



中國人民大學
RENMIN UNIVERSITY OF CHINA

信息学院
SCHOOL OF INFORMATION

程序设计1荣誉课程

3. 程序控制语句与枚举法

授课教师：游伟 副教授

授课时间：周一08:00 – 09:30, 周四16:00 – 17:30 (明德新闻楼0201)

上机时间：周四18:00 – 21:00 (理工配楼二层5号机房)

课程主页：<https://rucsesec.github.io/programming>

引子：博饼游戏

【背景】博饼是闽南地区盛行的中秋传统民俗活动。游戏时，需要6个骰子和一个大瓷碗。一般会进行多轮投掷，根据每轮投掷的骰子点数特征累计积分。如果某一轮投掷，出现规定特征以外的点数，那么游戏结束，否则继续进行。多轮投掷结束后，根据累计积分判定奖次。具体的点数特征和积分见下页。

【任务】编写程序，进行博饼游戏的积分统计。

【输入格式】第一行是一个正整数 n ($1 \leq n < 256$)，代表最多进行 n 轮投掷；
往后至多 n 行，每行6个正整数，代表每个骰子的点数

【输出格式】一个正整数，代表最终的累计积分（十六进制格式输出）

【输入样例】 6

【输出样例】： 20

4 1 4 4 5 4

1 1 1 1 1 1

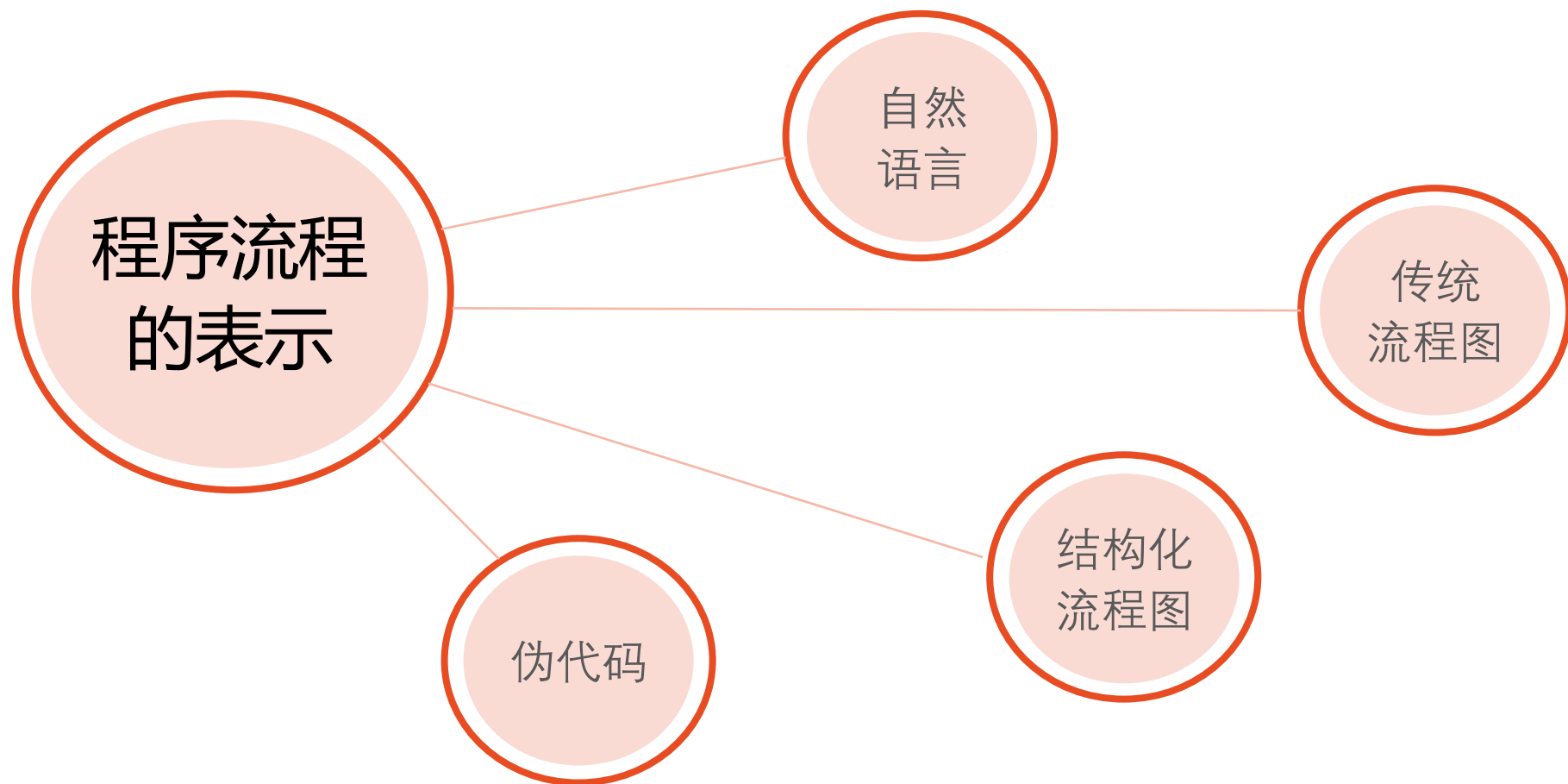
引子：博饼游戏

官级	别名	特征	积分
状元	插金花		2048
状元	红六勃		1024
状元	遍地锦		512
状元	黑六勃		256
状元	五红		128
状元	五子		64
状元	四红		32
榜眼	对堂		16
探花	三红		8
进士	四进		4
举人	二举		2
秀才	一秀		1
平民	出局	其它	-1

目录

1. 程序流程的表示
2. 关系运算/逻辑运算/条件运算
3. 分支语句
4. 循环语句
5. 枚举法解题

3.1 程序流程的表示



3.1.1 几个示例

■ 示例1：求 $1 \times 2 \times 3 \times 4 \times 5$

算法步骤

S1: 先求1乘以2，得到结果2

S2: 将步骤1得到的乘积2再乘以3，得到结果6

S3: 将6再乘以4，得24

S4: 将24再乘以5，得120

若题目改为: 求 $1 \times 3 \times 5 \times 7 \times 9 \times 11$

算法步骤

S1: 令 $p=1$ ，或写成 $1 \Rightarrow p$ (表示将1存放在变量 p 中)

S2: 令 $i=3$ 或写成 $3 \Rightarrow i$ (表示将2存放在变量 i 中)

S3: 使 p 与 i 相乘，乘积仍放在变量 p 中，可表示为: $p*i \Rightarrow p$

S4: 使 i 的值加 2 即 $i+2 \Rightarrow i$

S5: 若 $i \leq 11$ ，返回S3；否则，结束
或者 若 $i > 11$ ，结束；否则，返回S3

用这种方法表示的算法具有一般性、通用性和灵活性

3.1.1 几个示例

■ 示例2：求 $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + \frac{1}{99} - \frac{1}{100}$

sign：表示当前项的数值符号

term：表示当前项的值

sum：表示当前项的累加和

deno：表示当前项的分母

算法步骤

S1: sign=1

S2: sum=1

S3: deno=2

S4: sign=(-1) * sign

S5: term=sign * (1/deno)

S6: sum=sum+term

S7: deno=deno+1

S8: 若deno ≤ 100返回S4；否则算法结束

3.1.1 几个示例

- 示例3：给出一个大于或等于3的正整数，判断它是不是一个素数

解题思路：素数(prime)是指除了1和该数本身之外，不能被其他任何整数整除的数。

算法步骤

S1: 输入n的值

S2: $i=2$ (i作为除数)

S3: n被i除，得余数r

S4: 如果 $r=0$ ，表示n能被i整除，则输出n“不是素数”，算法结束；否则执行S5

S5: $i+1 \Rightarrow i$

S6: 如果 $i \leq \sqrt{n}$ ，返回S3；否则输出n的值以及“是素数”，然后结束

实际上，n不必被 $2 \sim (n-1)$ 之间的整数除，只须被 $2 \sim n/2$ 间整数除即可，甚至只须被 $2 \sim \sqrt{n}$ 之间的整数除即可。

3.1.1 几个示例

- 示例4：有50个学生，输出成绩在80分以上的学生的学号和成绩

n : 表示学生学号

下标 i : 表示第几个学生

n_1 : 表示第一个学生的学号

n_i : 表示第 i 个学生的学号

g : 表示学生的成绩

g_1 : 表示第一个学生的成绩

g_i : 表示第 i 个学生的成绩

算法步骤

S1: $1 \Rightarrow i$

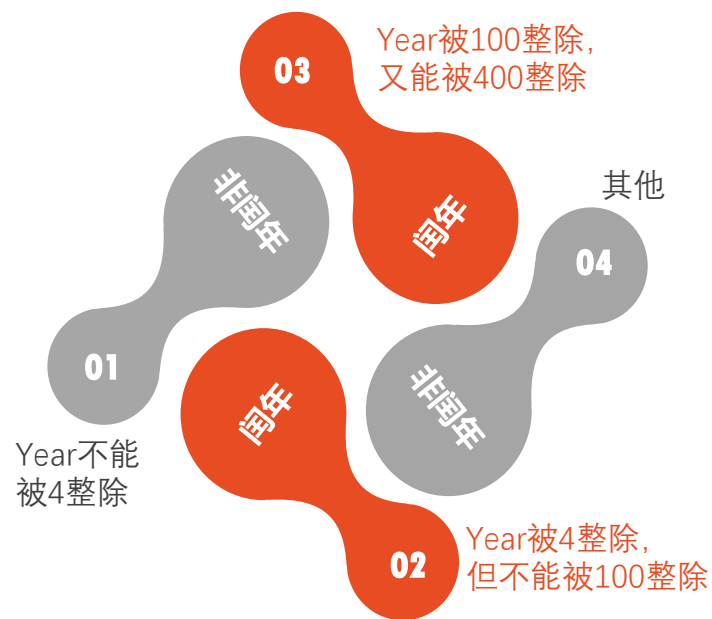
S2: 如果 $g_i \geq 80$, 则输出 n_i 和 g_i , 否则不输出

S3: $i+1 \Rightarrow i$

S4: 如果 $i \leq 50$, 返回到S2, 继续执行, 否则, 算法结束

3.1.1 几个示例

- 示例5：判定2000—2500年中的每一年是否为闰年，并输出结果



算法步骤

S1: 2000=>year

S2: 若year不能被4整除, 则输出year 的值和“不是闰年”。然后转到S6, 检查下一个年份

S3: 若year能被4整除, 不能被100整除, 则输出year的值和“是闰年”。然后转到S6

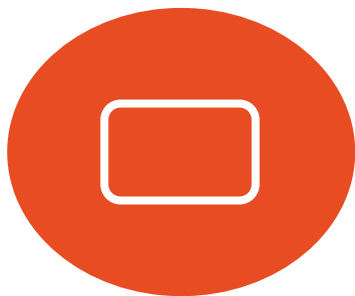
S4: 若year能被400整除, 输出year的值和“是闰年”, 然后转到S6

S5: 输出year的值和“不是闰年”

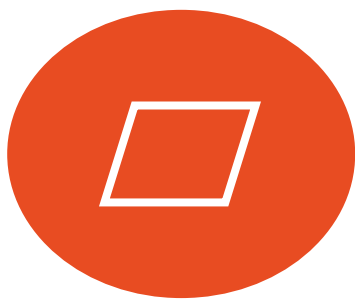
S6: year+1=>year

S7: 当year≤2500时, 转S2继续执行, 否则算法停止

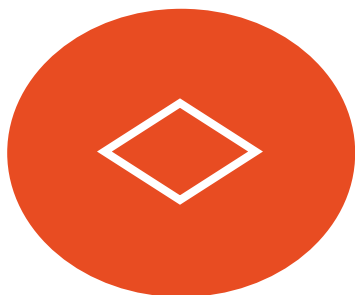
3.1.2 传统流程图



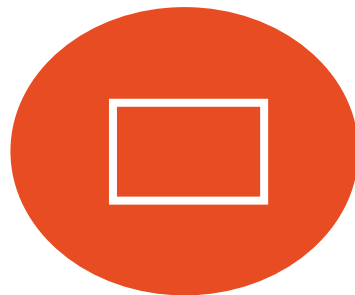
起止框



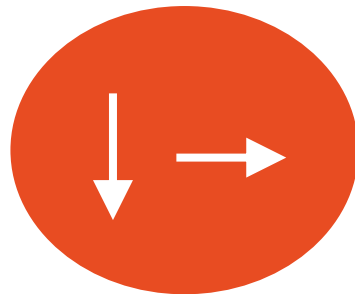
输入输出框



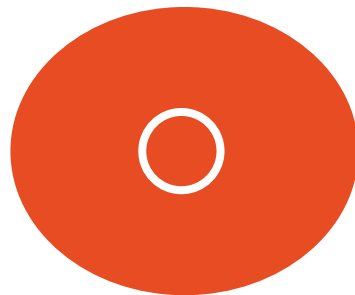
判断框



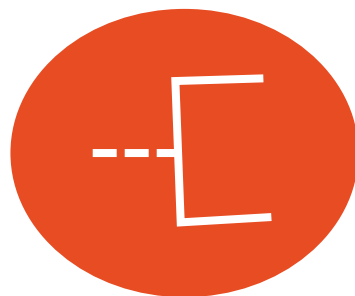
处理框



流程线



连接点



注释框

3.1.2 传统流程图

■ 示例1：求 $1 \times 2 \times 3 \times 4 \times 5$ （用流程图表示）

算法步骤

S1: $1 \Rightarrow p$

S2: $2 \Rightarrow i$

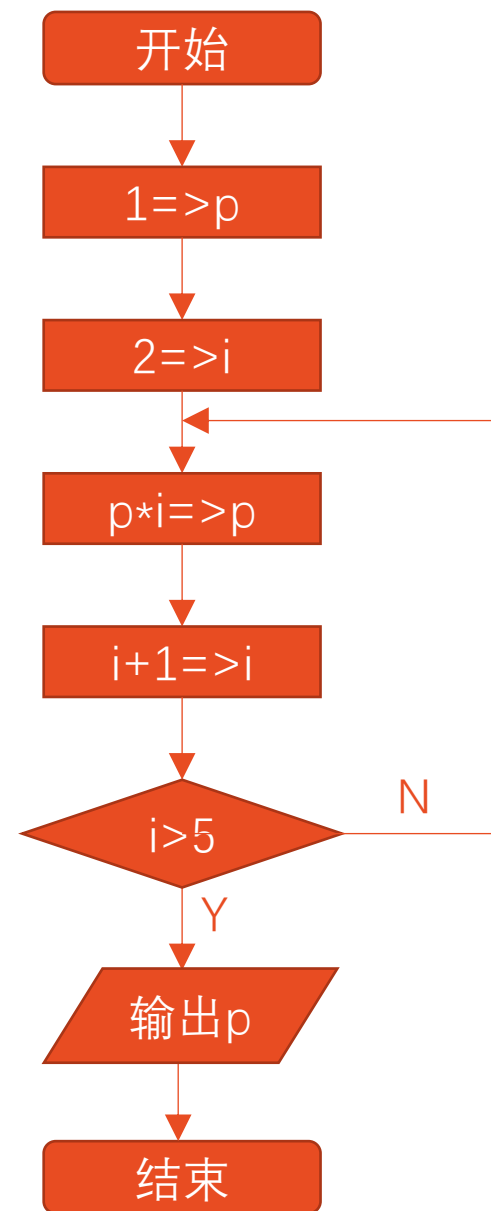
S3: $p * i \Rightarrow p$

S4: $i + 1 \Rightarrow i$

S5: 如果 $i \leq 5$ ，则返回S3；否则结束

P: 表示被乘数

i: 表示乘数



3.1.2 传统流程图

■ 示例2：求 $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots + \frac{1}{99} - \frac{1}{100}$
(用流程图表示)

算法步骤

S1: sign=1

S2: sum=1

S3: deno=2

S4: sign=(-1)*sign

S5: term=sign*(1/deno)

S6: sum=sum+term

S7: deno=deno+1

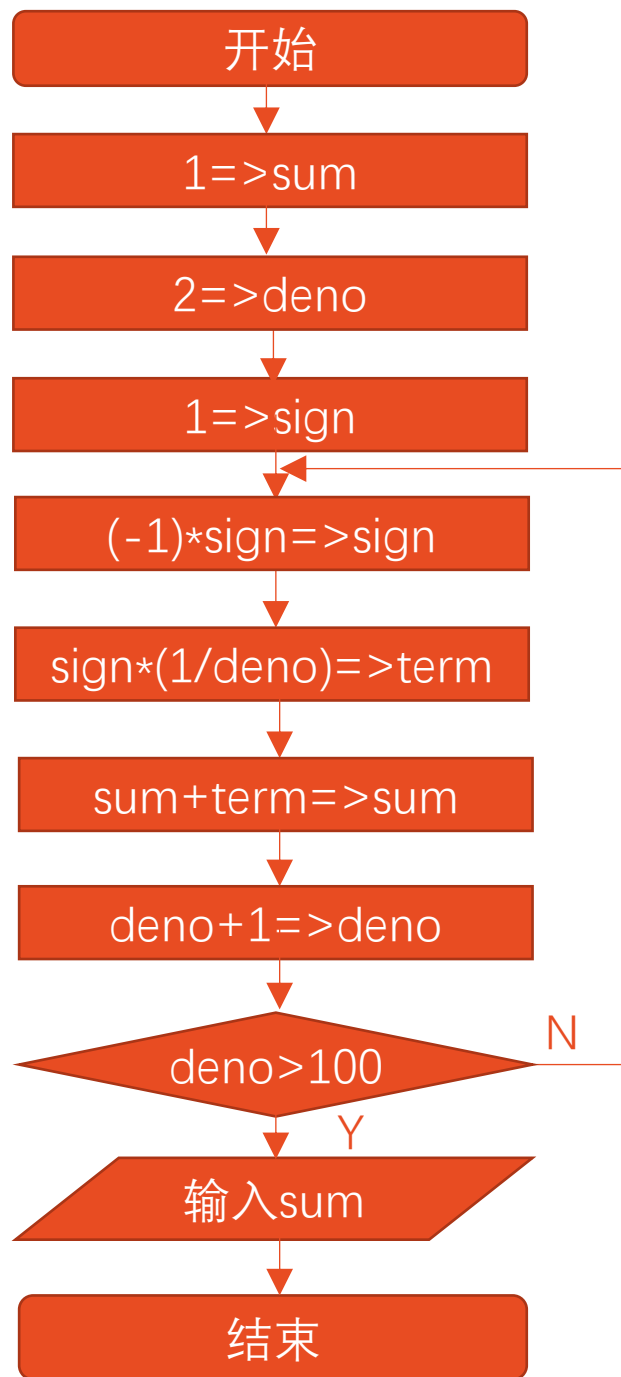
S8: 若deno ≤ 100返回S4；否则算法结束

sign：表示当前项的数值符号

term：表示当前项的值

sum：表示当前项的累加和

deno：表示当前项的分母



3.1.2 传统流程图

■ 示例3：给出一个大于或等于3的正整数，判断它是不是一个素数（用流程图表示）

算法步骤

S1: 输入n的值

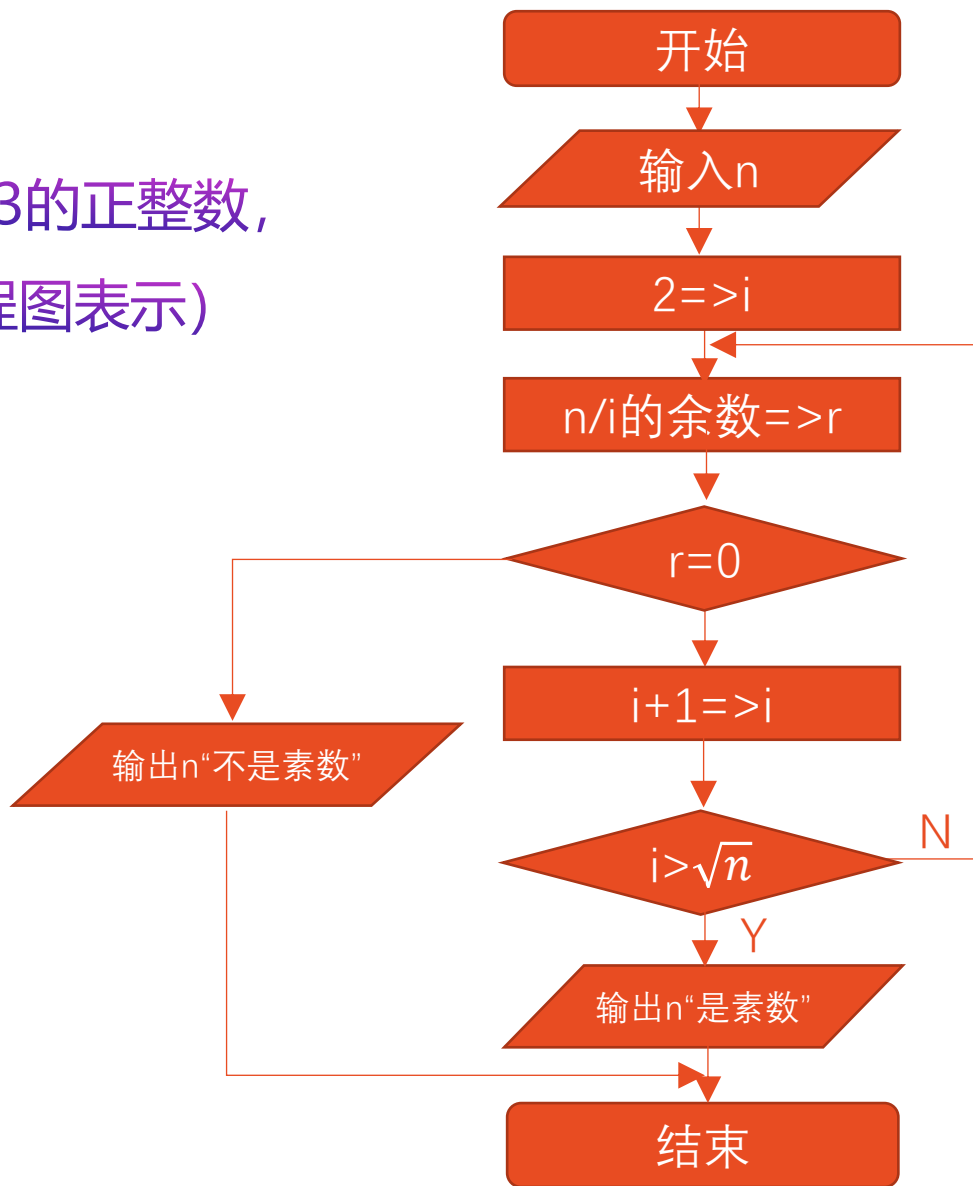
S2: $i=2$ （i作为除数）

S3: n被i除，得余数r

S4: 如果 $r=0$ ，表示n能被i整除，则输出n“不是素数”，算法结束；否则执行S5

S5: $i+1 \Rightarrow i$

S6: 如果 $i \leq \sqrt{n}$ ，返回S3；否则输出n的值以及“是素数”，然后结束



3.1.2 传统流程图

■ 示例4：有50个学生，输出成绩在80分以上的学生的学号和成绩（用流程图表示）

下标 i ：表示第几个学生

n ：表示学生学号

n_1 ：表示第一个学生的学号， n_i ：表示第 i 个学生的学号

g ：表示学生的成绩

g_1 ：表示第一个学生的成绩， g_i ：表示第 i 个学生的成绩

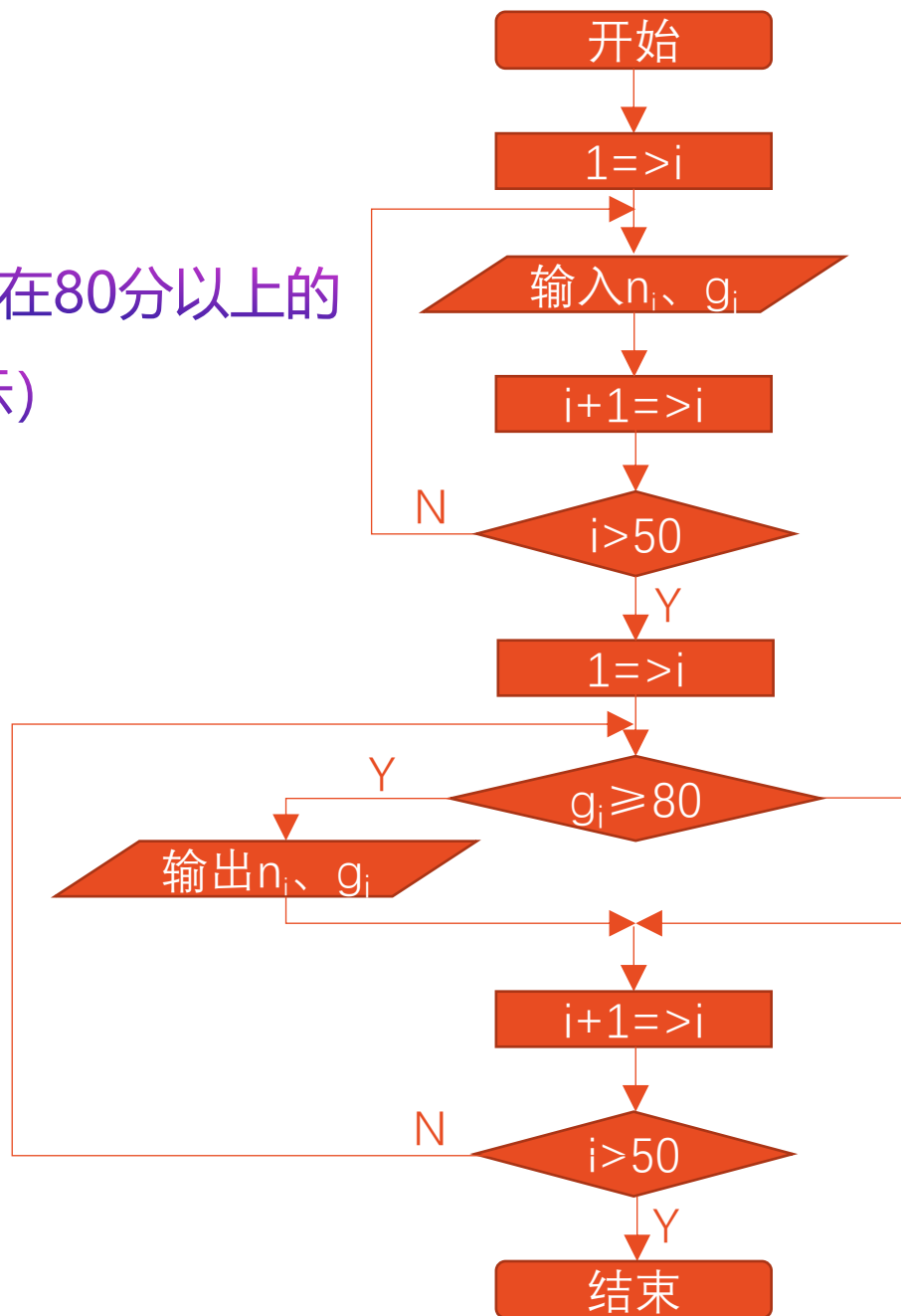
算法步骤

S1: $1 \Rightarrow i$

S2: 如果 $g_i \geq 80$ ，则输出 n_i 和 g_i ，否则不输出

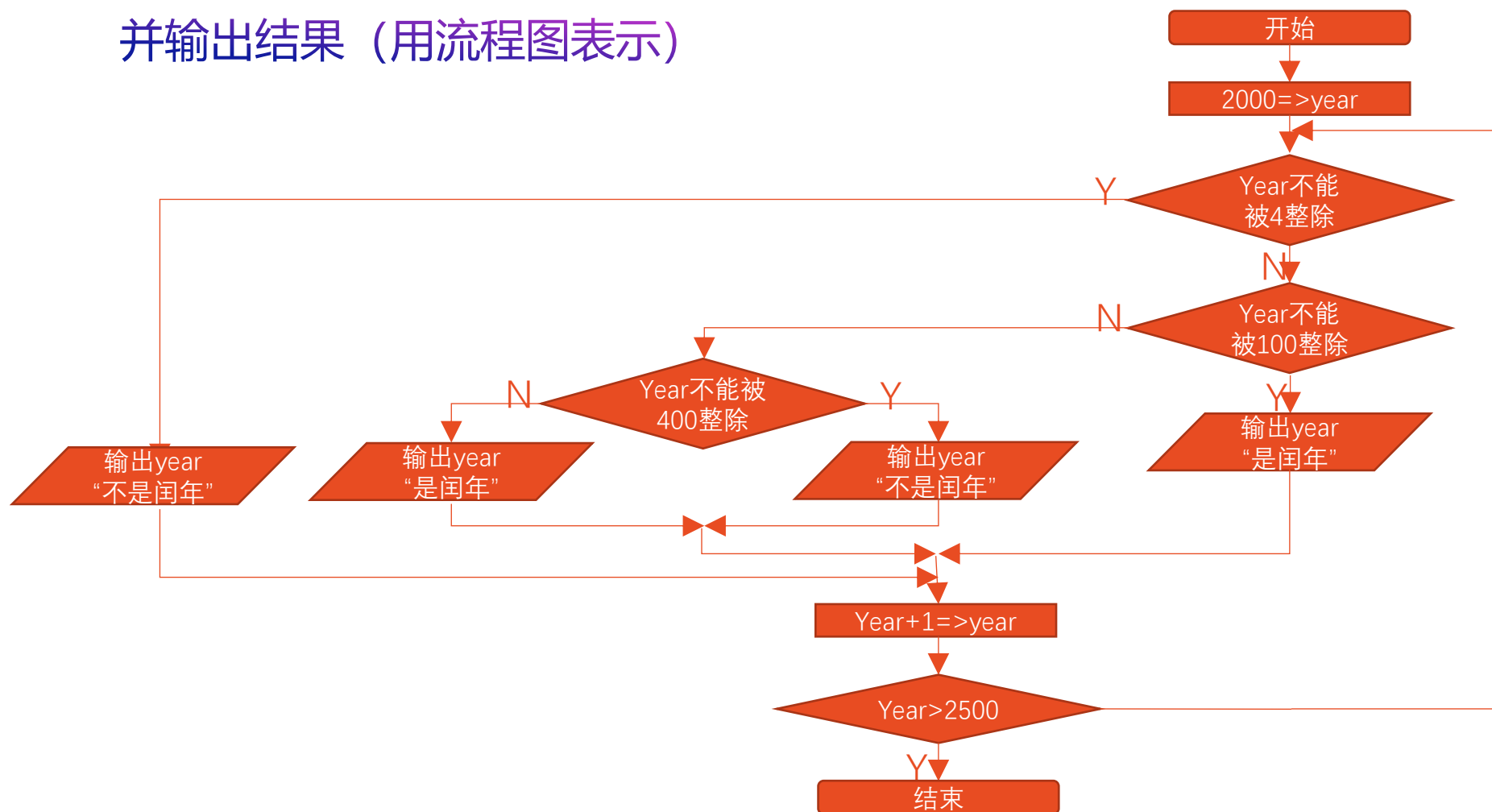
S3: $i+1 \Rightarrow i$

S4: 如果 $i \leq 50$ ，返回到S2，继续执行，否则，算法结束



3.1.2 传统流程图

- 示例5：判定2000—2500年中的每一年是否为闰年，并输出结果（用流程图表示）



3.1.3 结构化流程图

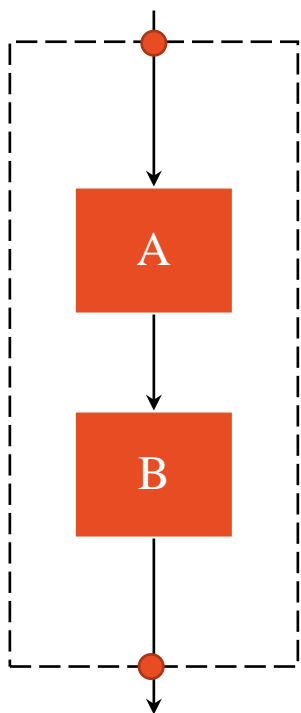
■ 传统流程图的弊端



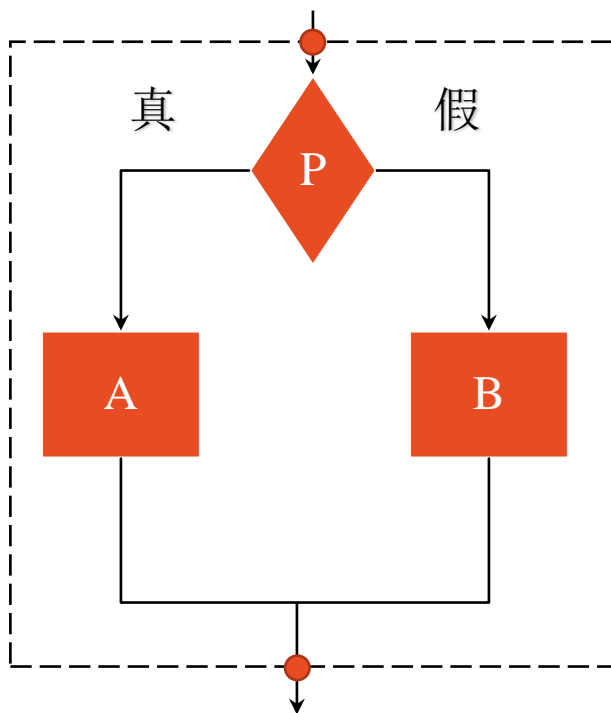
传统流程图用流程线指出各框的执行顺序，对流程线的使用没有严格限制。因此，使用者可不受限制地使流程随意地转来转去，使流程图变得毫无规律，阅读时要花很大精力去追踪流程，使人难以理解算法的逻辑。

3.1.3 结构化流程图

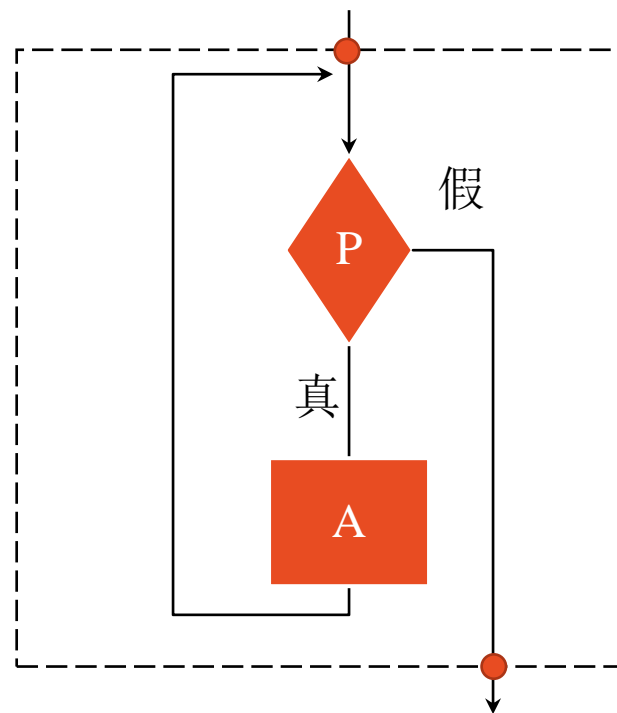
■ 三种基本结构



顺序结构



选择结构



循环结构

3.1.3 结构化流程图

■ 三种基本结构的特点



只有一个入口



只有一个出口



结构内的每一部分都有机会被执行到

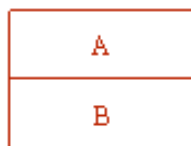


结构内不存在“死循环”

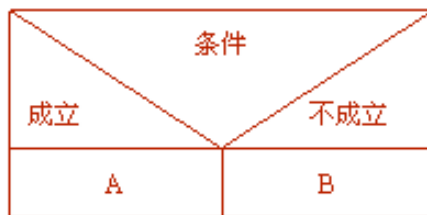
3.1.3 结构化流程图

■ N-S流程图表示法

- 完全去除了带箭头的流程线
- 全部流程写在一个矩形框内



(a) 顺序结构



(b) 分支结构



(c) 循环结构 (条件在前)



(d) 循环结构 (条件在后)

3.1.4 伪代码

- 伪代码：用介于自然语言和计算机语言之间的文字和符号，来描述程序流程
- 它如同一篇文章一样，自上而下地写下来。每一行(或几行)表示一个基本操作。
- 它不用图形符号，因此书写方便，格式紧凑，修改方便，容易看懂，也便于向计算机语言算法(即程序)过渡

3.1.4 伪代码

■ 示例1： $1 \times 2 \times 3 \times 4 \times 5$ (用伪代码表示)

P: 表示被乘数

i: 表示乘数

```
#include <stdio.h>
int main()
{
    int i,p;
    p=1;
    i=2;
    while(i<=5) {
        p=p*i;
        i=i+1;
    }
    printf("%d\n",p);
    return 0;
}
```

C 代 码

begin (算法开始)

1=>p

2=>i

while i≤5

{

p*i=>p

i+1=>i

}

print p

end (算法结束)

伪 代 码

3.1.4 伪代码

- 示例2：求 $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots + \frac{1}{99} - \frac{1}{100}$
(用流程图表示)

```
#include <stdio.h>
int main()
{
    int sign=1;
    double deno=2.0,sum=1.0,term;
    while(deno<=100)
    {
        sign=-sign;
        term=sign/deno;
        sum=sum+term;
        deno=deno+1;
    }
    printf("%f\n",sum);
    return 0;
}
```

C 代 码

```
begin      (算法开始)

1=>sign
1=>sum
2=>deno
while deno ≤ 100
{
    (-1)*sign=>sign
    sign*(1/deno)=>term
    sum+term=>sum
    deno+1=>deno
}
print sum
end        (算法结束)
```

伪 代 码

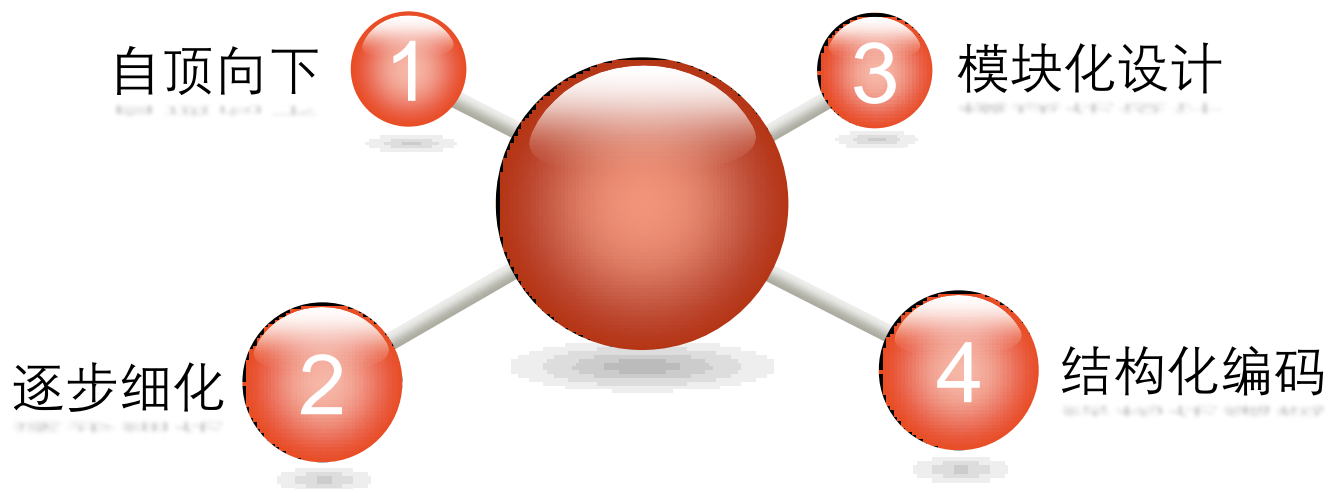
sign: 表示当前项的数值符号

term: 表示当前项的值

sum: 表示当前项的累加和

deno: 表示当前项的分母

结构化程序设计方法



思考

用流程图和伪代码的方法，
表示博饼游戏的积分累计流程

C语言运算符概览

算术运算符:	+ - * / % ++ --
赋值运算符:	= 及其扩展
求字节数 :	sizeof
强制类型转换:	(类型)
函数调用符运算符:	()
关系运算符:	< <= == > >= !=
逻辑运算符:	! &&
条件运算符:	?:
下标运算符:	[]
位运算符 :	<< >> ~ & ^
指针运算符:	* &
分量运算符:	. ->
逗号运算符:	,

3.2 关系运算/逻辑运算/条件运算

- 关系运算：将两个数值进行比较，判断其比较的结果是否符合给定的条件

- 关系表达式只能表达一些简单的条件
- 每个判断只是对一个条件进行测试

- 逻辑运算：通过逻辑运算符把简单的条件组合起来，能够形成更加复杂的条件

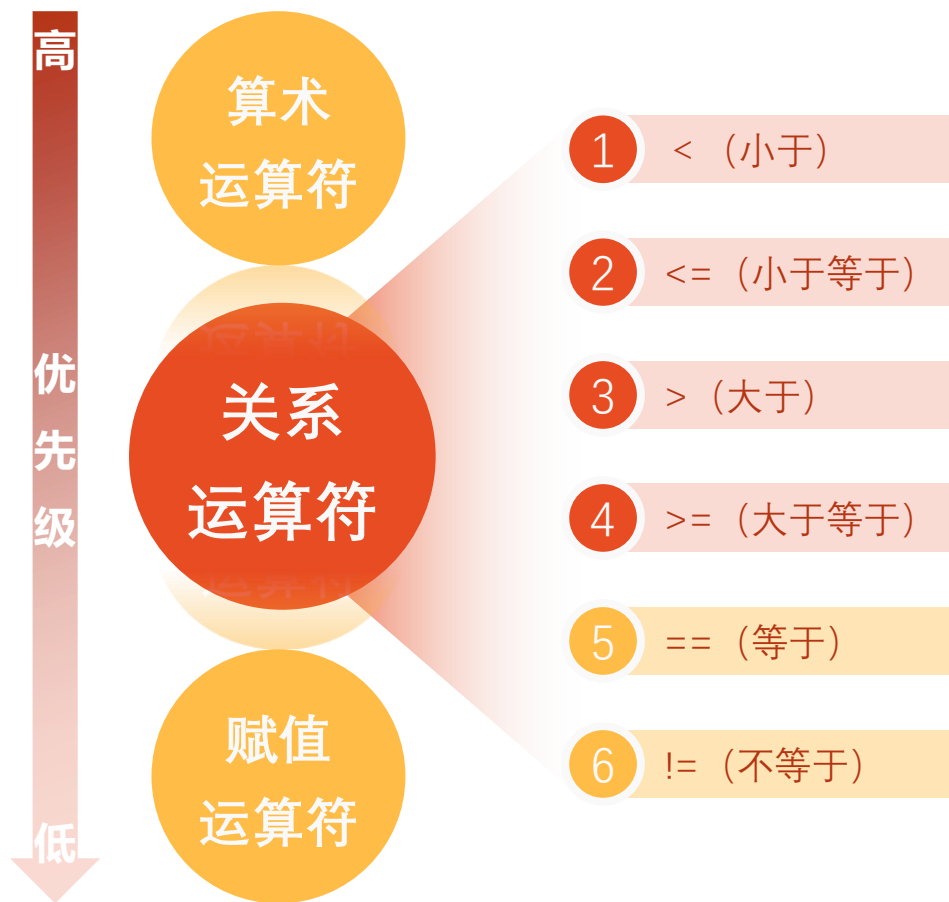
- 例： $10 > y > 5$ 的逻辑表达式 $(y > 5) \ \&\& \ (y < 10)$

- 例： $x < -10$ 或者 $x > 0$ 的逻辑表达式 $(x < -10) \ || \ (x > 0)$

- 条件运算：C语言中唯一的一个三目运行符，根据不同的结果来决定计算哪个子表达式

3.2.1 关系运算符与关系表达式

■ 运算符优先级



- 前 4 种关系运算符的优先级别相同，后 2 种也相同。前 4 种高于后 2 种。
- 关系运算符的优先级低于算术运算符。
- 关系运算符的优先级高于赋值运算符。

$c > a + b$ 等效于 $c > (a + b)$

(关系运算符的优先级低于算术运算符)

$a > b == c$ 等效于 $(a > b) == c$

(大于运算符 > 的优先级高于相等运算符 ==)

$a == b < c$ 等效于 $a == (b < c)$

(小于运算符 < 的优先级高于相等运算符 ==)

$a = b > c$ 等效于 $a = (b > c)$

(关系运算符的优先级高于赋值运算符)

3.2.1 关系运算符与关系表达式

- 用关系运算符将两个数值或数值表达式连接起来的式子，称为关系表达式
- 关系表达式的值是一个逻辑值，即“真”或“假”
- 在C的逻辑运算中，以“1”代表“真”，以“0”代表“假”

若 $a=3$, $b=2$, $c=1$, 则:

$d=a>b$, 由于 $a>b$ 为真, 因此关系表达式 $a>b$ 的值为1, 所以赋值后 d 的值为1。

$f=a>b>c$, 则 f 的值为0。因为“ $>$ ”运算符是自左至右的结合方向, 先执行“ $a>b$ ”得值为1, 再执行关系运算“ $1>c$ ”, 得值0, 赋给 f , 所以 f 的值为0

3.2.2 逻辑运算符与逻辑表达式

■ 运算符优先级

运算符	含义	举例	说明
!	逻辑非(NOT)	!a	如果a为假, 则!a为真;如果a为真, 则!a为假
&&	逻辑与(AND)	a && b	如果a和b都为真, 则结果为真, 否则为假
	逻辑或(OR)	a b	如果a和b有一个以上为真, 则结果为真, 二者都为假时, 结果为假

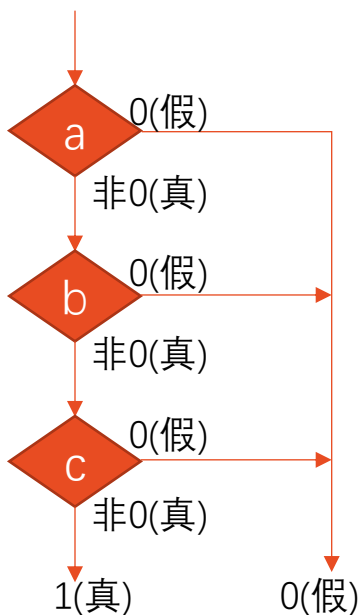
- “&&”和“||”是双目运算符, 要求有两个运算对象(操作数); “!”是单目运算符, 只要有一个运算对象
- 优先次序: !(非)→&&(与)→||(或), 即“!”为三者中最高的; 逻辑运算符中的“&&”和“||”低于关系运算符, “!”高于算术运算符
- 逻辑运算结果不是0就是1, 不可能是其他数值。而在逻辑表达式中作为参加逻辑运算的运算对象可以是0(“假”)或任何非0的数值(按“真”对待)

a	b	!a	!b	a && b	a b
真 (非0)	真 (非0)	假 (0)	假 (0)	真 (1)	真 (1)
真 (非0)	假 (0)	假 (0)	真 (1)	假 (0)	真 (1)
假 (0)	真 (非0)	真 (1)	假 (0)	假 (0)	真 (1)
假 (0)	假 (0)	真 (1)	真 (1)	假 (0)	假 (0)

3.2.2 逻辑运算符与逻辑表达式

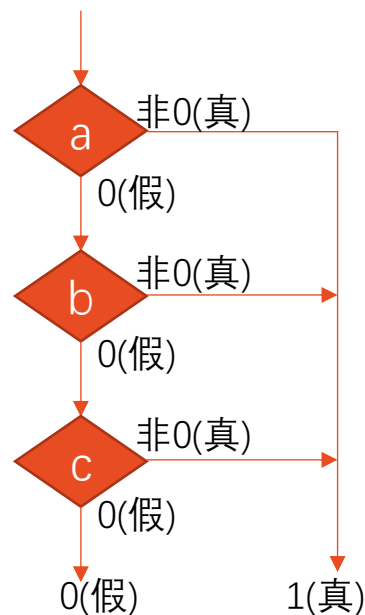
■ 在逻辑表达式的求解中，并不是所有的逻辑运算符都被执行，只是在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符。

- $a \ \&\& \ b \ \&\& \ c$ 。只有a为真(非0)时，才需要判别b的值。只有当a和b都为真时才需要判别c的值。



逻辑
表达式

- $a \ || \ b \ || \ c$ 。只要a为真(非0)，就不必判断b和c。只有a为假，才判别b。a和b都为假才判别c。



3.2.2 逻辑运算符与逻辑表达式

■ 示例：下列代码在特定输入下的输出

```
1. int main()
2. {
3.     int a = 0, b = 1, c = 0;
4.     int x, y, z;
5.     scanf("%d%d%d", &a, &b, &c);

6.     x = a; y = b; z = c;
7.     if (!x && y++) z--;
8.     printf("x=%d y=%d z=%d\n", x, y, z);

9.     x = a; y = b; z = c;
10.    if (!x && ++y) z--;
11.    printf("x=%d y=%d z=%d\n", x, y, z);

12.    x = a; y = b; z = c;
13.    if (!x || y++) z--;
14.    printf("x=%d y=%d z=%d\n", x, y, z);

15.    x = a; y = b; z = c;
16.    if (!x || ++y) z--;
17.    printf("x=%d y=%d z=%d\n", x, y, z);
18.}
```

输入1:

0 0 0

输出1:

x=0 y=1 z=0

x=0 y=1 z=-1

x=0 y=0 z=-1

x=0 y=0 z=-1

输入2:

0 1 0

输出2:

x=0 y=2 z=-1

x=0 y=2 z=-1

x=0 y=1 z=-1

x=0 y=1 z=-1

输入3:

1 0 0

输出3:

x=1 y=0 z=0

x=1 y=0 z=0

x=1 y=1 z=0

x=1 y=1 z=-1

输入4:

1 1 0

输出3:

x=1 y=1 z=0

x=1 y=1 z=0

x=1 y=2 z=-1

x=1 y=2 z=-1

程序控制语句

■ 选择语句：

- if () ... else ...： 两个分支
- switch ()： 多个分支

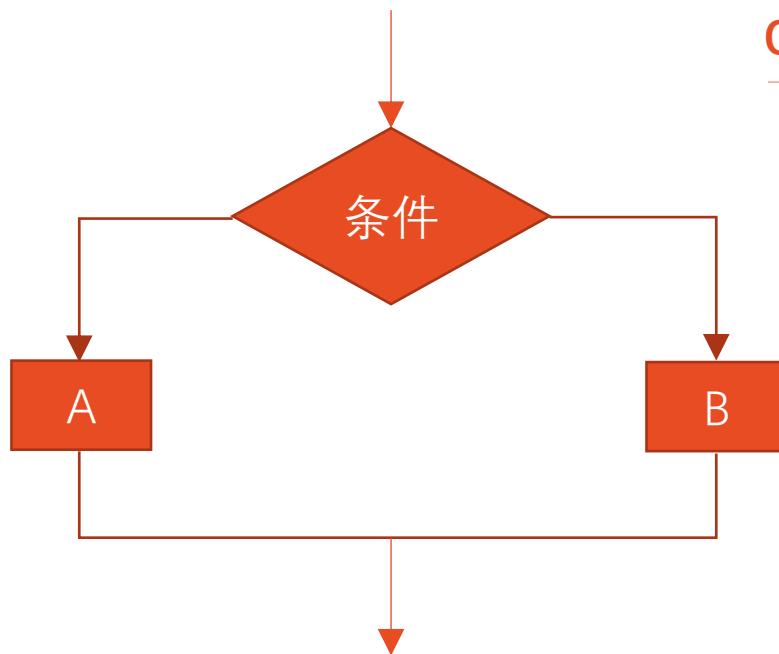
■ 循环语句：

- for () ...
- while () ...
- do ... while ()

■ 改变执行流程语句：

- 结束本次循环语句： continue
- 终止执行switch或循环语句： break
- 转向语句： goto（慎用）
- 从函数返回语句： return

3.3 选择语句



C语言有两种选择语句

- **if**语句，用来实现两个分支的选择结构
- **switch**语句，用来实现多分支的选择结构

3.3.1 if语句

■ 示例：输入两个实数，按由小到大的顺序输出这两个数

解题思路：只要做一次比较，然后进行一次交换即可。用if语句实现条件判断。

```
1. #include <stdio.h>
2. int main()
3. {
4.     float a,b,t;
5.     scanf("%f,%f",&a,&b);
6.     if(a>b)
7.     {                               //将a和b的值互换
8.         t=a;
9.         a=b;
10.        b=t;
11.    }
12.    printf("%5.2f,%5.2f\n",a,b);
13.    return 0;
14.}
```



两个变量值的互换

a=b; //把变量b的值赋给变量a，a的值等于b的值
b=a; //再把变量a的值赋给变量b，变量b值没有改变

因此，为了实现互换，必须借助于第三个变量

3.3.1 if语句

■ if语句的一般形式

```
if (表达式) 语句1  
[ else 语句2 ]
```

“表达式”可以是关系表达式、逻辑表达式，甚至数值表达式

方括号内的部分(即else子句)为可选的，既可以有，也可没有

语句1和语句2可以是一个简单的语句，也可以是一个复合语句，还可以是另一个if语句

形式1 没有else子句部分

```
if(表达式) 语句1
```

形式2 有else子句部分

```
if (表达式) 语句1  
else 语句2
```

形式3 在else部分又嵌套了多层的if语句

```
if(表达式1)      语句1  
else if(表达式2)  语句2  
else if(表达式3)  语句3  
      ⋮           ⋮  
else if(表达式m)  语句m  
else              语句m+1
```

3.3.1 if语句

■ if语句的嵌套

注意

if与else的配对关系：else总是与它上面的最近的未配对的if配对。

```
if()  
    if() 语句1  
else  
    if() 语句2
```

程序员把else写在与第1个if(外层if)同一列上，意图是使else与第1个if对应，但实际上else是与第2个if配对，因为它们相距最近。

```
if()  
    if() 语句1  
else  
    if() 语句2
```

你以为的

```
if()  
    if() 语句1  
else  
        if() 语句2
```

实际上的

如果if与else的数目不一样，为实现程序设计者的思想，可以加花括号来确定配对关系。

```
if()  
{  
    if() 语句1  
}  
else if() 语句2
```

内嵌if

```
if()  
    if() 语句1  
    else 语句2  
else  
    if() 语句3  
    else 语句4
```

内嵌if

内嵌if

3.3.1 if语句

■ 示例：写一程序，判断某一年是否为闰年

<div> <div>真</div> <div>假</div> </div>		year被4整除	
		year被100整除	
真	假	假	
		year被400整除	
真	假	leap=1	leap=0
leap=1	leap=0		
真		leap	
输出“闰年”		输出“非闰年”	

```
1. #include <stdio.h>
2. int main() {
3.     int year, leap;
4.     printf("enter year:");
5.     scanf("%d", &year);
```

```
6.    if(year%4==0) {
7.        if(year%100==0) {
8.            if(year%400==0)
9.                leap=1;
10.           else
11.               leap=0;
12.        }
```

```
13.         else
14.             leap=1;
15.     } else
16.         leap=0;
17.     if(leap) printf("%d is ",year);
18.     else printf("%d is not ",year);
19.     printf("a leap year.\n");
20.     return 0;
21. }
```

```
1. if (year%4!=0)
2.     leap=0;
3. else if (year%100!=0)
4.     leap=1;
5. else if (year%400!=0)
6.     leap=0;
7. else
8.     leap=1;
```

简化

```
(year%4==0 && year%100!=0) || (year%400==0)
```

```
1. if ( ? ? ? )
2.     leap=1;
3. else
4.     leap=0;
```

进一步简化

3.3.2 switch语句

■ 示例：要求按照考试成绩的等级输出百分制分数段，A等为90分以上，B等为80～89分，C等为70～79分，D等为60～69分，E等为60分以下。成绩的等级由键盘输入。

```
1. #include <stdio.h>
2. int main() {
3.     char grade;
4.     scanf("%c",&grade);
5.     printf("Your score:");
6.     switch(grade) {
7.         case 'A': printf("90~100\n");break;
8.         case 'B': printf("80~89\n");break;
9.         case 'C': printf("70~79\n");break;
10.        case 'D': printf("70~79\n");break;
11.        case 'E': printf("<60\n");break;
12.        default:  printf("enter data error!\n");
13.    }
14.    return 0;
15.}
```



等级grade定义为字符变量，从键盘输入一个大写字母，赋给变量grade，switch得到grade的值并把它和各case中给定的值('A','B','C','D'之一)相比较，如果和其中之一相同(称为匹配)，则执行该case后面的语句(即printf语句)。如果输入的字符与'A','B','C','D'都不相同，就执行default后面的语句，**注意在每个case后面后的语句中，最后都有一个break语句，它的作用是使流程转到switch语句的末尾(即右花括号处)。**

3.3.2 switch语句

思考：对于右侧代码，
输入为'A', 'B', 'C', 'D', 'E'
时，输出分别是什么？

```
1. #include <stdio.h>
2. int main() {
3.     char grade;
4.     scanf("%c",&grade); printf("Your score:");
5.     switch(grade) {
6.         case 'A': { printf("90~100\n");break; }
7.         case 'B': printf("80~89\n");
8.         case 'C': printf("70~79\n");break;
9.         case 'D': case 'E': printf("<60\n");break;
10.        default:  printf("enter data error!\n");
11.    }
12.}
```

switch(表达式)

```
{
    case 常量1 : 语句1
    case 常量2 : 语句2
        :
        :
    case 常量n : 语句n
    default :      语句n+1
}
```

- (1) 括号内的“表达式”，其值的类型应为整数类型(包括字符型)。
- (2) 花括号内是一个复合语句，包含多个case开头的语句行和最多一个default开头的语句行。case后面跟一个常量(或常量表达式)，它们和default都是起标号作用，用来标志一个位置。
- (3) 执行switch语句时，先计算switch后面的“表达式”的值，然后将它与各case标号比较，若与某一个case标号中的常量相同，控制流就转到匹配的case标号后面的语句。若没有任何匹配，控制流转去执行default标号后面的语句；若没有default标号，则不执行任何语句。
- (4) 在case子句中虽然包含了一个以上执行语句，但可以不必用花括号括起来，会自动顺序执行本case标号后面所有的语句。当然加上花括号也可以。
- (5) 各个case标号出现次序不影响执行结果，但每一个case常量必须互不相同；否则就会出现互相矛盾的现象
- (6) case标号只起标记的作用。在执行switch语句时，根据switch表达式的值找到匹配的入口标号，在执行完一个case标号后面的语句后，就从此标号开始执行下去，不再进行判断。因此，一般情况下，在执行一个case子句后，应当用break语句使流程跳出switch结构。
- (7) 多个case标号可以共用一组执行语句。

3.4 循环语句

■ 为什么需要循环语句？

- 要向计算机输入全班50个学生的成绩；
(重复50次相同的输入操作)
- 分别统计全班50个学生的平均成绩；
(重复50次相同的计算操作)

解决
方法

```
scanf("%f,%f,%f,%f,%f",&score1,&score2,&score3,&score4,&score5);  
//输入一个学生5门课的成绩  
aver=(score1+score2+score3+score4+score5)/5;  
//求该学生平均成绩  
printf("aver=%7.2f",aver);  
//输出该学生平均成绩
```

重复写50个同样的程序段？
如果是n个学生的情况怎么办？



```
i=1;           //设整型变量i初值为1  
while(i<=50)  //当i的值小于或等于50时执行花括号内的语句  
{  
    scanf("%f,%f,%f,%f,%f",&score1,&score2,&score3,&score4,&score5);  
    aver=(score1+score2+score3+score4+score5)/5;  
    printf("aver=%7.2f",aver);  
    i++;       //每执行完一次循环使i的值加1  
}
```

3.4.1 用while语句实现循环

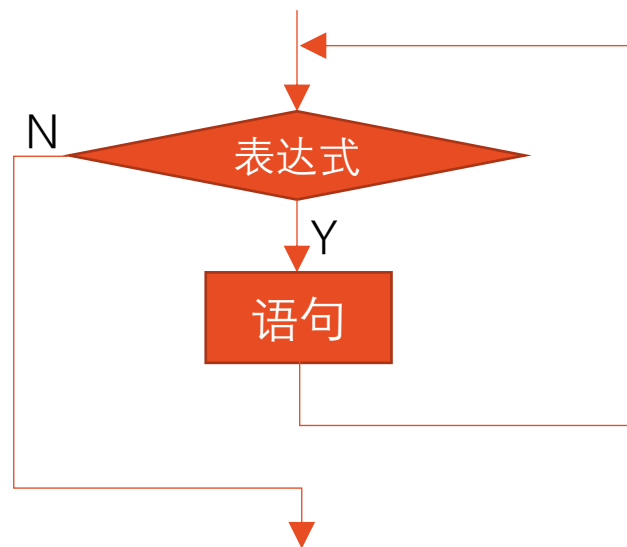
while(表达式) 语句

while语句可简单地记为：只要当循环条件表达式为真(即给定的条件成立)，就执行循环体语句。

“语句”就是循环体。循环体可以是一个简单的语句，可以是复合语句(用花括号括起来的若干语句)。

执行循环体的次数是由循环条件控制的，这个循环条件就是上面一般形式中的“表达式”，它也称为循环条件表达式。

当此表达式的值为“真” (以非0值表示)时，就执行循环体语句；为“假” (以0表示)时，就不执行循环体语句。



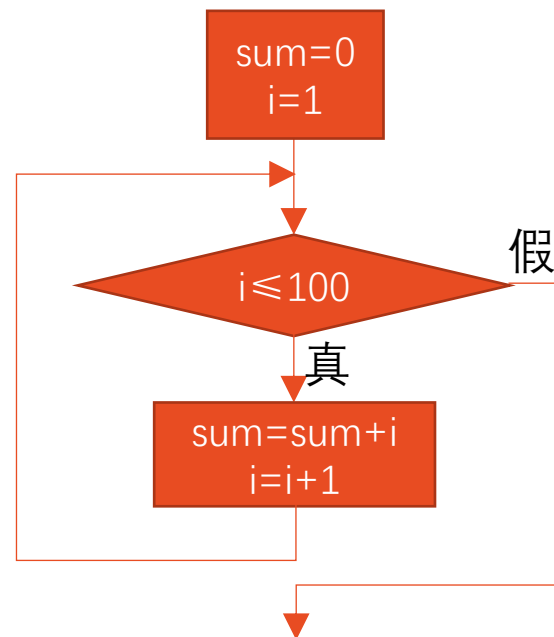
注意

while循环的特点是先判断条件表达式，后执行循环体语句。

3.4.1 用while语句实现循环

■ 示例：求 $1+2+3+\dots+100$ ，即 $\sum_{n=1}^{100} n$

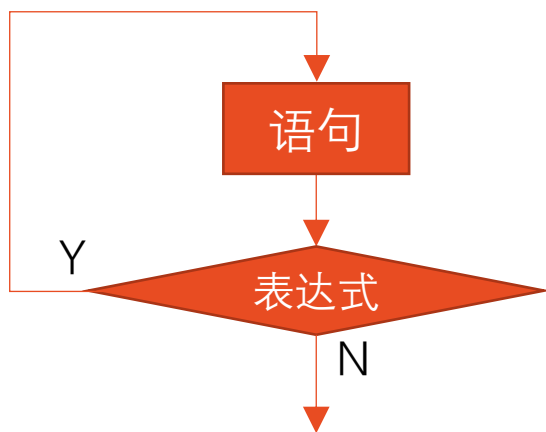
```
1. #include<stdio.h>
2. int main()
3. {
4.     int i=1,sum=0; //定义变量i的初值为1,sum的初值为0
5.     while(i<=100) //当i>100, 条件表达式i<=100的值为假, 不执行循环体
6.     { //循环体开始
7.         sum=sum+i; //第1次累加后, sum的值为1
8.         i++; //加完后, i的值加1, 为下次累加做准备
9.     } //循环体结束
10.    printf("sum=%d\n",sum); //输出1+2+3...+100的累加和
11.    return 0;
12.}
```



- (1) 循环体如果包含一个以上的语句，应该用花括号括起来，作为复合语句出现。
- (2) 不要忽略给`i`和`sum`赋初值，否则它们的值是不可预测的，结果显然不正确。
- (3) 在循环体中应有使循环趋向于结束的语句。如本例中的“`i++`；”语句。如果无此语句，则`i`的值始终不变，循环永远不结束。

3.4.2 用do...while语句实现循环

do
语句
while(表达式);



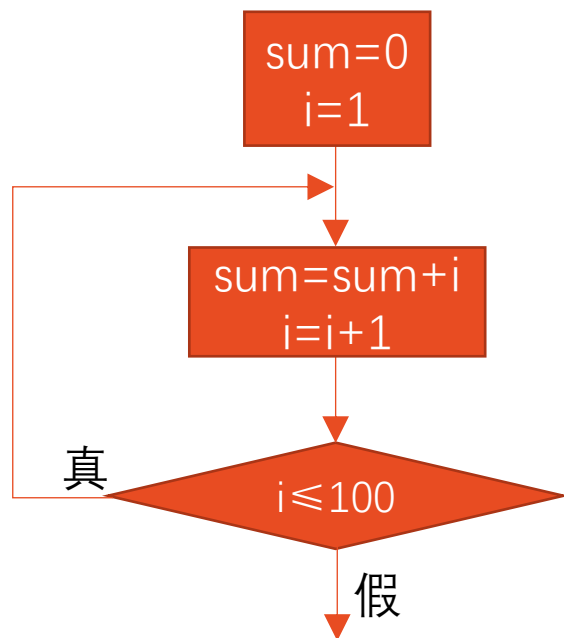
注意

do...while语句的特点是，先无条件地执行循环体，然后判断循环条件是否成立。

3.4.2 用do...while语句实现循环

■ 示例：求 $1+2+3+\dots+100$ ，即 $\sum_{n=1}^{100} n$

```
1. #include <stdio.h>
2. int main()
3. {
4.     int i=1, sum=0;
5.     do
6.     {
7.         sum=sum+i;
8.         i++;
9.     }while(i<=100);
10.    printf("sum=%d\n", sum);
11.    return 0;
12. }
```



在一般情况下，用while语句和用do...while语句处理同一问题时，若二者的循环体部分是一样的，那么结果也一样。

但是如果while后面的表达式一开始就为假(0值)时，两种循环的结果是不同的。

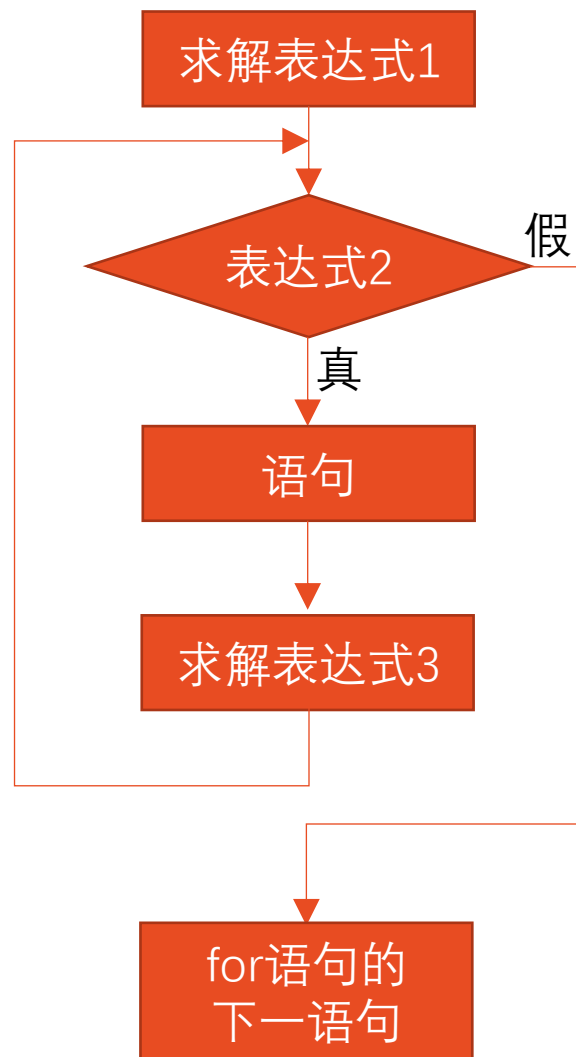
3.4.3 用for语句实现循环

**for(表达式1; 表达式2; 表达式3)
语句**

表达式1: 设置初始条件，只执行一次。可以为零个、一个或多个变量设置初值。

表达式2: 是循环条件表达式，用来判定是否继续循环。在每次执行循环体前先执行此表达式，决定是否继续执行循环。

表达式3: 作为循环的调整，例如使循环变量增值，它是在执行完循环体后才进行的。



3.4.3 用for语句实现循环

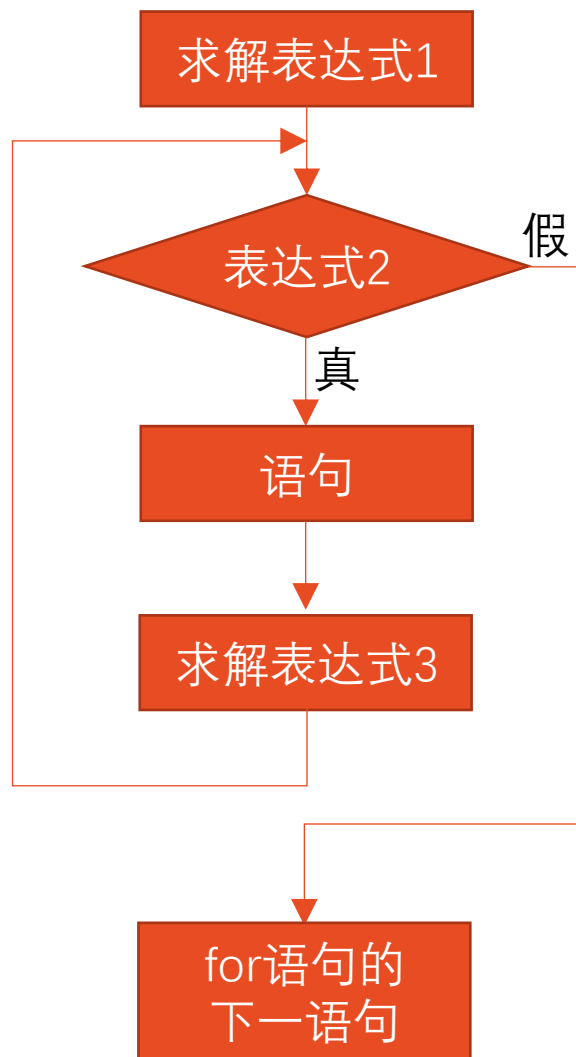
for(表达式1; 表达式2; 表达式3)
语句

≡

```
表达式1;  
while 表达式2  
{  
    语句  
    表达式3  
}
```

for语句的执行过程如下:

- (1) 求解表达式1。
 - (2) 求解表达式2, 若此条件表达式的值为真(非0), 则执行for语句中的循环体, 然后执行第(3)步。若为假(0), 则结束循环, 转到第(5)步。
 - (3) 求解表达式3。
 - (4) 转回步骤(2)继续执行。
- 注意: 在执行完循环体后, 循环变量的值“超过”循环终值, 循环结束。
- (5) 循环结束, 执行for语句下面的一个语句。

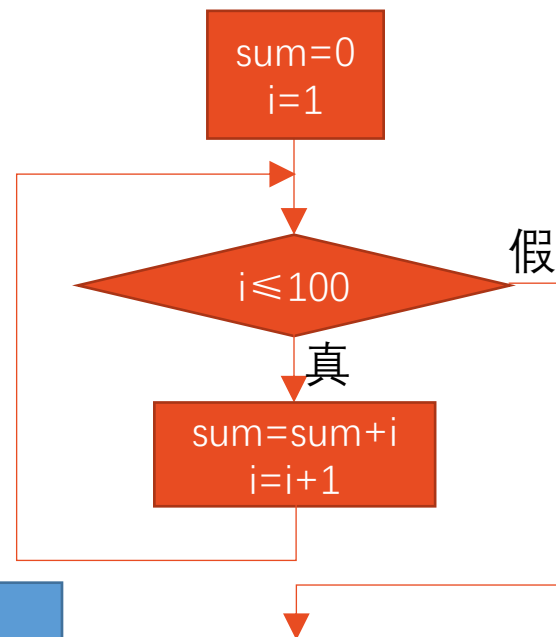


3.4.3 用for语句实现循环

■ 示例：求 $1+2+3+\dots+100$ ，即 $\sum_{n=1}^{100} n$

```
1. #include <stdio.h>
2. int main()
3. {
4.     int i, sum=0;
5.     for (i = 1; i <= 100; i++)
6.     {
7.         sum=sum+i;
8.     }
9.     printf("sum=%d\n", sum);
10.    return 0;
11. }
12. }
```

可简写为for (i = 1, sum = 0; i <= 100; sum+=i, i++);



3.4.3 用for语句实现循环

for(表达式1; 表达式2; 表达式3)
语句

注意

- “表达式1”可以省略，即不设置初值，但表达式1后的分号不能省略。例如: for(; i<=100;i++)。应当注意: 由于省略了表达式1，没有对循环变量赋初值，因此，为了能正常执行循环，应在for语句之前给循环变量赋以初值。
- 表达式2也可以省略，即不用表达式2来作为循环条件表达式，不设置和检查循环的条件。此时循环无终止地进行下去,也就是认为表达式2始终为真。
- 表达式3也可以省略，但此时程序设计者应另外设法保证循环能正常结束。
- 甚至可以将3个表达式都可省略，即不设初值，不判断条件(认为表达式2为真值)，循环变量也不增值，无终止地执行循环体语句，显然这是没有实用价值的。
- 表达式1可以是设置循环变量初值的赋值表达式，也可以是与循环变量无关的其他表达式。表达式3也可以是与循环控制无关的任意表达式。但不论怎样写for语句，都必须使循环能正常执行。
- 表达式1和表达式3可以是一个简单的表达式，也可以是逗号表达式，即包含一个以上的简单表达式，中间用逗号间隔。
- 表达式2一般是关系表达式或逻辑表达式，但也可以是数值表达式或字符表达式，只要其值为非零，就执行循环体。
- for语句的循环体可为空语句，把本来要在循环体内处理的内容放在表达式3中，作用是一样的。可见for语句功能强，可以在表达式中完成本来应在循环体内完成的操作。
- C 99允许在for语句的“表达式1”中定义变量并赋初值。

3.4.4 几种循环的比较

- 3种循环都可以用来处理同一问题，一般情况下可以互相代替
- while循环、do...while循环和for循环都可以用break语句跳出循环，用continue语句结束本次循环
- 在while循环和do...while循环中，只在while后面的括号内指定循环条件，因此为了使循环能正常结束，应在循环体中包含使循环趋于结束的语句(如i++，或i=i+1等)；此外，循环变量初始化的操作应在while和do...while语句之前完成
- for循环可在表达式1中完成循环变量初始化，在表达式3中包含使循环趋于结束的操作，甚至可以将循环体中的操作全部放到表达式3中。因此for语句的功能更方便，凡用while循环能完成的，用for循环都能实现

3.4.5 循环的嵌套

01

```
while()  
{  
    :  
    while()  
    {...}  
}
```

} 内层
循环

02

```
do  
{  
    :  
    do  
    {...}  
    while();  
}while();
```

} 内层
循环

03

```
for(;;)  
{  
    :  
    for(;;)  
    {...}  
}
```

} 内层
循环

04

```
while()  
{  
    :  
    do  
    {...}  
    while();  
    :  
}
```

} 内层
循环

05

```
for(;;)  
{  
    :  
    while()  
    {...}  
    :  
}
```

} 内层
循环

06

```
do  
{  
    :  
    for(;;)  
    {...}  
}while();
```

} 内层
循环

3.4.6 用break和continue语句改变循环流程

■ 用break语句提前终止循环

```
break;
```

作用：使流程跳到循环体之外，接着执行循环体下面的语句。

注意：break语句只能用于循环语句和switch语句之中，而不能单独使用。

■ 用continue语句提前结束本次循环

```
continue;
```

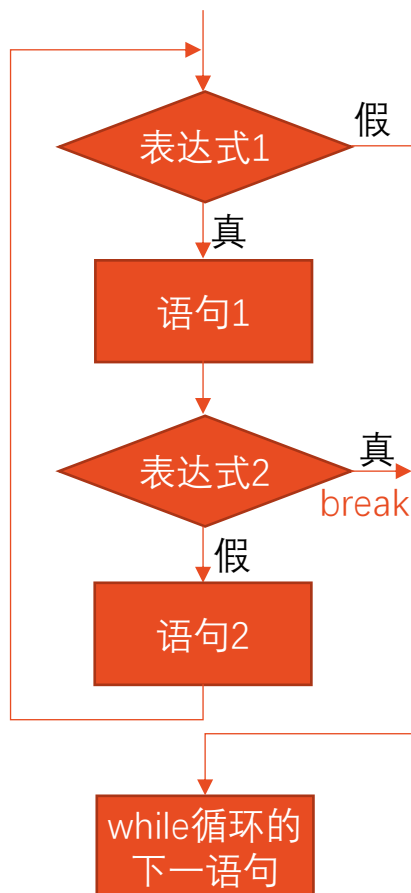
作用：结束本次循环，即跳过循环体中下面尚未执行的语句，转到循环体结束点前，接着执行for语句中的“表达式3”，然后进行下一次是否执行循环的判定。

3.4.6 用break和continue语句改变循环流程

■ break语句和continue语句的区别

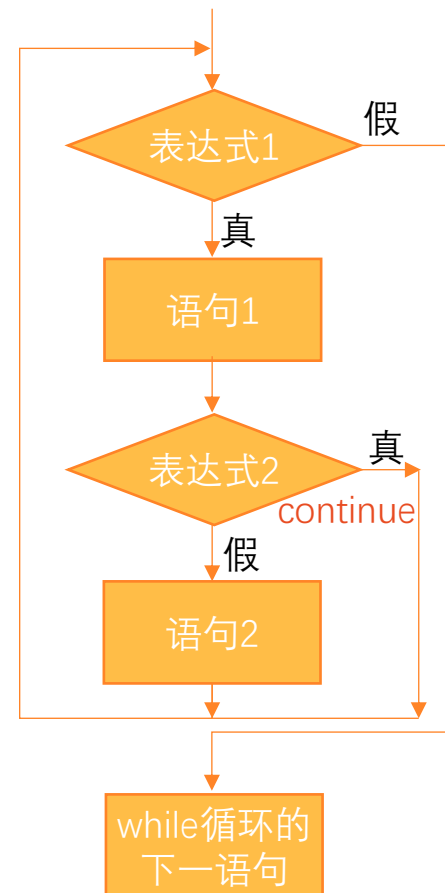
break;

```
while(表达式1)
{
    语句1
    if(表达式2) break;
    语句2
}
```



continue;

```
while(表达式1)
{
    语句1
    if(表达式2) continue;
    语句2
}
```



`continue`语句只结束本次循环，而非终止整个循环。`break`语句结束整个循环，不再判断执行循环的条件是否成立。

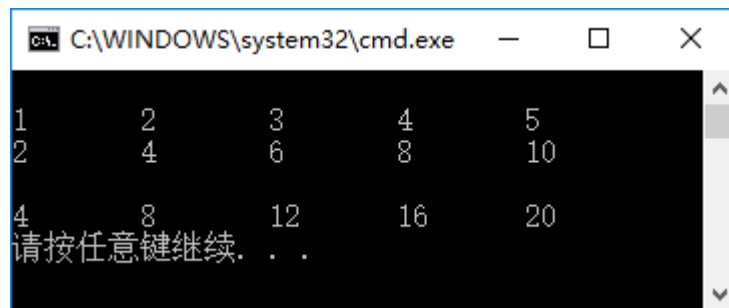
3.4.6 用break和continue语句改变循环流程

■ 示例：输出以下4×5的矩阵

```
1. #include <stdio.h>
2. int main()
3. {
4.     int i,j,n=0;
5.     for(i=1;i<=4;i++)
6.         for(j=1;j<=5;j++,n++)    //n: 累计输出数据个数
7.         {
8.             if(n%5==0) printf("\n"); //输出5个数据后换行
9.             printf("%d\t",i*j);
10.        }
11.    printf("\n");
12.    return 0;
13.}
```

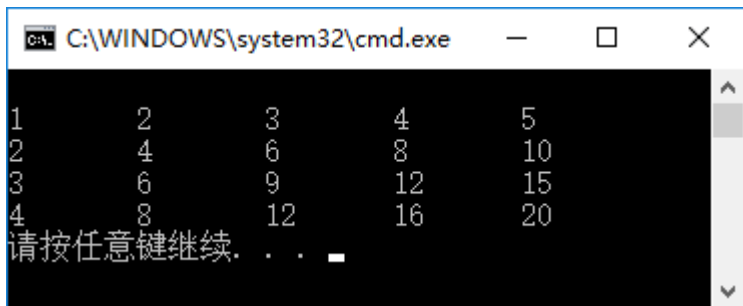
1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20

if (i==3 && j==1) break;

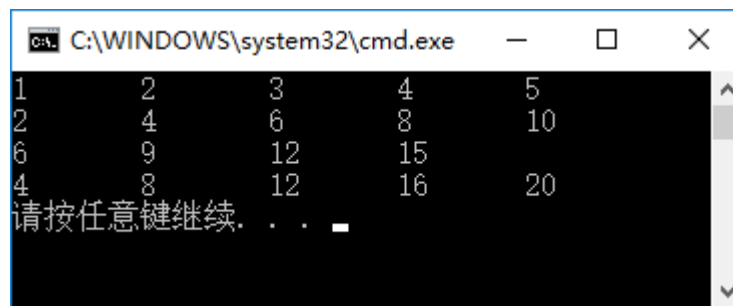


```
C:\WINDOWS\system32\cmd.exe
1      2      3      4      5
2      4      6      8     10
      4      8     12     16     20
请按任意键继续. . .
```

if (i==3 && j==1) continue;



```
C:\WINDOWS\system32\cmd.exe
1      2      3      4      5
2      4      6      8     10
3      6      9     12     15
4      8     12     16     20
请按任意键继续. . .
```



```
C:\WINDOWS\system32\cmd.exe
1      2      3      4      5
2      4      6      8     10
6      9     12     15
4      8     12     16     20
请按任意键继续. . .
```

思考

编程实现博饼游戏的积分累计