



# 第四章 程序设计基本结构

## 模块4.1：循环和关系表达式

主讲教师：同济大学电子与信息工程学院 陈宇飞  
同济大学电子与信息工程学院 龚晓亮



# 目录

- for循环
- while循环
- do-while循环
- 嵌套循环
- 循环的应用案例



# 目录

- for循环

1. for循环的组成部分
2. 修改步长
3. 递增(++)和递减(--)运算符
4. 前缀和后缀格式
5. 组合赋值运算符
6. 复合语句（语句块）
7. 逗号运算符
8. 关系表达式
9. 赋值和比较
10. for语句的扩展使用



# 1.1 for循环的组成部分

```
//forloop.cpp
#include<iostream>
int main()
{
    using namespace std;
    int i;    //create a counter
    //initialize; test; update
    for(i = 0; i < 5; i++)
        cout << "C++ knows loops. \n";
    cout << "C++ knows when to stop. \n";
    return 0;
}
```

## 基本步骤:

- 设置初始值 (loop initialization)
- 执行测试 (loop test)
- 执行循环操作 (loop body)
- 更新用于循环的值 (loop update)

```
C++ knows loops.
C++ knows loops.
C++ knows loops.
C++ knows loops.
C++ knows loops.
C++ knows when to stop.
```



# 1.1 for循环的组成部分

```
//num_test.cpp --use numeric test in for loop
```

```
#include<iostream>
```

```
int main()
```

```
{
```

```
    using namespace std;
```

```
    cout << "Enter the starting countdown value: ";
```

```
    int limit;
```

```
    cin >> limit;
```

```
    int i;
```

```
    for(i = limit; i; i--) //quits when i is 0
```

```
        cout << "i = " << i << endl;
```

```
    cout << "Done now that i = " << i << "\n";
```

```
    return 0;
```

```
}
```

```
Enter the starting countdown value: 4
```

```
i = 4
```

```
i = 3
```

```
i = 2
```

```
i = 1
```

```
Done now that i = 0
```

```
Enter the starting countdown value: 0
```

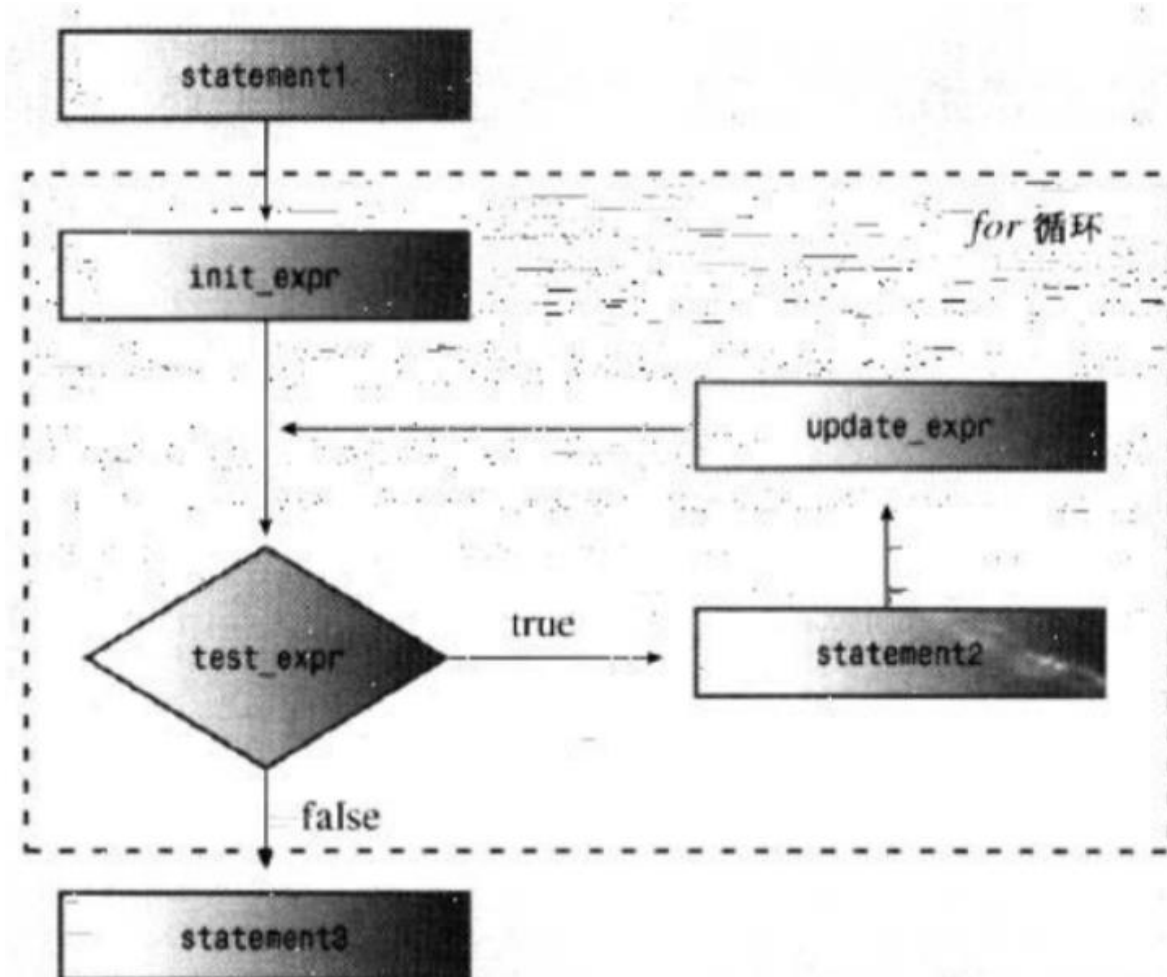
```
Done now that i = 0
```



# 1.1 for循环的组成部分

✓for语句的控制部分使用3个表达式

```
statement1  
for (int_expr; test_expr; update_expr)  
    statement2  
statement3
```





# 1.1 for循环的组成部分

## ✓表达式和语句

- 任何值或者任何有效的值和运算符的组合都是表达式
- 每个C++表达式都有值
- C++将赋值表达式的值定义为左侧成员的值

10;     //值为10的表达式

22 + 27; //值为49的表达式

x = 20;   //值为20的赋值表达式

maids = (cooks = 4) + 3; //表达式cooks=4的值为4，因此maids的值为7

x = y = z = 0;   //赋值运算符是从右向左结合的，因此首先将0赋值给z，

然后将z=0赋值给y，以此类推，完成快速将若干变量设置为相同值的操作

```
//express.cpp  --values of expressions
#include<iostream>
int main()
{
    using namespace std;
    int x;
    cout << "The expression x = 100 has the value ";
    cout << (x = 100) << endl;
    cout << "Now x = " << x << endl;
    cout << "The expression x < 3 has the value ";
    cout << (x < 3) << endl;
    cout << "The expression x > 3 has the value ";
    cout << (x > 3) << endl;
    cout.setf(ios_base::boolalpha); //命令cout显示true和false，而不是1和0
    cout << "The expression x < 3 has the value ";
    cout << (x < 3) << endl;
    cout << "The expression x > 3 has the value ";
    cout << (x > 3) << endl;
    return 0;
}
```

```
The expression x = 100 has the value 100
Now x = 100
The expression x < 3 has the value 0
The expression x > 3 has the value 1
The expression x < 3 has the value false
The expression x > 3 has the value true
```





# 1.1 for循环的组成部分

✓表达式的副作用 (side effect)

➤ 操作改变了内存中数据的值

- 判定表达式  $x = 100$

副作用：修改被赋值者的值（将100赋值给x）

- 判定表达式  $x + 15$

计算新值但不修改x，无副作用

- 判定表达式  $++x + 15$

副作用：x的值加1



# 1.1 for循环的组成部分

## ✓表达式和语句

➤ 加分号，所有表达式都可以成为语句

- 表达式 `age = 100`
- 语句 `age = 100; //表达式语句`
- `rodents + 6; //valid, but useless, statement`

➤ 语句删除分号，不一定是表达式

- `int toad; //statement`
- `eggs = int toad * 1000; //invalid, not an expression`
- `cin >> int toad; //can't combine declaration with cin`
- `int fx = for (i = 0; i < 4; i++)  
cout << i; //not possible`



# 1.1 for循环的组成部分

## ✓修改规则

### ➤原来的句法

```
for (expression; expression; expression)  
    statement
```

### ➤调整后也允许的句法

```
for (for-init-statement condition; expression)  
    statement
```

```
for (int i = 0; i < 5; i++) //变量只存在于for语句中!  
    cout << "C++ knows loops. \n";  
cout << i << endl; //oops! i no longer defined!
```



## 1.2 修改步长

✓可以通过修改更新表达式来修改步长

```
//bigstep.cpp  --count as directed
#include<iostream>
int main()
{
    using namespace std;
    cout << "Enter an integer: ";
    int by;
    cin >> by;
    cout << "Counting by " << by << "s;\n";
    for (int i = 0; i < 100; i = i + by)
        cout << i << endl;
    return 0;
}
```

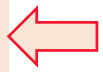
```
Enter an integer: 17
Counting by 17s;
0
17
34
51
68
85
```



# 1.3 递增(++)和递减(--)运算符

```
#include <iostream>
int main()
{
    using namespace std;
    int a = 20;
    int b = 20;
    cout << "a  = " << a << "; b = " << b << endl;
    cout << "a++ = " << a++ << "; ++b = " << ++b << endl;
    cout << "a  = " << a << "; b = " << b << endl;
    return 0;
}
```

a++使用a的当前值计算表达式，然后将a的值加1；  
++b先将b的值加1，然后使用新的值来计算表达式。



```
a  = 20; b = 20
a++ = 20; ++b = 21
a  = 21; b = 21
```



# 1.3 递增(++)和递减(--)运算符

## ✓运算符的区别

```
int x = 5;  
int y = ++x; //change x, then assign to y  
           // y is 6, x is 6
```

```
int z = 5;  
int y = z++; //assign to y, then change z  
           // y is 5, z is 6
```

## ✓不要在同一条语句对同一个值递增或递减多次，规则将模糊不清

`x = 2 * x++ * (3 - ++x);` //不同系统不同编译器将生成不同结果，不建议用



# 1.4 前缀格式和后缀格式

## ✓前后缀的区别

```
for (n = lim; n > 0; --n)
    ...
```

```
for (n = lim; n > 0; n--)
    ...
```

- 从逻辑上说没有区别：表达式的值未被使用，因此只存在副作用（值减1）
- 执行速度有差别
  - **前缀**：将值减1，然后返回结果；
  - **后缀**：首先复制一个副本，将其减1，然后将复制的副本返回。
  - 因此：对于内置类型，采用哪种格式差别不大；但对于用户定义的类型，如果有用户定义的递增和递减运算符，作为**前缀格式的效率**高（后续课程）。



# 1.5 组合赋值运算符

✓合并加法和赋值的运算符

`i = i + by;`



`i += by;`

操作符	作用（）
<code>+=</code>	将L+R赋给L
<code>-=</code>	将L-R赋给L
<code>*=</code>	将L*R赋给L
<code>/=</code>	将L/R赋给L
<code>%=</code>	将L%R赋给L





## 1.6 复合语句（语句块）

- ✓使用花括号来表示复合语句（语句块）。如果在语句块中定义一个新的变量，则仅当程序执行该语句块中的语句时，该变量存在。执行完该语句块，变量将被释放
- ✓注意，在外部定义的变量，在语句块内部也是被定义了（后续详解）

```
int sum = 0;
int number;
for (int i =1; i <= 5; i++)
{
    cout << "Value: " << i << ": ";
    cin >> number;
    sum += number;
}
//i no longer defined
```

//block starts here

//block ends here



## 1.6 复合语句（语句块）

✓仅使用缩进但省略花括号有时是不符合预期的

```
for (int i =1; i <= 5; i++)  
    cout << "Value: " << i << ": "; //loop ends here  
    cin >> number;                    //after the loop  
    sum += number;  
cout << "Five exquisite choices indeed! ";
```



## 1.6 复合语句（语句块）

- ✓如果在语句块中定义一个新的变量，则仅当程序执行该语句块中的语句时，该变量才存在

```
int x = 20;
{                                     //block starts
    int y = 100;
    cout << x << endl;             //ok
    cout << y << endl;             //ok
}                                     //block ends
cout << x << endl;                 //ok
cout << y << endl;                 //invalid, won't compile
```



## 1.6 复合语句（语句块）

- ✓如果在语句块中声明一个变量，而外部语句块中也有一个相同名称的变量，在声明位置到内部语句块结束的范围内，新变量将隐藏旧变量；然后旧变量再次可见

```
int x = 20;           //original x
{                     //block starts
    cout << x << endl; //use original x
    int x = 100;       //new x
    cout << x << endl; //use new x
}                     //block ends
cout << x << endl;    //use original x
...
```



# 1.7 逗号运算符

✓逗号运算符允许将多个表达式放到句法只允许放一个表达式的地方

```
++j, --i    //two expressions count as one for syntax purposes
```

✓逗号并不总是逗号运算符

```
int i, j;    //comma is a separator here, not an operator
```

✓确保先计算第一个表达式，然后计算第二个表达式

```
i = 20, j = 2* i;    //i is set to 20, then j set to 40
```

✓C++规定，逗号表达式的值是最后的值

```
cats = 17, 240;    //cats变量值为17，整个表达式值为240  
cats = (17, 240);    //cats变量值为240
```

✓逗号运算符的优先级最低



# 1.8 关系表达式

✓C++提供了6种关系运算符，如果比较结果为真，则其值为true，否则为false

```
for (x = 20; x > 5; x--) //continue while x is greater than 5
for (x = 1; y != x; ++x) //continue while y is not equal to x
for (cin >> x; x == 0; cin >> x) //continue while x is 0
```

操作符	含义
<	小于
<=	小于或等于
==	等于
>	大于
>=	大于或等于
!=	不等于



## 1.9 赋值和比较

✓不要使用=来比较两个量是否相等，而要使用==

```
musicians == 4 //comparison
```

整个表达式值为true 或者 false

```
musicians = 4 //assignment
```

整个表达式值为4

```
for (i = 0; i == 20; i++)  
    cout << "Hello World!\n";
```

```
for (i = 0; i = 20; i++)  
    cout << "Hello ALPD!\n";
```

**//死循环！！**

智能编程环境下会给提示，普通编译环境尚不能

➤ 思考下述两种写法，哪种更好？

```
for (i = 0; i == 20; i++)
```

```
for (i = 0; 20 == i; i++)
```



# 1.10 for语句的扩展使用

✓for语句的三处表达式可以省略，可以是简单表达式，也可以是多个简单表达式组合形式的逗号表达式

```
int i, sum=0;
for(i=1; i<=100; i++)
    sum=sum+i;
```

```
int i, sum;
for(i=1, sum=0; i<=100; i++)
    sum=sum+i;
```

```
int i, sum;
for(i=1, sum=0; i<=100; sum=sum+i, i++);
```

```
int i, sum;
for(i=1, sum=0; i<=100; sum=sum+i++);
```

最后加一个分号，表示for循环无语句序列/含一个空语句





# 目录

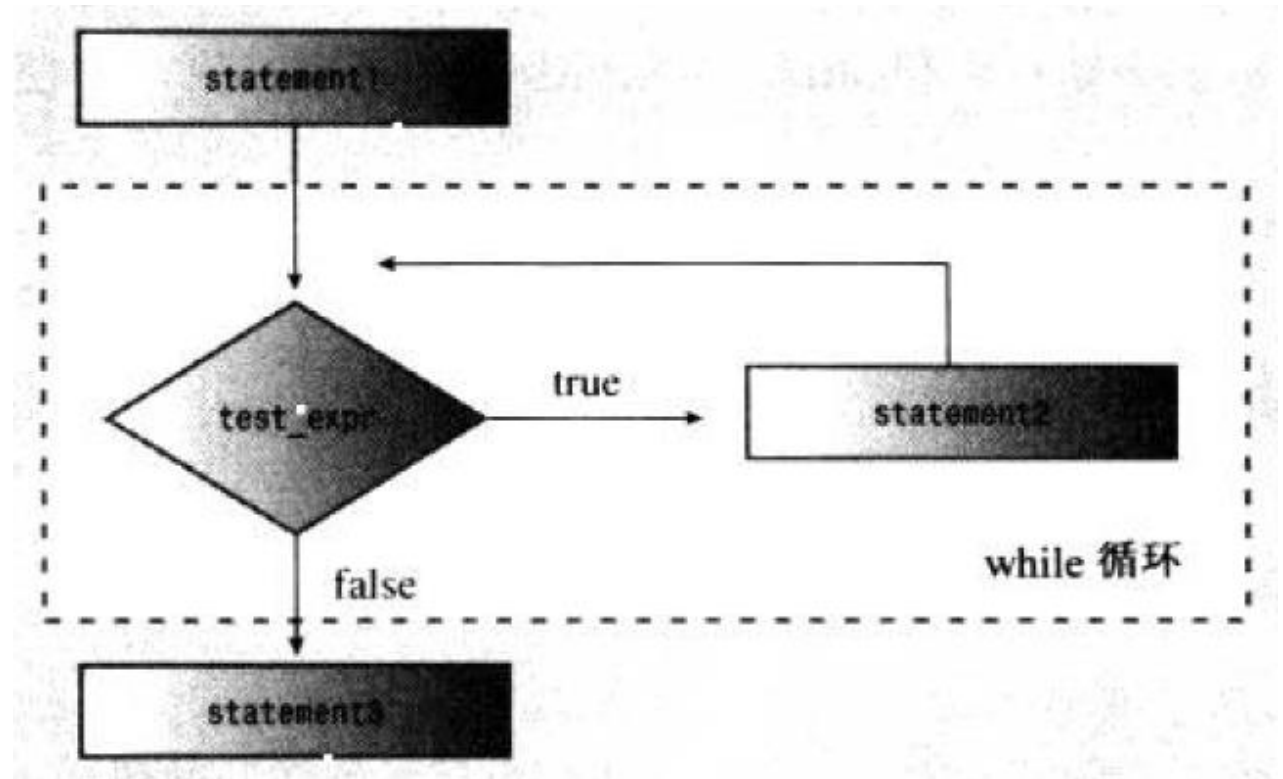
- while 循环

1. for 与 while
2. 副作用和顺序点
3. 编写延时循环

## 2.1 for 与 while

✓while循环：没有初始化和更新部分的for循环

```
statement1  
while (test_expr)  
    statement2  
statement3
```





## 2.1 for 与 while

✓C++中，for和while循环本质上相同

```
statement1  
for (int_expr; test_expr; update_expr)  
    statement2  
statement3
```



```
int_expr;  
while (test_expr)  
{  
    statements  
    update_expr;  
}
```

```
while (test_expr)  
    body
```



```
for (; test_expr;)  
    body
```



## 2.1 for 与 while

✓for和while循环的不同之处:

- 在for循环中省略了测试条件时, 将认为条件为true;
- 在for循环中, 可使用初始化语句声明一个局部变量;
- 如果循环体包括continue语句, 情况将有所不同 (后续课程)

```
for (;;)    //永真  
    body
```

```
for (int i = 0; i < 5; i++)  
    body
```

✓通常的做法:

- 使用for循环来为循环计数: 因其可以将所有相关信息 (初始值、终止值和更新计数器的方法) 放在同一个地方
- 无法预先知道循环将执行的次数时, 常使用while循环



## 2.2 副作用和顺序点

- ✓副作用指的是在计算表达式时对某些东西（如存储在变量中的值）进行了修改
- ✓顺序点是程序执行过程中的一个点，需确保对所用的副作用都进行了评估
- ✓在C++中，语句中的分号就是一个顺序点，这意味着程序处理下一条语句之前，赋值运算符、递增运算符和递减运算符执行所有的修改都必须完成。另外，任何完整的表达式末尾都是顺序点

```
while (guest++ < 10)  
    cout << guest << endl;
```

- 1，确保副作用（guests加1）在程序进入cout之前完成
- 2，通过后缀++确保将guest同10比较后再将其值加1



## 2.3 编写延时循环

//waiting.cpp --using clock() in a time-delay loop

```
#include <iostream>
```

```
#include <ctime>
```

```
int main()
```

```
{    using namespace std;
```

```
    cout << "Enter the delay time, in second: ";
```

```
    float secs;
```

```
    cin >> secs;
```

```
    clock_t delay = secs * CLOCKS_PER_SEC; //convert to clock ticks
```

```
    cout << "starting\n";
```

```
    clock_t start = clock();
```

```
    while (clock() - start < delay); //wait until time elapses
```

```
        cout << "done \n";
```

```
    return 0;
```

```
}
```

- 头文件ctime定义了符号常量CLOCKS\_PER\_SEC，该常量等于每秒钟包含的系统时间单位数
- clock()函数返回程序开始执行后所用的系统时间

- clock()函数返回类型不尽相同。将变量声明为clock\_t类型，编译器把它转换为long、unsigned int或适合系统的其他类型



# 目录

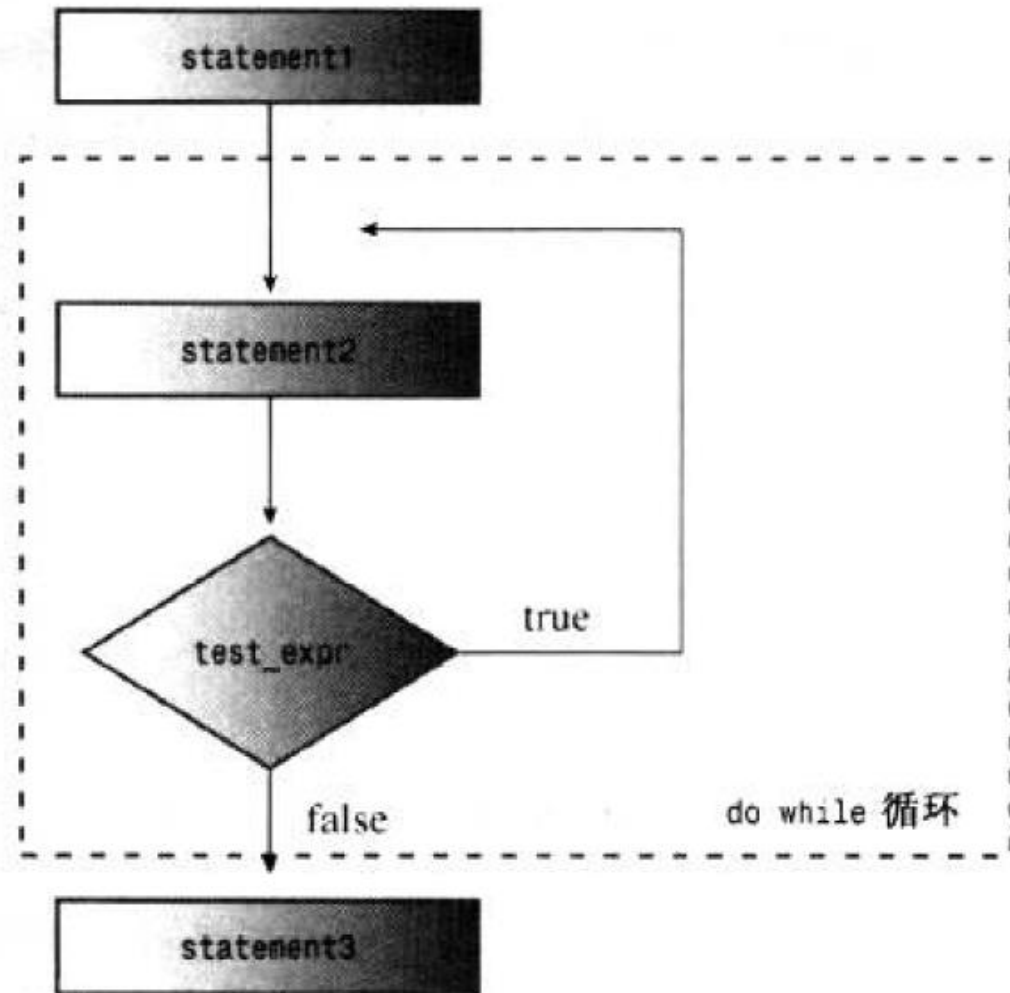
- do-while循环
  1. do-while循环的结构
  2. do-while和while循环

## 3.1 do-while循环的结构

✓do-while循环：出口条件循环

```
statement1  
do  
    statement2  
while (test_expr);  
statement3
```

循环通常至少执行一次！！







## 3.2 do-while和while循环

### ✓二者的区别

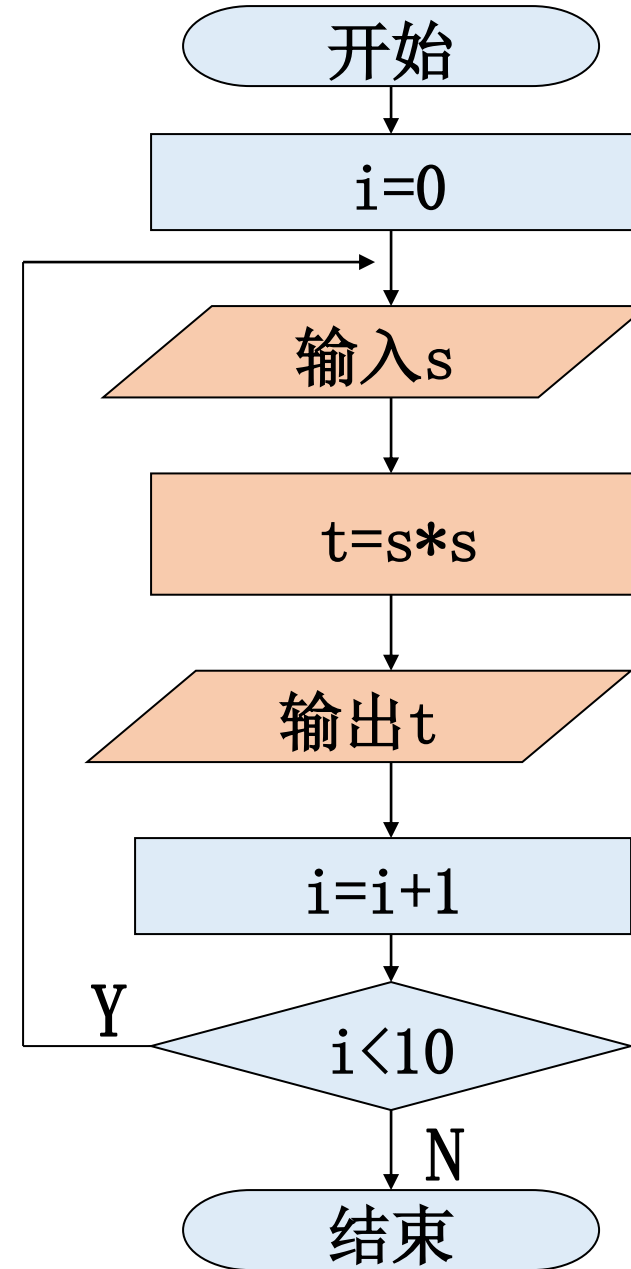
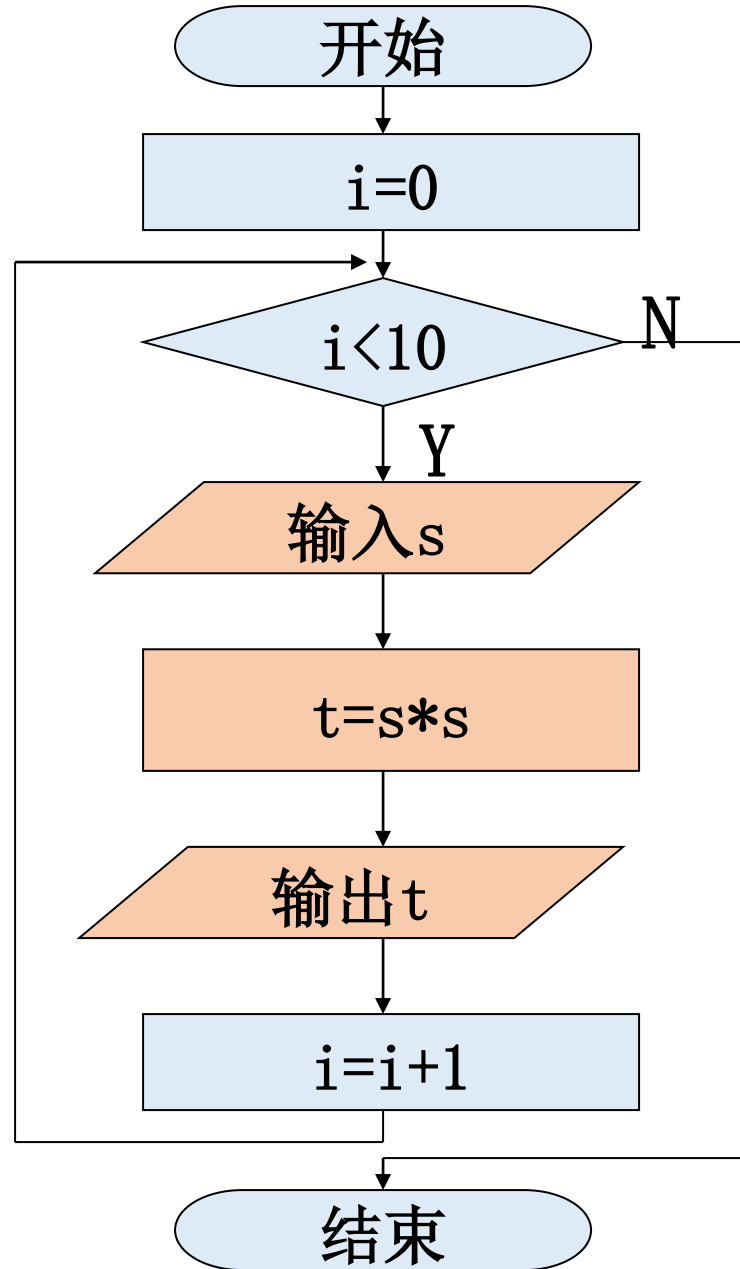
```
do  
    body  
while (test_expression);
```

```
while (test_expr)  
    body
```

- do-while: 首先执行循环体, 然后再判定测试表达式, 决定是否继续执行循环  
(至少执行一次)
- while: 首先判定测试表达式, 再决定是否进入执行循环体  
(可能一次都不执行)



# 例：输出10个数字的平方



问题：当 $i$ 的初值为20时，左右的区别？



## 例：输出10个数字的平方

```
#include <iostream>
int main()
{
    using namespace std;
    int i = 0, t, s;
    while (i < 10)
    {
        cout << "请输入s的数值: ";
        cin >> s;
        t = s * s;
        cout<< "t=" << t << endl;
        i++;
    }
    return 0;
}
```

```
do
{
    cout << "请输入s的数值: ";
    cin >> s;
    t = s * s;
    cout<< "t=" << t << endl;
    i++;
} while (i < 10);
```

当i的初值为20时，左侧程序不执行循环，右侧程序执行一次循环



# 目录

- 嵌套循环
  - 嵌套循环的形式
  - 嵌套循环的执行



# 4.1 嵌套循环的形式

✓ 嵌套循环可以有多种组合形式

三重循环的例子:

for(...) {  
    ① while(...) {  
        (一) for(...) {  
            ...  
        }  
        (二) ...  
    }  
} ②

外for循环的其它语句，与while并列；  
while循环的其它语句，与内for并列

<pre>while(...) {     while(...) {         ...     } }</pre>	<pre>do {     do{         ...     } while(...); } while(...);</pre>
<pre>for(...) {     for(...) {         ...     } }</pre>	<pre>while(...) {     do(...) {         ...     } while(...); }</pre>
<pre>for(...) {     while(...) {         ...     } }</pre>	<pre>do (...) {     for(...) {         ...     } } while(...);</pre>



## 4.2 嵌套循环的执行

✓外层循环每执行一次，内层循环都要执行一遍

```
for(i=1;i<=100;i++)  
    for(j=1;j<=100;j++)  
        cout << i*j << ' ' ;
```

cout 语句执行了10000遍

```
for(i=1;i<=100;i++) {  
    cout << i << endl;  
    for(j=1;j<=100;j++) {  
        cout << i*j << endl;  
        for(k=1;k<=100;k++)  
            cout << i*j*k << ' ' ;  
        cout << j*i << endl;  
    }  
    cout << i << endl;  
}
```

红语句执行了100 遍

蓝语句执行了10000 遍

粉语句执行了1000000 遍



# 目录

- 循环的应用案例

例：用公式  $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$  求  $\pi$  值 (到最后一项绝对值  $< 10^{-7}$  为止)



```
int main()
{
    int s=1;
    double n=1, t=1, pi=0;
    while(fabs(t) > 1e-7) {
        pi = pi + t;    //pi为累加和（实际表示pi/4）
        n = n + 2;      //n为分母
        s = -s;         //分子在正负1间变化
        t = s/n;        //每项的值
    }
    pi = pi * 4;
    cout << pi << endl;
    return 0;
}
```





例：用公式  $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$  求  $\pi$  值 (到最后一项绝对值  $< 10^{-7}$  为止)

//迭代部分如下

```
while(fabs(t) > 1e-7) {  
    pi = pi + t;    //pi为累加和（实际表示pi/4）  
    n = n + 2;      //n为分母  
    s = -s;         //分子在正负1间变化  
    t = s/n;        //每项的值  
}
```

$$\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} \rightarrow \frac{1}{1} + \frac{-1}{3} + \frac{1}{5} + \frac{-1}{7}$$

## 对比不同精度的执行时间



```
#include <iostream>
#include <iomanip>      //格式输出
#include <cmath>        //fabs
#include <windows.h>    //取系统时间
using namespace std;
int main()
{
    int    s=1;
    double n=1, t=1, pi=0;

    LARGE_INTEGER tick, begin, end;
    QueryPerformanceFrequency(&tick);    //取计数器频率
```



## 对比不同精度的执行时间

```
int main()  
{
```

```
...
```

```
QueryPerformanceCounter (&begin); //取开始时定时器计数
```

```
while(fabs(t) > 1e-6) {  
    pi = pi + t;  
    n  = n + 2;  
    s  = -s;  
    t  = s/n;  
}
```

```
QueryPerformanceCounter (&end); //取结束时定时器计数
```

```
/* 执行到此, 求得 pi/4 的值 */
```

## 对比不同精度的执行时间



```
int main()  
{
```

```
...
```

```
/* 执行到此，求得 pi/4 的值 */
```

```
pi = pi * 4;
```

```
cout << "n=" << setprecision(10) << n << endl;
```

```
cout << "pi=" << setiosflags(ios::fixed) << setprecision(9) << pi << endl;
```

```
cout << "计数器频率：" << tick.QuadPart << "Hz "
```

```
    << "时钟计数：" << end.QuadPart - begin.QuadPart << endl;
```

```
cout << setprecision(6) <<
```

```
    (end.QuadPart - begin.QuadPart)/double(tick.QuadPart) << "秒" << endl;
```

```
return 0;
```

```
}
```



例：用公式  $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$  求  $\pi$  值(到最后一项绝对值 $<10^{-7}$ 为止)

(1) n, t, pi为double型

精度为1e-6: 3.141590654 (0.002631s) n= 1000001

1e-7: 3.141592454 (0.029328s) n= 10000001

1e-8: 3.141592634 (0.324512s) n= 100000001

1e-9: 3.141592652 (3.208342s) n=1000000001

(因为机器配置不同，时间值可能不同)

(2) n, t, pi为float型

精度为1e-6: 3.141593933 (0.010819) n= 1000001

1e-7: 3.141596556 (0.115626) n=10000001

1e-8: 无结果，为什么？

不同编译器的运行时间不同，掌握基本规律即可



# 总结

## • for循环

1. for循环的组成部分
2. 修改步长
3. 递增(++)和递减(--)运算符
4. 前缀和后缀格式
5. 组合赋值运算符
6. 复合语句（语句块）
7. 逗号运算符
8. 关系表达式
9. 赋值和比较
10. for语句的扩展使用

## • while循环

1. for 与 while
2. 副作用和顺序点
3. 编写延时循环

## • do-while循环

1. do-while循环的结构
2. do-while和while循环

## • 嵌套循环

1. 嵌套循环的形式
2. 嵌套循环的执行

## • 循环的应用案例