12     << getCourseName() << "!\n" << std::endl;
13 }
14 void GradeBook::determineClassAverage() const {
15     int total = 0; // sum of grades
16     size_t counter = 0; // number of grades entered so far
17     while (counter < 10) {
18         std::cout << "Enter Grade: ";
19         int grade;
20         std::cin >> grade;
21         total = total + grade;
22         counter = counter + 1;
23     }
24     int average = total / 10;
25     std::cout << "\nTotal of all 10 grades is " << total
26     << ", Average is " << average << std::endl;
27 }

```



程序一：计算 10 人的平均分（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.determineClassAverage();
8  }
```

程序输出

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

Enter Grade: 67
Enter Grade: 78
Enter Grade: 89
Enter Grade: 67
Enter Grade: 87
Enter Grade: 98
Enter Grade: 93
Enter Grade: 85
Enter Grade: 82
Enter Grade: 100

Total of all 10 grades is 846, Average is 84



Line 15-23: 计数器控制的循环

```
15 int total = 0; // sum of grades
16 size_t counter = 0; // number of grades entered so far
17 while (counter < 10) {
18     std::cout << "Enter Grade: ";
19     int grade;
20     std::cin >> grade;
21     total = total + grade;
22     counter = counter + 1;
23 }
```

- `int total`; 用于统计一系列数值的和, 称为累加器 (accumulator)
- `size_t counter`; 用于统计循环次数, 称为计数器 (counter)
 - 若循环 n 次, 计数器取值通常为 $1, \dots, n$ 或者 $0, \dots, n-1$
 - 由于计数器不会为负数, 可采用非负类型 `size_t`, `unsigned int`
- 累加器和计数器都必须初始化, 否则其中可能为垃圾值
- 循环的组成: 初始化、循环条件、循环体、更新
 - `counter = counter + 1`; 更新了计数器, 为下次循环做准备
 - 当第 11 次判断循环条件时不满足, 退出循环
 - 注意此时 `counter` 值为 10



定数循环与不定数循环

- 能确定执行次数的循环称为定数循环 (definite loops)³
- 若我们无法提前确定输入的行数，则无法提前确定循环次数，此时称为不定数循环 (indefinite loops)，由其他条件判定结束

不确定输入个数的情况下，何时结束？

- 使用与输入有明显区分的标记值 (sentinel value)
 - 输入手机号，以 # 号键结束
 - 输入正分数，以 -1 标记结束
 - 标记值又称信号值 (signal value)、哑值 (dummy value)
- 自顶向下、逐步求精的程序开发
 - ① 计算全班平均成绩 = 读取输入 + 更新总和和人数 + 输出比值
 - ② 读取输入 = 循环读取，循环条件判断是否是标记值
 - ③ 更新总和和人数 = 初始化 + 循环内加法
 - ④ 输出比值 = 总和除以人数，需要判断人数不为零

³我们会在后续章节中学到，定数循环更常用for循环表示



程序二：计算任意人的平均分

由于接口与实现分离，我们无需修改头文件和 main.cpp，仅修改实现

```
1  /** @file GradeBook.cpp */
2  #include "GradeBook.h"
3  #include <iomanip>
4  #include <iostream>
5  GradeBook::GradeBook(std::string name) { setCourseName(name); }
6  void GradeBook::setCourseName(std::string name) {
7      courseName = (name.size() <= 25) ? name : name.substr(0,25);
8  }
9  std::string GradeBook::getCourseName() const { return courseName; }
10 void GradeBook::displayMessage() const {
11     std::cout << "Welcome to the grade book for\n" << getCourseName() << "!\n" << std::endl;
12 }
13 void GradeBook::determineClassAverage() const {
14     int total{}, grade{}; unsigned counter{};
15     std::cout << "Enter grade or -1 to quit: ";
16     std::cin >> grade;
17     while (grade != -1) {
18         total = total + grade;
19         counter = counter + 1;
20         std::cout << "Enter grade or -1 to quit: ";
21         std::cin >> grade;
22     }
23     if (counter == 0) std::cout << "No grades entered\n"; // edge case
24     else {
25         double average = static_cast<double>(total) / counter;
26         std::cout << "Average is " << std::setprecision(2) << std::fixed << average << std::endl;
27     }
28 }
```



Line 14-22: 标记控制的循环

```
14 int total{}, grade{}; unsigned counter{};
15 std::cout << "Enter grade or -1 to quit: ";
16 std::cin >> grade;
17 while (grade != -1) {
18     total = total + grade;
19     counter = counter + 1;
20     std::cout << "Enter grade or -1 to quit: ";
21     std::cin >> grade;
22 }
```

- 依然有累加器total和计数器counter
- 17 行的终止条件换成了对输入值grade的判断
- 如果不想将同样的std::cout写两遍，也可以搭配if

```
int total{}, grade{}; size_t counter{};
while (grade != -1) { // first iteration grade == 0
    std::cout << "Enter grade or -1 to quit: ";
    std::cin >> grade;
    if (grade != -1) {
        total = total + grade;
        counter = counter + 1;
    }
}
```



Line 25: 类型转换

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

25

```
double average = static_cast<double>(total) / counter;
```

类型转换

`static_cast<目标类型>(变量名);`

- 整数之间的除法是整除，只有一方为浮点数时才能调用小数除法
- 有时类型转换是隐式的，例如
 - `double average = total; average = average / counter;`
 - `double average = total * 1. / counter;`
 - `if (counter)`
 - 显式转换类型是一个好习惯
- 也可以通过转换类型来避免溢出
 - `long sum = static_cast<long>(num1) + num2;`
- 二进制下某些小数是不精确的（如 $\frac{1}{3}$ ），因此通常不要做 double 之间是否相等的判断
 - `2. / 3 == 1 - 1. / 3; // false`



Line 26: 浮点数输出

<https://en.cppreference.com/cpp/io/manip>

26

```
std::cout << "Average is " << std::setprecision(2) << std::fixed << average << std::endl;
```

- `std::cout` 输出浮点数时默认保留 6 位有效数字
 - `std::cout << 1.2345678 << std::endl; // output 1.23457`
- 可以通过流操纵符设定以下输出格式，均为四舍五入
- 输出方式⁴
 - `std::defaultfloat` // 默认，最多 n 个有效数字，舍去多余的零
 - `std::fixed` // 小数点后固定 n 位
 - `std::scientific` // 科学计数法，小数点后固定 n 位
- 位数⁵（有参数的设置需要额外 `#include <iomanip>`）
 - `std::setprecision(n)`
- 其他（默认均为 no）
 - `std::showpoint` // 保证 n 个有效数字（不舍 0），一定有小数点
 - `std::boolalpha` // 将 `bool` 类型打印为 `true/false`

⁴<https://cplusplus.com/reference/ios/fixed/>

⁵<https://cplusplus.com/reference/ios/setprecision/>



运用嵌套控制语句

- 与if类似，while的循环体一样可以嵌套

编写程序解决问题

- 输入：10 个学生的考试成绩（1 为通过，2 为不通过），每次用“Enter result”提示
- 输出：通过和未通过的考生人数；若 8 个及以上学生通过考试，额外输出“Bonus to instructor!”
- 10 个学生 → 次数固定，计数循环
- 1 为通过，2 为不通过 → 选择语句
 - 出现 1 与 2 以外的情况？→ 非法输入处理
- 统计人数 → 计数器/累加器
- 额外输出 → 选择语句



程序三：统计考试通过人数

```
1  /** @file main.cpp */
2  #include <iostream>
3  int main() {
4      unsigned int total_cnt{};
5      unsigned int pass_cnt{};
6      unsigned int fail_cnt{};
7
8      while (total_cnt < 10) {
9          std::cout << "Enter result (1 = pass, 2 = fail): ";
10         int grade;
11         std::cin >> grade;
12         if (grade == 1) {
13             pass_cnt = pass_cnt + 1;
14         } else if (grade == 2) {
15             fail_cnt = fail_cnt + 1;
16         } else {
17             std::cerr << "Unexpected input grade = " << grade << "!\n";
18             return -1;
19         }
20         total_cnt = total_cnt + 1;
21     }
22
23     std::cout << "Passed " << pass_cnt << "\nFailed " << fail_cnt << std::endl;
24     if (pass_cnt >= 8) {
25         std::cout << "Bonus to instructor!" << std::endl;
26     }
27 }
```



练习：C++ 的 while 循环

小练习

请写程序读入一个大于 1 的整数 x ，输出 x 是否为质数^a。

^a质数只能被分解为 1 和自己相乘，而没有其他因式分解



练习：C++ 的 while 循环

小练习

请写程序读入一个大于1的整数 x ，输出 x 是否为质数。

```
1 // example.cpp
2 #include <iostream>
3 int main() {
4     std::cout << "Enter a positive integer: ";
5     unsigned int x;
6     std::cin >> x;
7     if (x <= 1) {
8         std::cerr << "Unexpected input x = " << x << std::endl;
9         return -1;
10    }
11    unsigned int factor = 2;
12    while (factor <= x / factor) {
13        if (x % factor == 0) {
14            std::cout << "Divisor found: " << x << " = "
15                << factor << " * " << x / factor << std::endl;
16            return 0;
17        }
18        factor = factor + 1;
19    }
20    std::cout << "No divisor found, " << x << " is prime!";
21 } // end function main
```



目录

算法与编程

if 语句

while 语句

缩写运算符





缩写运算符

- C++ 提供了一些缩写运算符来让程序更简洁

- 复合赋值运算符

- `x += 3;` 等价于 `x = x + 3;`
- `x -= 4;` 等价于 `x = x - 4;`
- `x *= 5;` 等价于 `x = x * 5;`
- `x /= 6;` 等价于 `x = x / 6;`
- `x %= 7;` 等价于 `x = x % 7;`
- 和之前一样，赋值运算符都是右结合，并返回左侧变量

```
y += x += 3; // x increase by 3, then y increase by x
x += 3 += 4; // error
```

- 当增减量为 1 时，`x += 1;` 和 `x -= 1;` 可以进一步缩写，称为自增与自减运算符

- 前置时，先将变量增/减 1，然后返回变量的新值
- `++x;` 等价于 `x += 1;` 等价于 `x = x + 1;`
- `--x;` 等价于 `x -= 1;` 等价于 `x = x - 1;`
- 后置时，返回当前值，再将变量增/减 1
- `x++;` 等价于 `{ int y = x; x = x + 1; return y; }`
- `x--;` 等价于 `{ int y = x; x = x - 1; return y; }`



自增运算符的效率（拓展）

- 许多时候自增运算符的返回值并不重要，例如更新计数器时

```
while (counter < 10) {  
    ...  
    ++counter; // counter++;  
}
```

- 理论上，此时使用前置的++counter;效率更高
 - 前置的++c可以直接原地修改，后置的c++需要创建临时对象
 - 事实上，大部分现代编译器都会做优化，产生一样的汇编代码⁶
- 在需要用到值时，需要关注二者的差异

```
int counter = 0;  
while (counter++ < 10) { ... } // 10 iterations {0 ... 9}  
int cnt = 0;  
while (++cnt < 10) { ... } // 9 iterations {1 ... 9}
```

⁶[https://stackoverflow.com/questions/79942587/](https://stackoverflow.com/questions/79942587/which-form-of-incrementation-is-best-for-speed-in-c)



总结

- 理解用于解决问题的算法概念和有效使用控制结构的重要性✓
 - 算法、伪代码、UML 活动图
 - 自顶向下、逐步求精
- 灵活运用顺序结构、选择结构和循环结构✓
 - `if () { ... } else { ... }`
 - `while () { ... }`
 - 复合语句、嵌套
- 熟练掌握和运用算术运算符、自增自减、赋值和逻辑运算符✓
 - 已经学习: `:: . + - * / % << >> > >= < <= == != =`
 - 本章新增: `static_cast<> ++ -- ?: += -= *= /= %=`
 - 逻辑运算符将在下一章介绍
- 理解运算符的优先级✓
 - 自行对照书本图 4.20 记忆, 不确定的情况下加括号, 尤其注意输入输出/比较和其他的关系



参考文献

- [1] Corrado Böhm and Giuseppe Jacopini. “Flow diagrams, turing machines and languages with only two formation rules”. In: *Commun. ACM* 9.5 (May 1966), pp. 366–371. ISSN: 0001-0782. DOI: 10.1145/355592.365646. URL: <https://doi.org/10.1145/355592.365646>.
- [2] Edsger W. Dijkstra. “Letters to the editor: go to statement considered harmful”. In: *Commun. ACM* 11.3 (Mar. 1968), pp. 147–148. ISSN: 0001-0782. DOI: 10.1145/362929.362947. URL: <https://doi.org/10.1145/362929.362947>.