计算机科学基础I
——for循环

东南大学 集成电路学院 朱彬武
E-mail: bwzhu@seu.edu.cn

回顾——switch-case语句

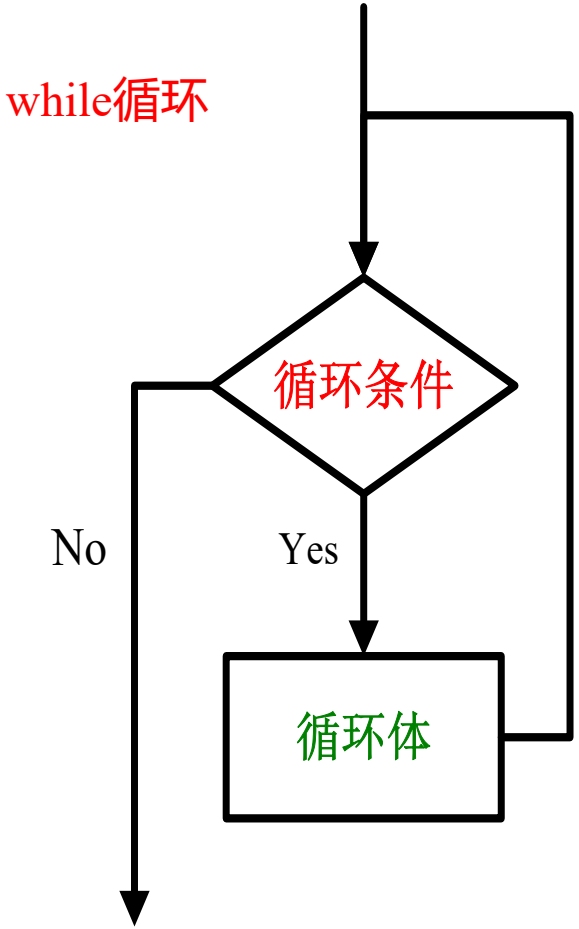
编写程序将一个百分制成绩转换为五分制成绩，转换规则

- 大于等于90分为A；
- 小于90且大于等于80为B；
- 小于80且大于等于70为C；
- 小于70且大于等于60为D；
- 小于60为E

```
// 用score / 10 来得到十位数，方便分段
switch (score / 10) {
    case 10:  // 100分，利用没有break的贯穿特性输出A
    case 9:
        cout << "A" << endl;
        break;
    case 8:
        cout << "B" << endl;
        break;
    case 7:
        cout << "C" << endl;
        break;
    case 6:
        cout << "D" << endl;
        break;
    default:
        cout << "E" << endl;
}
```

- 控制表达式只能是整数或字符型的结果，不能是浮点数或字符串
- case后面必须跟常量，不能是变量，常量可以是常数，也可以是常数计算的表达式

回顾——while循环



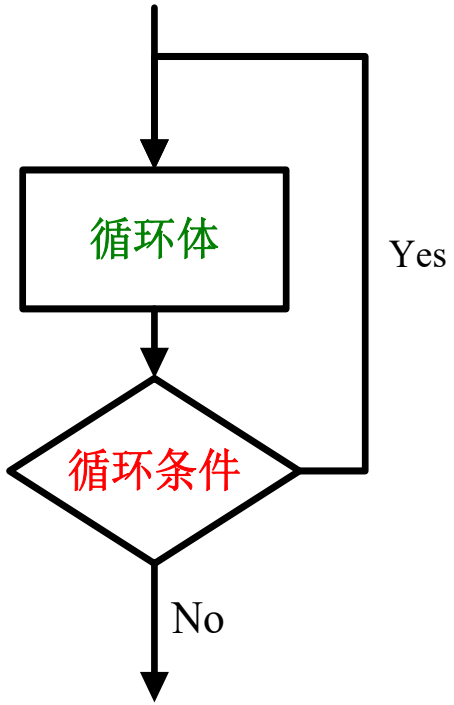
- while语句

```
while（<条件表达式>）{
        循环体
}
```

- 当条件为真，不断执行{}内的循环体，条件为假则跳过
- 循环体内一定要有改变条件的机会，否则会陷入死循环

回顾——do-while循环

do-while循环



- do-while语句

```
do{
        循环体
}while(<条件表达式>);
```

- 在进入循环的时候不做检查，而是在执行完一轮循环体的代码之后，再来检查循环的条件是否满足，如果满足则继续下一轮循环，不满足则结束循环。

- 注意while后面要有分号，表明do-while语句的结束

for循环

计算阶乘

- $n! = 1 \times 2 \times 3 \times \ldots \times n$
- 写一个程序，让用户输入n，然后计算输出n!
- 如何用程序实现?

在写循环的时候先要想清楚需要哪些变量：

- 条件变量
- 累积（累加or累乘）变量
- 状态变量
- ……

计算阶乘——while循环实现


```
int n;
int fact = 1;
int i = 1;
cin >> n;
while(i <= n){
    fact *= i;
    i++;
}
```

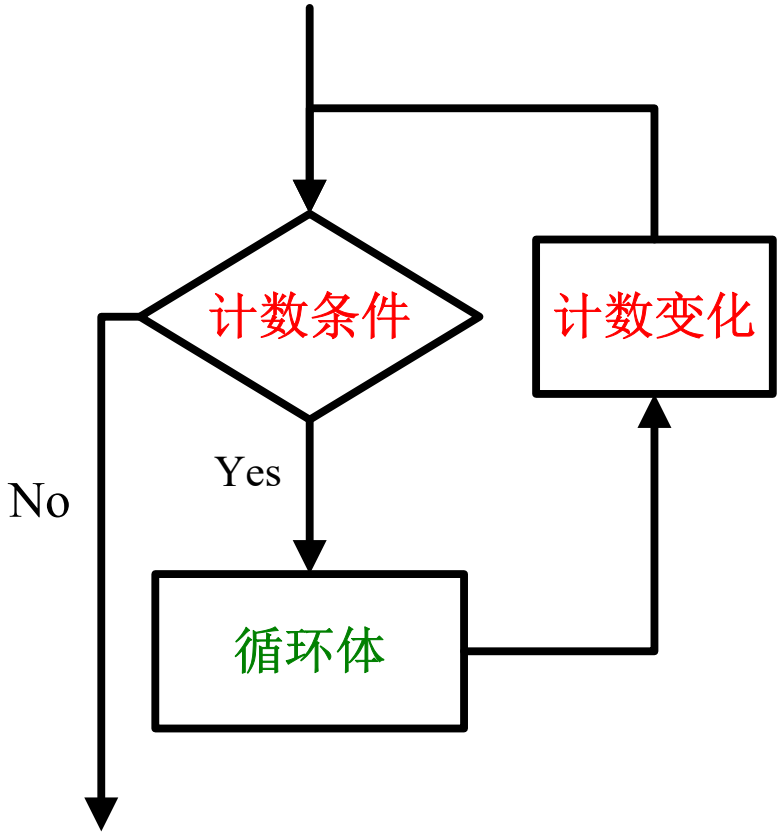
对于这种明确知道次数的循环，C++语言还提供了另一种实现语句——for循环

计算阶乘——for循环实现


```
int n;
int fact = 1;
int i;
cin >> n;
for(i = 1; i <= n; i++){
    fact *= i;
}
```

- 初始条件：i = 1
- 循环继续的条件：i <= n
- 循环每轮的动作：i++

for循环流程图



- for循环就像一个计数循环：设定一个计数器，初始化它，然后在计数器到达某个值之前，重复执行循环体
- 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循环语句


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

- for语句

  for（初始化计数变量；计数条件；计数变化）{
  循环体
  }

- 首先初始化计数变量，当计数条件成立时，重复做循环体，每一轮循环在做完循环体内语句后，执行计数变化

输出调试大法

- 输出调试大法：在程序运行过程中，用 cout 打印关键信息，帮助我们理解程序的执行流程、定位错误原因
- 为什么输出调试有用？可以帮助我们看到变量的真实值，知道程序卡在哪一步，不需要调试的情况下快速定位问题。
- 输出什么最有用？
  - 变量值（尤其是循环、条件中的）
  - 程序执行进度（进入函数、分支）
  - 关键逻辑结果（计算、判断）
- 技巧
  - 加前缀让输出更清晰：cout << "[DEBUG] i=" << i << endl;
  - 用分隔线区分不同阶段：cout << "----- start the loop -----" << endl; 调试完记得删除或注释掉这些输出

for循环更好的写法


```
int n;
int fact = 1;
cin >> n;
for(int i = 1; i <= n; i++){
    fact *= i;
}
```

通常循环控制变量i只在循环里被使用，在循环外面没有任何用处。因此，可以把变量i的定义写到for循环里面去。

多余的一轮循环


```
int n;
int fact = 1;
cin >> n;
for(int i = 1; i <= n; i++){
    fact *= i;
}
```

```
int n;
int fact = 1;
cin >> n;
for(int i = 2; i <= n; i++){
    fact *= i;
}
```

- 第一轮循环做的是1×1，结果仍然是1，似乎没有必要。是否可以省去这一轮循环?
- 那么i改成从多少合适呢？修改之后，程序对所有的n都正确吗?

尝试培养优化循环的思维

- 有了循环之后意味着你可以做更复杂的事情，但是不同方法达成同样的目标需要用到的循环次数可能差异很大

- 以简单的排序功能为例，排序借助循环来实现，但不同的实现方法效率差别很大，以对10万个随机数排序为例：

排序方法	时间（秒）	计算复杂度
冒泡排序	40.224	$O(n^2)$
选择排序	34.125	$O(n^2)$
插入排序	21.671	$O(n^2)$
归并排序	6.908	O(nlogn)
快速排序	0.139	O(nlogn)

可以省略的初始化计数变量

- 除了可以从1乘到n来计算n!，还可以从n乘到1来计算，这时候该怎么实现?

```
int n;
int fact = 1;
cin >> n;
for(int i = n; i > 1; i--){
    fact *= i;
}
```

从n开始做阶乘

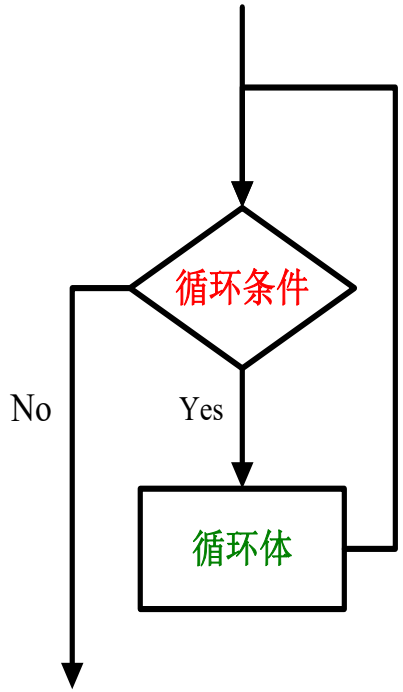
```
int n;
int fact = 1;
cin >> n;
for(n = n; n > 1; n--){
    fact *= n;
}
```

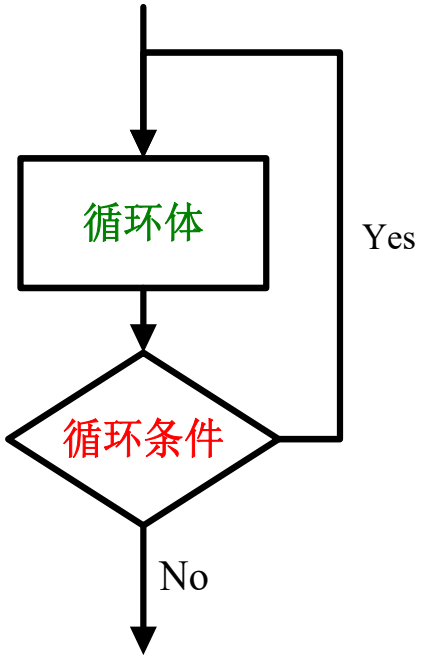
直接将n本身作为计数器

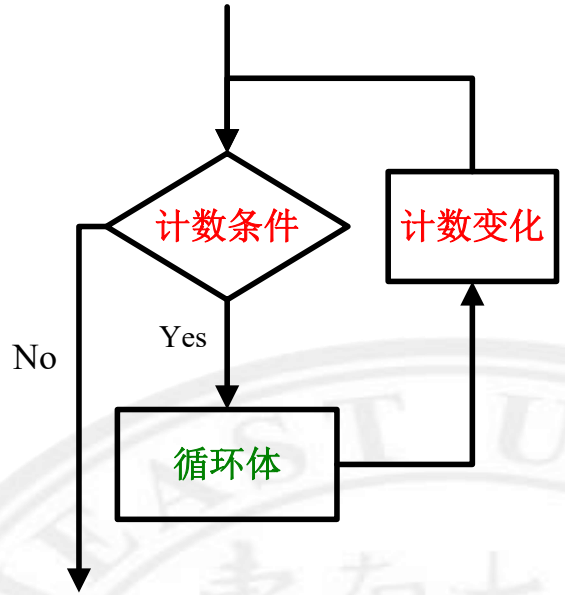
```
int n;
int fact = 1;
cin >> n;
for(; n > 1; n--){
    fact *= n;
}
```

省略初始化计数器，但是不能省略;

三种循环







如何选择使用哪种循环，一般遵循：

（1）如果有固定次数，用for循环（2）如果必须执行一次，用do-while

（3）其他情况用while

循环控制：break，continue

判断一个数是否是素数

- 写一段程序判断一个数是否是素数
- 素数的定义：只能被1和自己整除的数，不包括1
  - 2、3、5、7、11、13、17...

```
int num;
cin >> num;

for(   ){


}
```

程序实现1.0


```
int num;
cin >> num;

for(int i = 2; i < num; i++){
    if(num % i == 0){
        cout << "不是素数" << endl;
    }
}

......
```

- 如果这么写的话，似乎对于合数，程序正常。但是如是素数呢？输出“是素数”放哪里？
- 另外，如果一个数是合数的话，程序看起来会反复执行“不是素数”，有点奇怪。

程序实现2.0——加入标记变量


```
int num;
int isPrime = 1;
cin >> num;
for(int i = 2; i < num; i++){
    if(num % i == 0){
        isPrime = 0;
    }
}
if(isPrime == 1){
    cout << "是素数" << endl;
}
else{
    cout << "不是素数" << endl;
}
```

- 把变量isPrime作为标记，最后根据变量是0还是1判断是否是素数。（设定标记变量（flag）用于记录某种状态，这是编程的一个常用技巧）
- 但是仍然存在的问题是：假设输入是6，当i为2时，由于可以整除，已经可以判断输入不是素数了，那么还有必要让循环继续下去吗？

程序实现最终版——加入break语句


```
int num;
int isPrime = 1;
cin >> num;
for(int i = 2; i < num; i++){
    if(num % i == 0){
        isPrime = 0;
        break;
    }
}
if(isPrime == 1){
    cout << "是素数" << endl;
}
else{
    cout << "不是素数" << endl;
}
```

- break语句在switch-case里的作用是跳出switch-case语句
- 在循环里的作用则是跳出循环，此时当输入是6，i为2时，由于可以整除，已经可以判断输入不是素数了，利用break直接结束整个循环

continue语句


```
int num;
int isPrime = 1;
cin >> num;
for(int i = 2; i < num; i++){
    if(num % i == 0){
        isPrime = 0;
        continue;
    }
    cout << i << endl;
}
```

- continue语句：跳过循环这一轮剩下的语句进入下一轮
- 此时当num为6时，输出应该是：

4
5

形象理解break和continue

- 音乐老师让一排同学轮流唱歌：
  - 如果第二位同学唱得很难听，大家都听不下去了，于是老师发出continue指令，此时该同学停止唱歌，下一位同学继续唱歌；
  - 如果第三位同学仍然唱得很难听，大家觉得唱歌活动可以结束了，于是老师发出break指令，此时该同学停止唱歌，下一位同学也不需要继续唱歌了。

嵌套的循环

输出100以内的素数


```
int num;
int isPrime = 1;
cin >> num;
for(int i = 2; i < num; i++){
    if(num % i == 0){
        isPrime = 0;
        break;
    }
}
if(isPrime == 1){
    cout << "是素数" << endl;
}
else{
    cout << "不是素数" << endl;
}
```

已经有了一个判断是否是素数的程序，基于这段代码，怎么改可以实现我们想要的功能?

嵌套的循环


```
int num = 2;
int isPrime;
for(int num = 2; num < 100; num++){
    isPrime = 1;
    for(int i = 2; i < num; i++){
        if(num % i == 0){
            isPrime = 0;
            break;
        }
    }
    if(isPrime == 1){
        cout << num << endl;
    }
}
```

- 嵌套的循环：循环里面还有循环
- 注意点：
  - 计数变量要用不同的变量
  - num步进以后要重置isPrime

输出前50个素数

- 如果要输出前50个素数，似乎我们无法明确知道num的范围
- 根据我们选择循环的原则，这时候用while循环会更合适

```
int num = 2;
int cnt = 0;
int isPrime;
while(...){
    isPrime = 1;
    for(int i = 2; i < num; i++){
        if(num % i == 0){
            isPrime = 0;
            break;
        }
    }

    if(isPrime == 1){
        cout << num << endl;
        //...
    }
    //...
}
```

输出前50个素数——程序实现


```
int num = 2;
int cnt = 0;
int isPrime;
while(cnt < 50){
    isPrime = 1;
    for(int i = 2; i < num; i++){
        if(num % i == 0){
            isPrime = 0;
            break;
        }
    }

    if(isPrime == 1){
        cout << num << endl;
        cnt++;
    }
    num++;
}
```

- cnt变量：用于计数
- cnt<50： while循环条件
- cnt++：当且仅当num是素数时才会执行
- num++：相当于for循环的计数步进

跳出嵌套的循环——goto语句

凑硬币

- 如何用2角、5角和7角的硬币凑出n角的金额
- $2x + 5y + 7z = n$
- 如何解这个方程？ 枚举法
- 程序结构：三层循环
  - x=0, y=0, z=1…n/7
  - x=0, y=1, z=1…n/7
  - …
  - x=1, y=0, z=1…n/7
  - …

枚举所有可能


```
int total = 14;
int two, five, seven;
for(two = 0; two <= total/2; two++){
    for(five = 0; five <= total/5; five++){
        for(seven = 0; seven <= total/7; seven++){
            if(two*2 + five*5 + seven*7 == total){
                cout << "2角=" << two
                    << ", 5角=" << five
                    << ", 7角=" << seven << endl;
            }
        }
    }
}
```

现在程序会枚举所有可能的情况，但是如果我只需要找到一个解，怎么办？

只找到一个解——尝试加入一个break


```
int total = 14;
int two, five, seven;
for(two = 0; two <= total/2; two++){
    for(five = 0; five <= total/5; five++){
        for(seven = 0; seven <= total/7; seven++){
            if(two*2 + five*5 + seven*7 == total){
                cout << "2角=" << two
                    << ", 5角=" << five
                    << ", 7角=" << seven << endl;
                break;
            }
        }
    }
}
```

加入一个break够了吗?

不够！因为break只能跳出它所在的那层循环

只找到一个解——接力break


```
int total = 14;
int two, five, seven;
int exit = 0;
for(two = 0; two <= total/2; two++){
    for(five = 0; five <= total/5; five++){
        for(seven = 0; seven <= total/7; seven++){
            if(two*2 + five*5 + seven*7 == total){
                cout << "2角=" << two
                    << ", 5角=" << five
                    << ", 7角=" << seven << endl;
                exit = 1;
                break;
            }
        }
        if(exit)
            break;
    }
    if(exit)
        break;
}
```

借助标记变量exit来记录是否满足条件的状态。

由于有三层循环，因此需要三个break才能结束整个嵌套循环。

只找到一个解——goto语句


```
    int total = 14;
    int two, five, seven;
    int exit = 0;
    for(two = 0; two <= total/2; two++){
        for(five = 0; five <= total/5; five++){
            for(seven = 0; seven <= total/7; seven++){
                if(two*2 + five*5 + seven*7 == total){
                    cout << "2角=" << two
                        << ", 5角=" << five
                        << ", 7角=" << seven << endl;
                    goto out;
                }
            }
        }
    }
out:
    return 0;
```

- 语法规则：

  goto <标号>

  <标号>:

- 应用场景：跳出多层的嵌套循环
- 由于goto语句降低程序可读性，并且容易引发内存泄漏的问题，基本不会用goto