for循环

阶乘

阶乘

- $n! = 1 \times 2 \times 3 \times 4 \times \ldots \times n$

阶乘

- n! = 1×2×3×4×…×n
- 写一个程序，让用户输入n，然后计算输出n!

阶乘

- n! = 1×2×3×4×…×n
- 写一个程序，让用户输入n，然后计算输出n!
- 变量：

阶乘

- n! = 1×2×3×4×…×n
- 写一个程序，让用户输入n，然后计算输出n!
- 变量：
  - 显然读用户的输入需要一个int的n，然后计算的结果需要用一个变量保存，可以是int的factor，在计算中需要有一个变量不断地从1递增到n，那可以是int的i

```
int n = in.nextInt();
int factor = 1;
int i=1;
while ( i<=n )
{
 factor = factor * i;
 i = i + 1;
}
System.out.println(factor);
```

```
int n = in.nextInt();
int factor = 1;
int i=1;
while ( i<=n )
{
 factor = factor * i;
 i = i + 1;
}
System.out.println(factor);
```

不能算很大的阶乘，为什么

```
int n = in.nextInt();
int factor = 1;
int i=1;
for ( i=1; i<=n; i=i+1 )
{
 factor = factor * i;
}
System.out.println(factor);
```

小套路

- 做求和的程序时，记录结果的变量应该初始化为0，而做求积的变量时，记录结果的变量应该初始化为1

for循环

for循环像一个计数循环：设定一个计数器，初始化它，然后在计数器到达某值之前，重复执行循环体，而每执行一轮循环，计数器值以一定步进进行调整，比如加1或者减1

```
for ( i=0; i<5; i=i+1 ) {
    System.out.println(i);
}
```

```
for（ 初始化 ； 条件； 单步动作 ）{
}
```

1. 第一个部分是一个初始化，可以定义一个新的变量：int count=10或者直接赋值：i=10。

2. 第二个部分是循环维持的条件。这个条件是先验的，与while循环一样，进入循环之前，首先要检验条件是否满足，条件满足才执行循环；条件不满足就结束循环。

3. 第三个部分是步进，即每轮执行了循环体之后，必须执行的表达式。通常我们在这里改变循环变量，进行加或减的操作。

for = 对于

- for (count=10; count>0; count=count-1)

- 就读成：“对于一开始的count=10，当count>0时，重复做循环体，每一轮循环在做完循环体内语句后，使得count递减。”

- 循环控制变量i只在循环里被使用了，在循环外面它没有任何用处。因此，我们可以把变量i的定义写到for语句里面去

```
int n = in.nextInt();
int factor = 1;
for ( int i=1; i<=n; i=i+1 )
{
 factor = factor * i;
}
System.out.println(factor);
```

try

try

- 1×1还是1，所以程序的循环不需要从1开始，那么改成从多少开始合适呢？这样修改之后，程序对所有的n都正确吗？这样的改动有价值吗？

try

- 1×1还是1，所以程序的循环不需要从1开始，那么改成从多少开始合适呢？这样修改之后，程序对所有的n都正确吗？这样的改动有价值吗？
- 除了可以从1乘到n来计算n!，还可以从n乘到1来计算吧？试试把循环换个方向来计算n。这时候，还需要循环控制变量i吗？

```java
int n = in.nextInt();
int factor = 1;
int i=1;
while ( i<=n )
{
 factor = factor * i;
 i = i + 1;
}
System.out.println(factor);
```

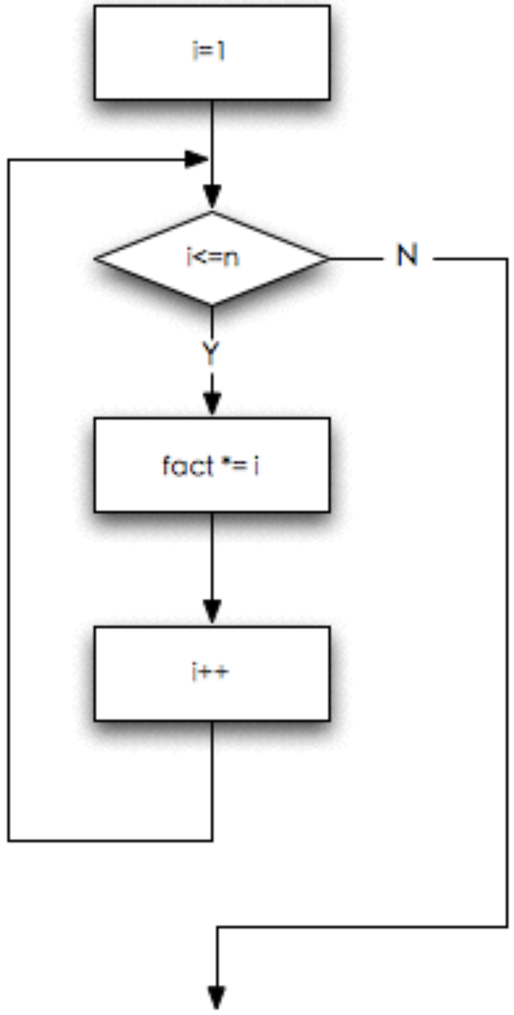
```java
int n = in.nextInt();
int factor = 1;
for ( int i=1; i<=n; i=i+1
{
 factor = factor * i;
}
System.out.println(factor)
```

for == while

```
for ( int i=1; i<=n; i=i+1 )
{
  factor = factor * i;
}
```

=

```
int i=1;
while ( i<=n )
{
  factor = factor * i;
  i = i + 1;
}
```



for循环

for (初始动作; 条件; 每轮的动作) {

}

- for中的每一个表达式都是可以省略的

for (; 条件;) == while (条件)

空循环

```
for ( i=0; i<10; i++ );

for ( i=0; i<10; i++ )
    System.out.println(i);
```

空循环

```
for ( i=0; i<10; i++ );
```

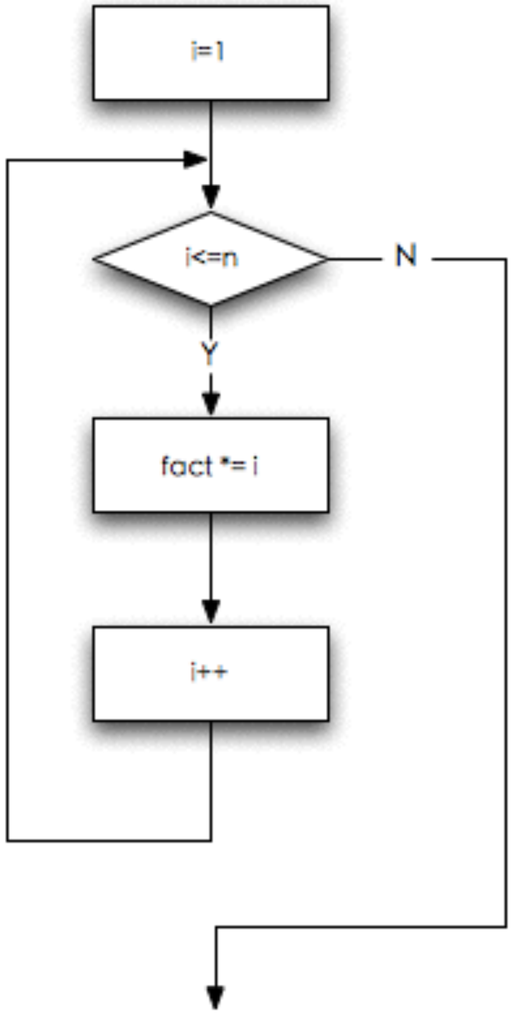
强烈建议：只要是for语句，就一定跟上一对大括号

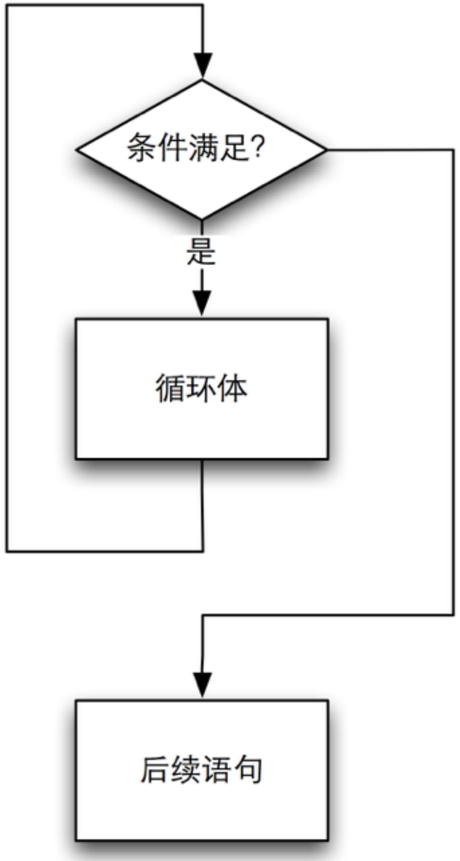
```
for ( i=0; i<10; i++ )
    System.out.println(i);
```

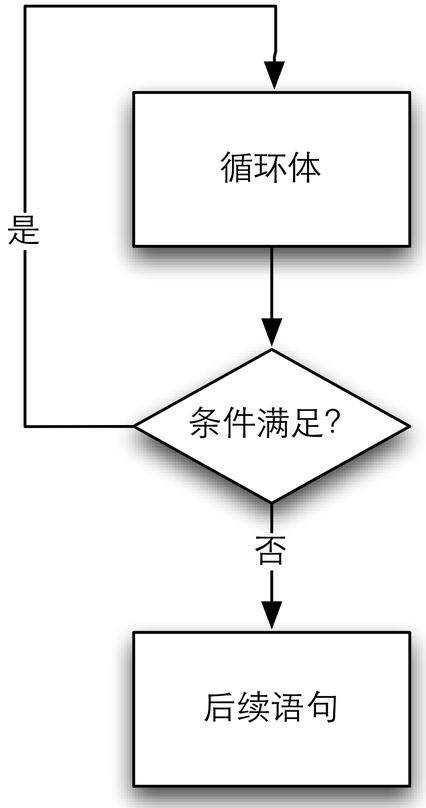
循环次数

- for (i=0; i<n; i=i+l)
- 则循环的次数是n，而循环结束以后，i的值是n。循环的控制变量i，是选择从0开始还是从l开始，是判断i<n还是判断i<=n，对循环的次数，循环结束后变量的值都有影响

三种循环







Tips for loops

- 如果有固定次数，用for
- 如果必须执行一次，用do_while
- 其他情况用while

复合赋值

复合赋值

- 5个算术运算符，+ - * / %，可以和赋值运算符“=”结合起来，形成复合赋值运算符：“+=”、“-=”、“*=”、“/=”和“%=”
  - total += 5;
  - total = total + 5;
- 注意两个运算符中间不要有空格

复合赋值

复合赋值

- total += (sum+100)/2;

复合赋值

- total += (sum+100)/2;
  - total = total + (sum+100)/2;

复合赋值

- total += (sum+100)/2;
  - total = total + (sum+100)/2;
- total * = sum+12;

复合赋值

- total += (sum+100)/2;
  - total = total + (sum+100)/2;
- total * = sum+12;
  - total = total*(sum+12);

复合赋值

- total += (sum+100)/2;
  - total = total + (sum+100)/2;
- total * = sum+12;
  - total = total*(sum+12);
- total /= 12+6;

复合赋值

- total += (sum+100)/2;
  - total = total + (sum+100)/2;
- total * = sum+12;
  - total = total*(sum+12);
- total /= 12+6;
  - total = total / (12+6);

递增递减运算符

- “++”和“--”是两个很特殊的运算符，它们是单目运算符，这个算子还必须是变量。这两个运算符分别叫做递增和递减运算符，他们的作用就是给这个变量+1或者-1。
  - count++;
  - count += 1;
  - count = count + 1;

前缀后缀

- ++和--可以放在变量的前面，叫做前缀形式，也可以放在变量的后面，叫做后缀形式。
- a++的值是a加1以前的值，而++a的值是加了1以后的值，无论哪个，a自己的值都加了1了。

计算

- a = 14;
- t1 = a++;
- t2 = ++a;

前缀后缀

表达式	运算	表达式的值
count++	给count加1	count原来的值
++count	给count加1	count+1以后的值
count--	给count减1	count原来的值
--count	给count减1	count-1以后的值