

第 2 章

算法——程序的灵魂

算法 + 数据结构 = 程序

Algorithm + Data Structures = Programs



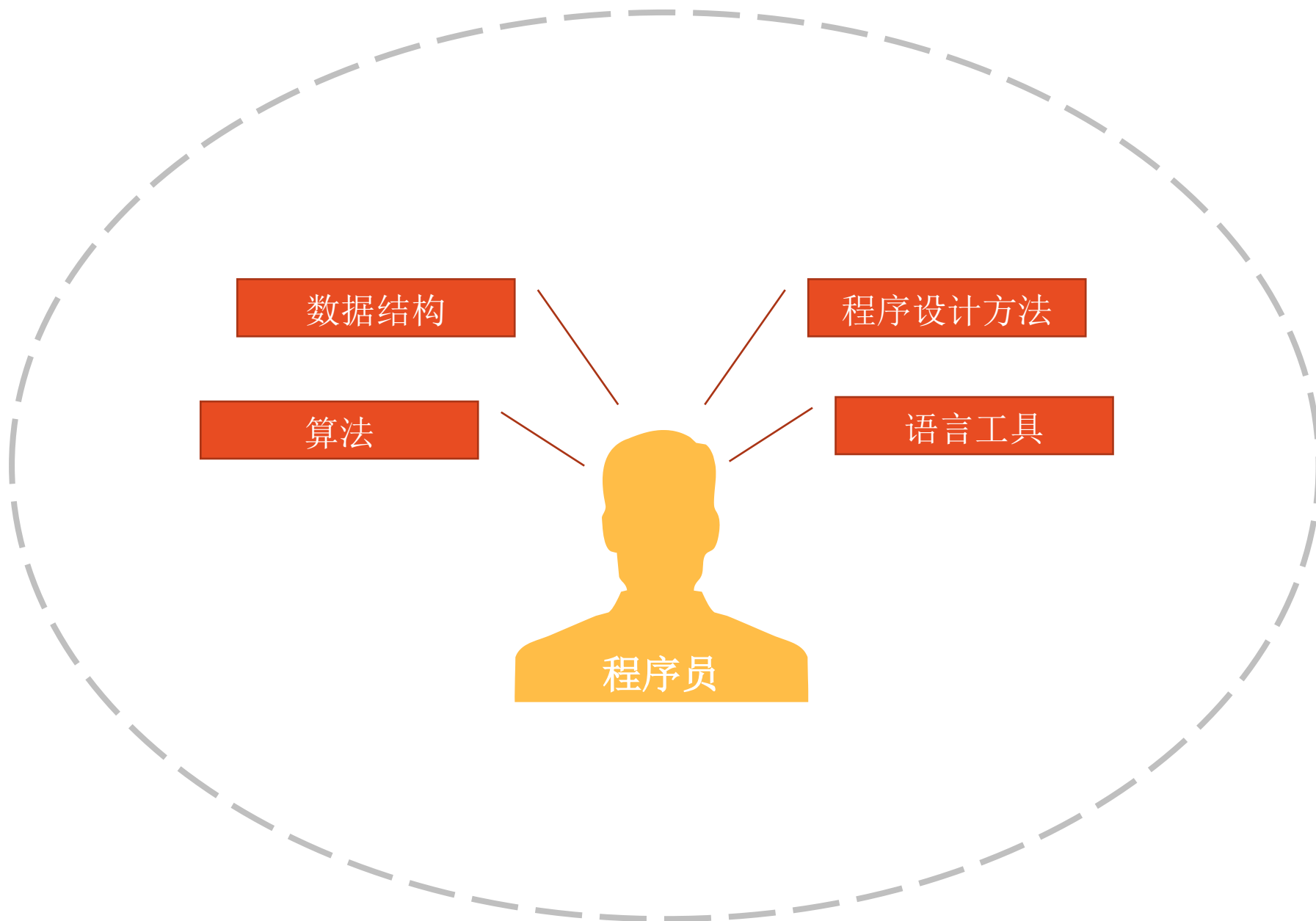
沃思 (Niklaus Wirth)

数据结构

对数据的描述。在程序中要指定用到哪些数据，以及这些数据的类型和数据的组织形式。

算法

对操作的描述。即要求计算机进行操作的步骤



算 法

广义地说，为解决一个问题而采取的方法和步骤，就称为“算法”。

对同一个问题，可以有不同的解题方法和步骤。

为了有效地进行解题，不仅需要保证算法正确，还要考虑算法的质量，选择合适的算法。



算法

数值运算的目的是求数值解。

由于数值运算往往有现成的模型，可以运用数值分析方法，因此对数值运算的算法的研究比较深入，算法比较成熟。

数值运算算法

非数值运算算法

计算机在非数值运算方面的应用远超在数值运算方面的应用。

非数值运算的种类繁多，要求各异，需要使用者参考已有的类似算法，重新设计解决特定问题的专门算法。

简单的算法举例

S: Step
P: Product
i: Item

【例2.1】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



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 $i=2$, 或写成 $2 \Rightarrow i$ (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1, 即 $i+1 \Rightarrow i$
S5: 如果i不大于5 ($i \leq 5$), 返回重新执行S3及其后续步骤S4和S5; 否则, 算法结束。最后得到p的值就是5!的值。

用这种方法表示的算法具有一般性、通用性和灵活性

简单的算法举例

【例2.1】求 $1 \times 2 \times 3 \times 4 \times 5$

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 $i=2$, 或写成 $2 \Rightarrow i$ (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:
 $p \times i \Rightarrow p$
S4: 使 i 的值加1, 即 $i+1 \Rightarrow i$
S5: 如果 i 不大于5 ($i \leq 5$), 返回重新执行S3及其后续步骤S4和S5; 否则, 算法结束。最后得到 p 的值就是 $5!$ 的值。



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

算法步骤

S1: $p = 1$;
S2: $i = 3$;
S3: $p = p \times i$;
S4: $i = i + 2$;
S5: if ($i \leq 11$) goto S3
S6: output p ;

简单的算法举例

【例2.2】有50个学生，要求输出成绩在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: $1 \Rightarrow i$

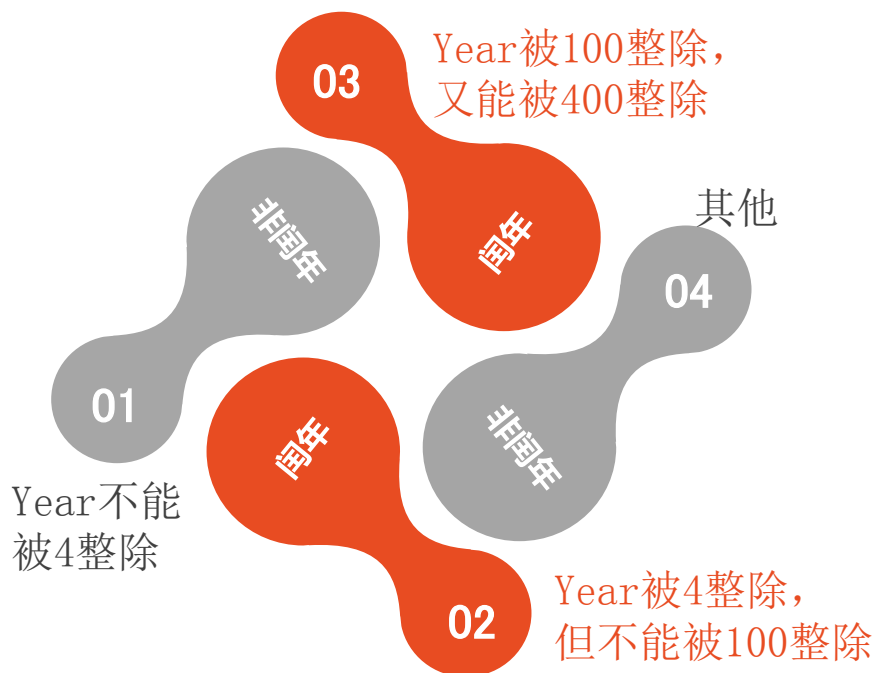
S2: 如果 $g_i \geq 80$, 则输出 n_i 和 g_i , 否则不输出

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

简单的算法举例

【例2.3】判定2000—2500年中的每一年是否为闰年，并将结果输出



算法步骤

S1: 2000= \Rightarrow 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

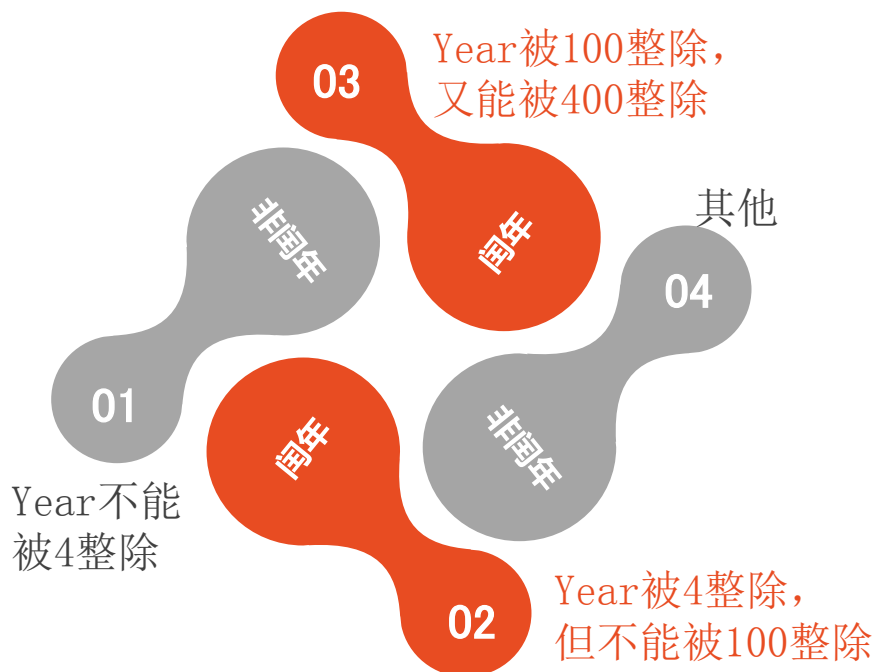
S5: 输出year的值和“不是闰年”

S6: year+1= \Rightarrow year

S7: 当year \leq 2500时, 转S2继续执行, 否则算法停止

简单的算法举例

【例2.3】判定2000—2500年中的每一年是否为闰年，并将结果输出



算法步骤

S1: 2000= \Rightarrow year

S2: 若year能被4整除，但不能被100整除，则输出year 的值和“是闰年”。然后转到S5，检查下一年份

S3: 若year能被100整除，并且被400整除，输出year的值和“是闰年”。然后转到S5，检查下一年份

S4: 输出year的值和“不是闰年”

S5: year+1= \Rightarrow year

S6: 当year \leq 2500时，转S2继续执行，否则算法停止

简单的算法举例

【例2.4】求 $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots + \frac{1}{99} - \frac{1}{100}$

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

简单的算法举例

【例2.5】 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： 所谓素数(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: $i = i + 1$

S6: 如果 $i \leq n-1$ ，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。

算法的特性

1

有穷性

一个算法应包含有限的操作步骤，而不能是无限的

2

确定性

算法中的每一个步骤都应当是确定的，而不应当是含糊的、模棱两可的

3

有零个或多个输入

所谓输入是指在执行算法时需要从外界取得必要的信息

4

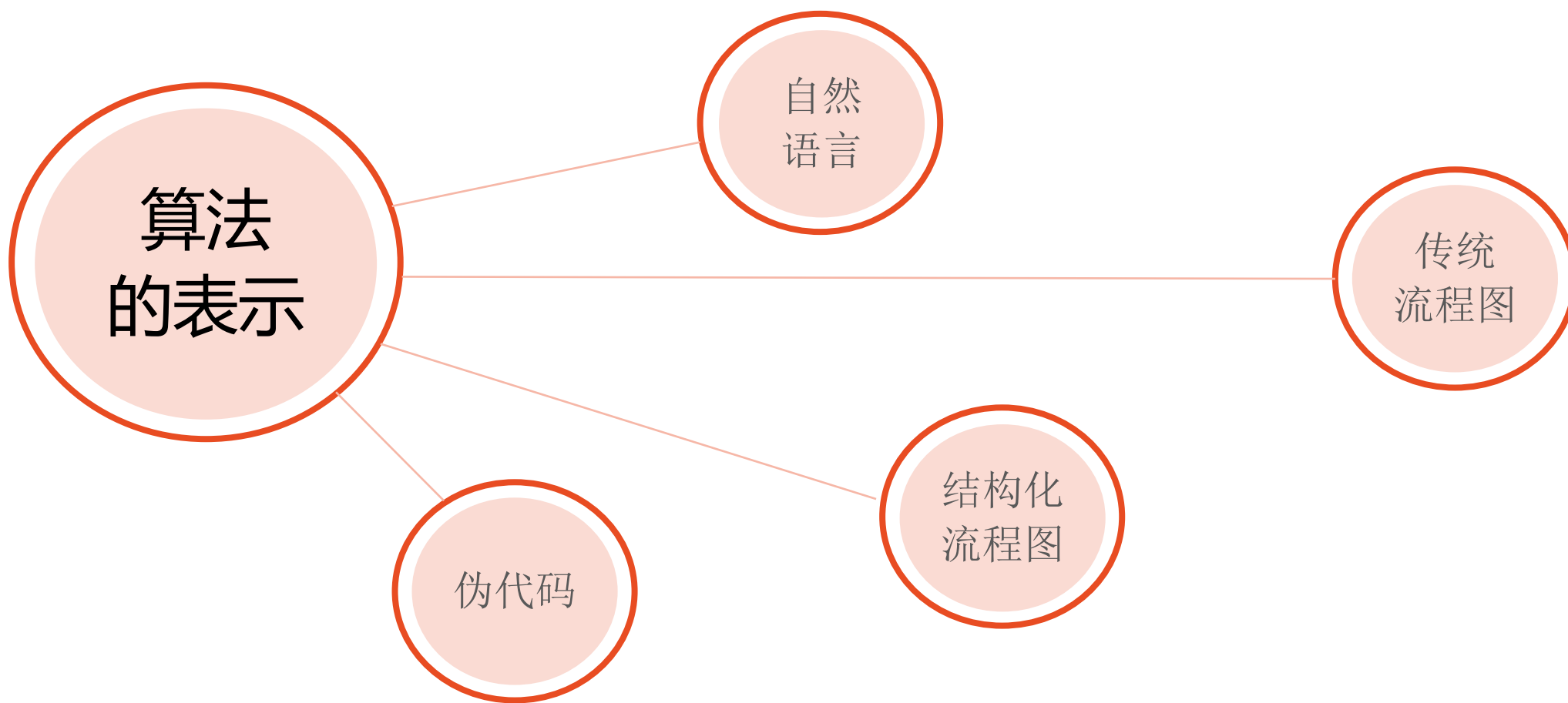
有一个或多个输出

算法的目的是为了求解，“解” 就是输出

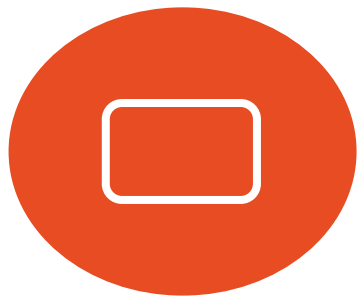
5

有效性

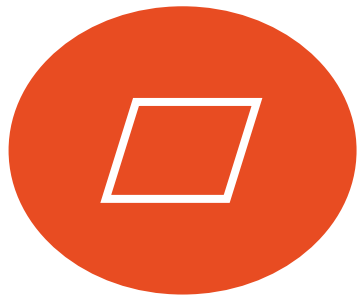
算法中的每一个步骤都应当能有效地执行，并得到确定的结果



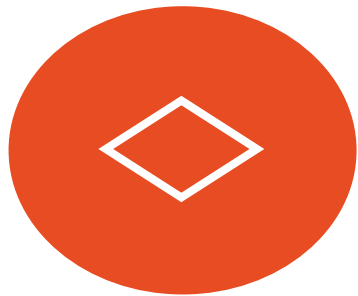
用流程图表示算法



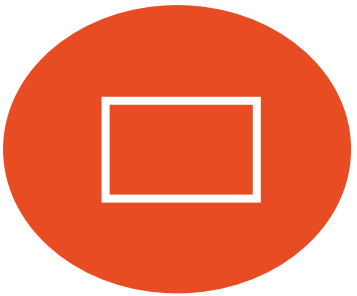
起止框



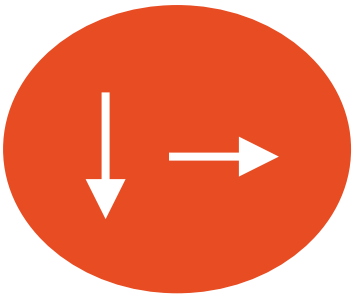
输入输出框



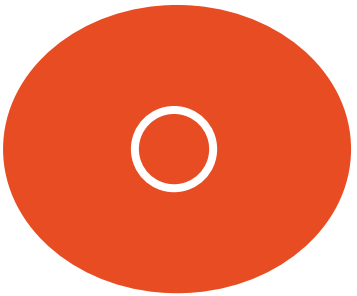
判断框



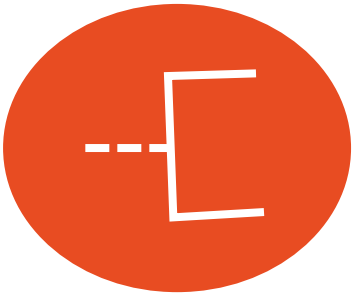
处理框



流程线



连接点



注释框

算法的流程图表示举例

【例2.6】将例2.1的算法用流程图表示。

求 $1 \times 2 \times 3 \times 4 \times 5$ 。

算法步骤

S1: $1 \Rightarrow p$

S2: $2 \Rightarrow i$

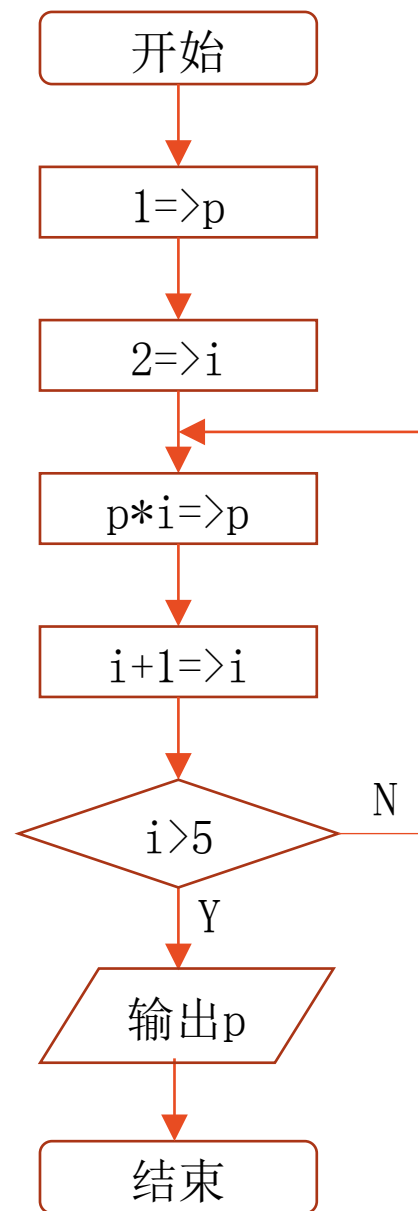
S3: $p * i \Rightarrow p$

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



算法的流程图表示举例

【例2.7】例2.2的算法用流程图表示。

有50个学生，要求输出成绩在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

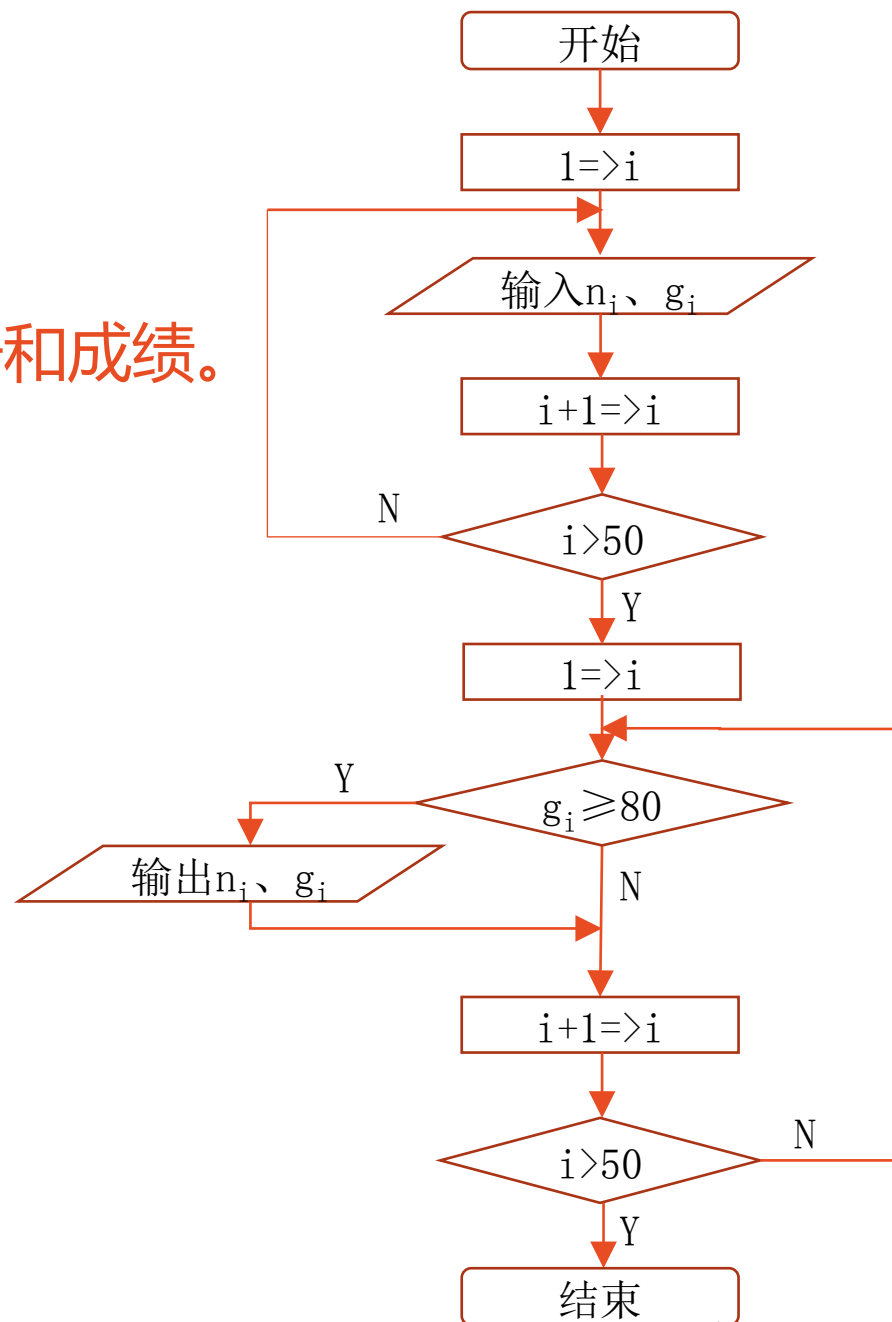
算法步骤

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: $i+1 \Rightarrow i$

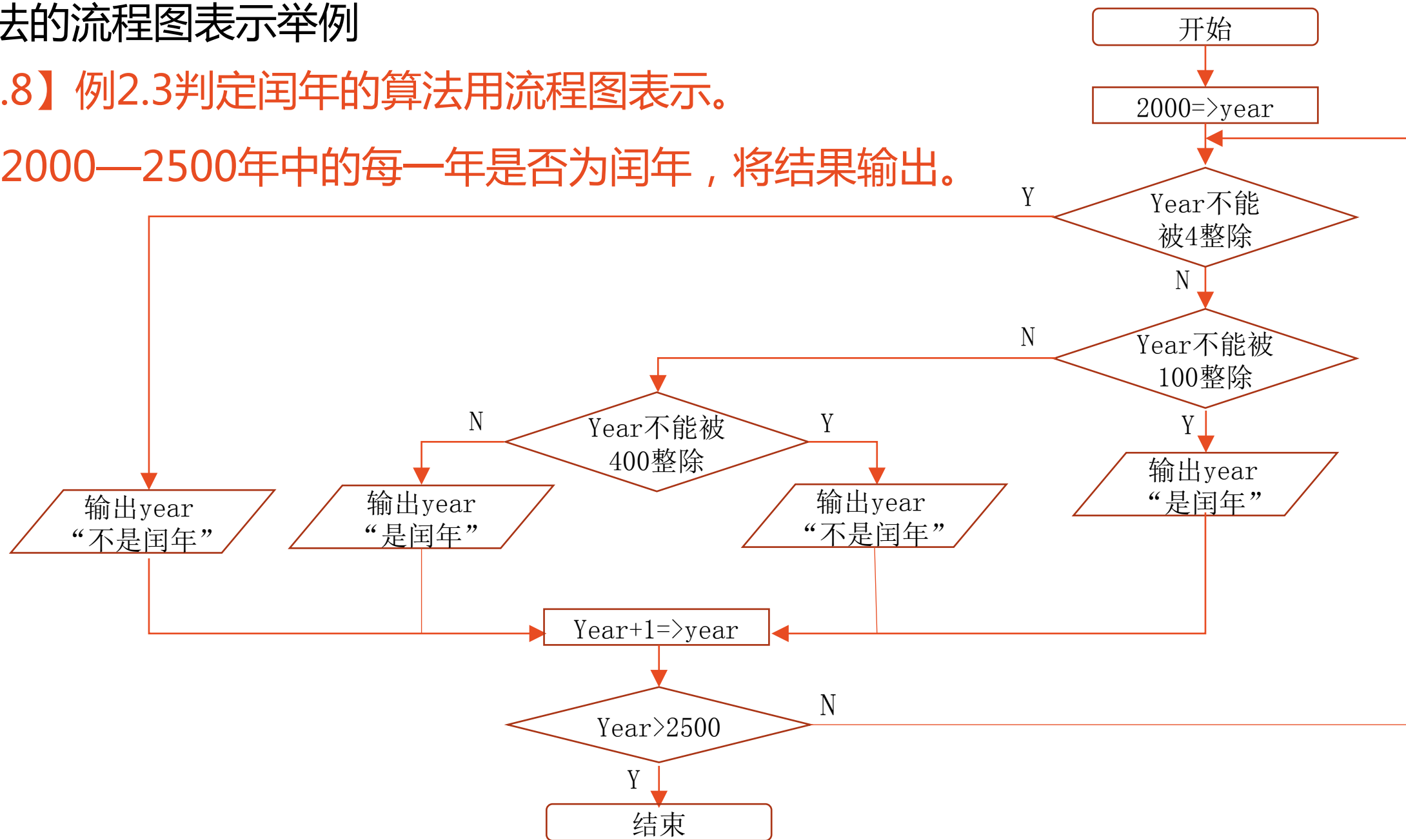
S4: 如果 $i \leq 50$ ，返回到S2，继续执行，否则，算法结束



算法的流程图表示举例

【例2.8】例2.3判定闰年的算法用流程图表示。

判定2000—2500年中的每一年是否为闰年，将结果输出。



算法的流程图表示举例

【例2.9】将例2.4的算法用流程图表示。

$$\text{求 } 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots + \frac{1}{99} - \frac{1}{100}$$

算法步骤

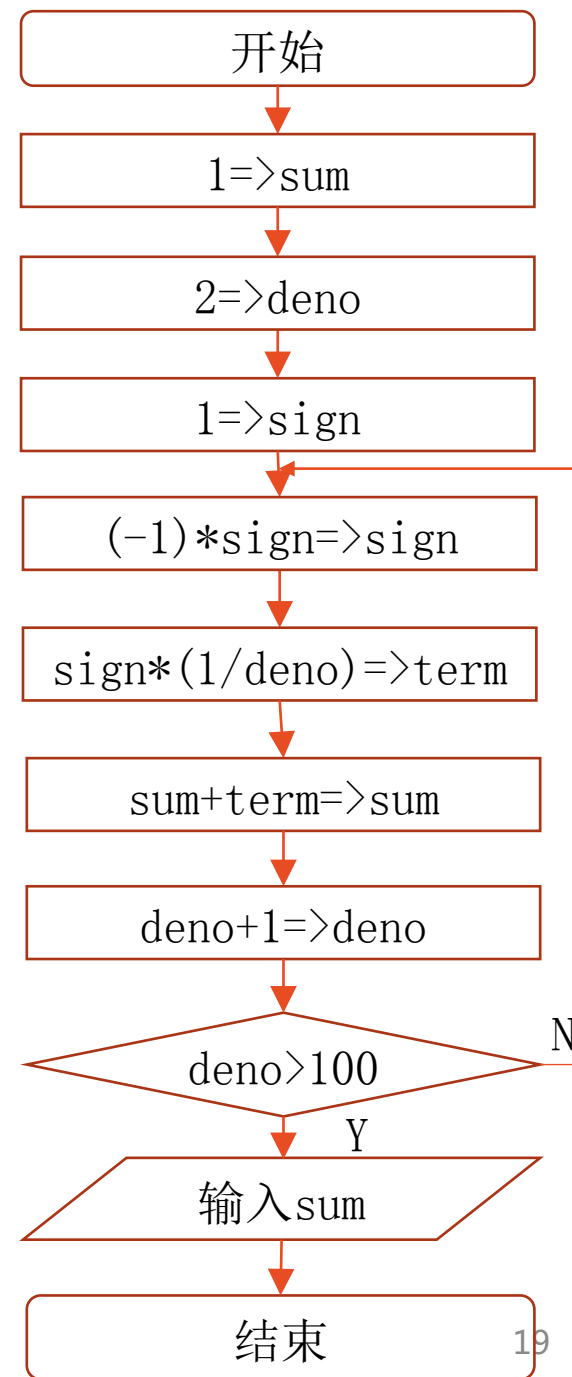
S1: sign = 1
S2: sum = 1
S3: deno = 2
S4: sign = (-1)*sign
S5: term = sign*(1/deno)
S6: sum = sum+term
S7: deno = deno+1
S8: 若deno≤100返回S4；否则算法结束

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



简单的算法举例

【例2.10】例2.5判断素数的算法用流程图表示。

对一个大于或等于3的正整数，判断它是不是一个素数。

算法步骤

S1: 输入n的值

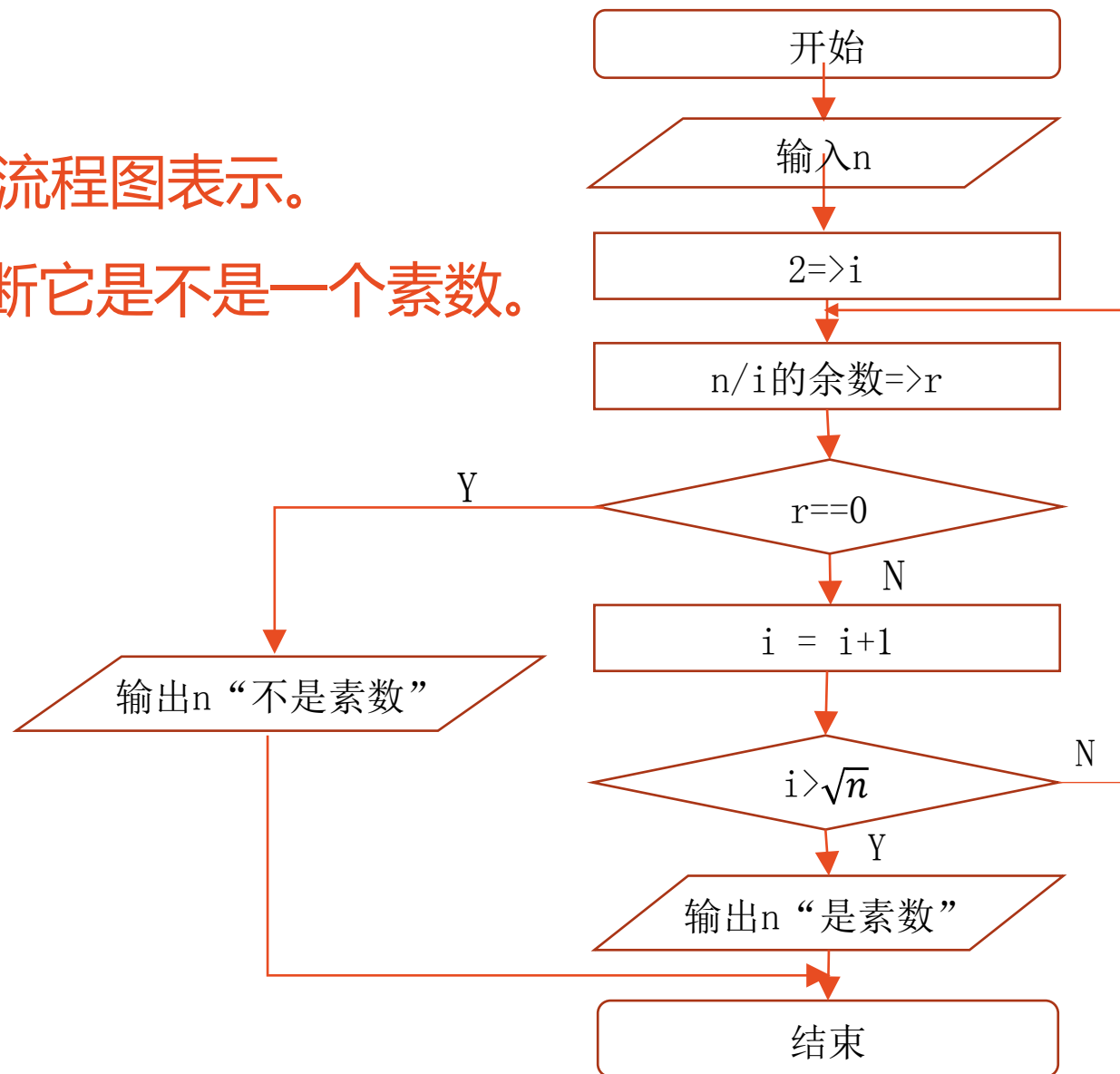
S2: $i = 2$ (i作为除数)

S3: n被i除，得余数r

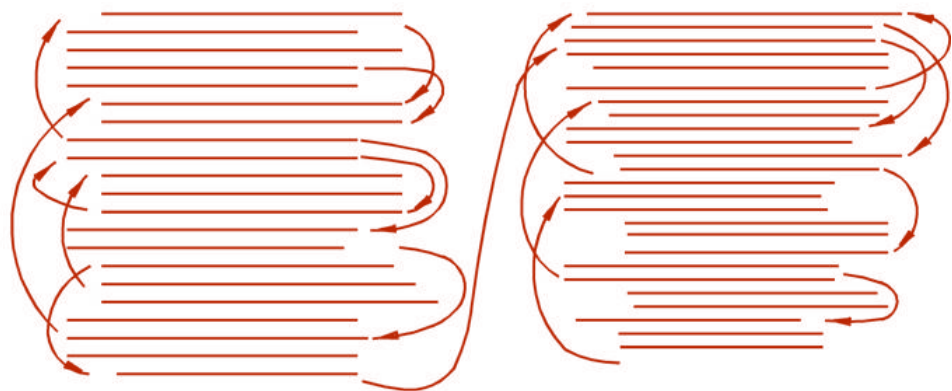
S4: 如果 $r = 0$ ，表示n能被i整除，则输出n“不是素数”，算法结束；否则执行S5

S5: $i = i + 1$

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

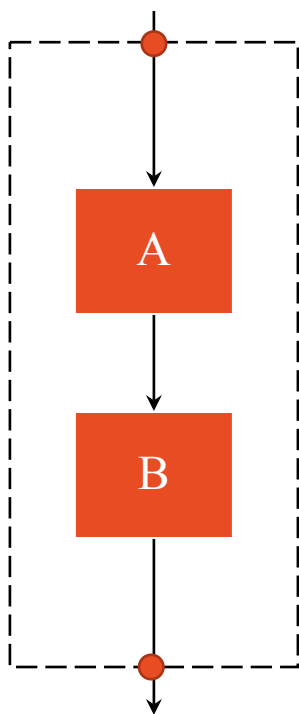


传统流程图的弊端

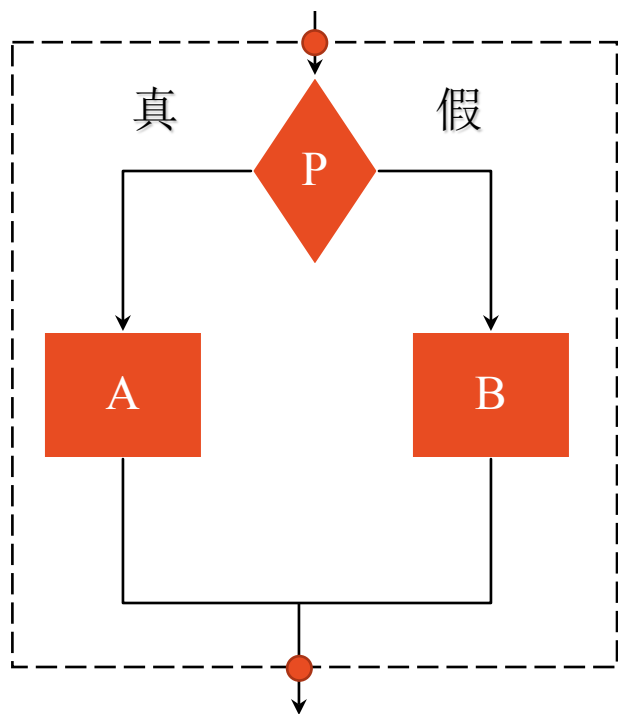


传统的流程图用流程线指出各框的执行顺序，对流程线的使用没有严格限制。因此，使用者可以不受限制地使流程随意地转来转去，使流程图变得毫无规律，阅读时要花很大精力去追踪流程，使人难以理解算法的逻辑。

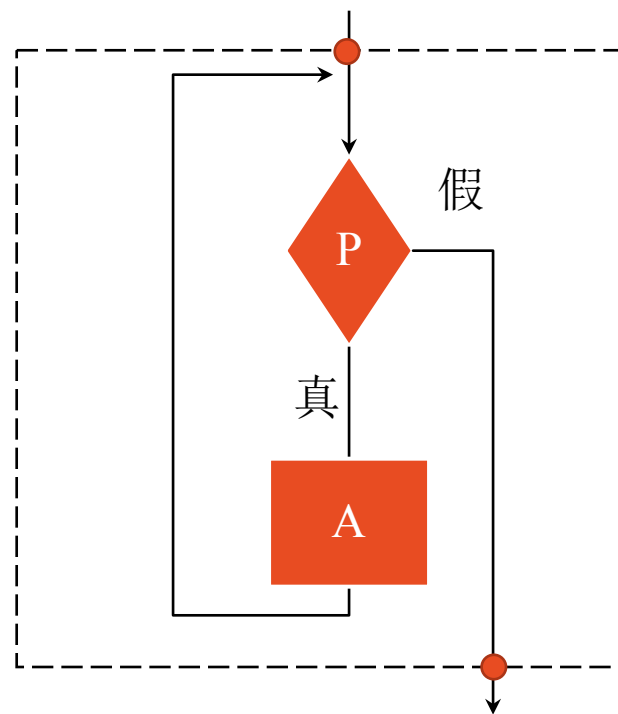
三种基本结构



顺序结构



选择结构



循环结构

三种基本结构的特点



只有一个入口



只有一个出口

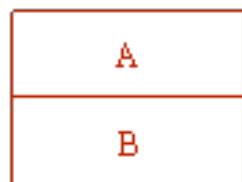


结构内的每一部分都有机会被执行到

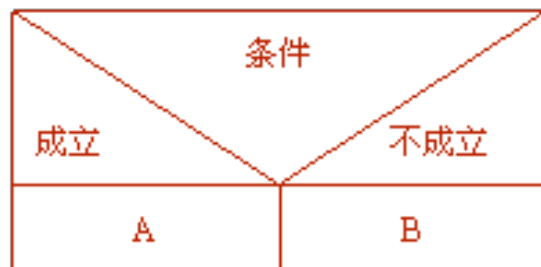


结构内不存在“死循环”

用N-S流程图表示算法



(a) 顺序结构



(b) 分支结构



(c) 循环结构(条件在前)

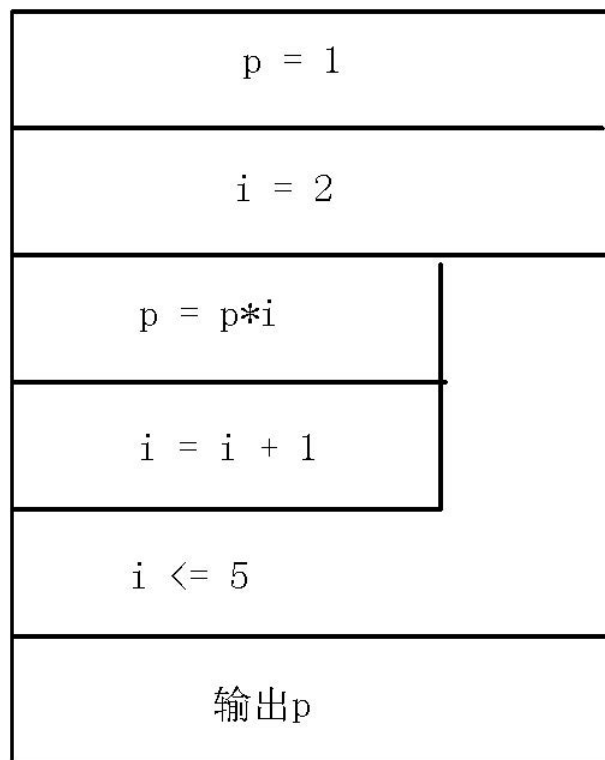


(d) 循环结构(条件在后)

传统流程图与N-S流程图

【例2.6】将例2.1的算法用流程图表示。

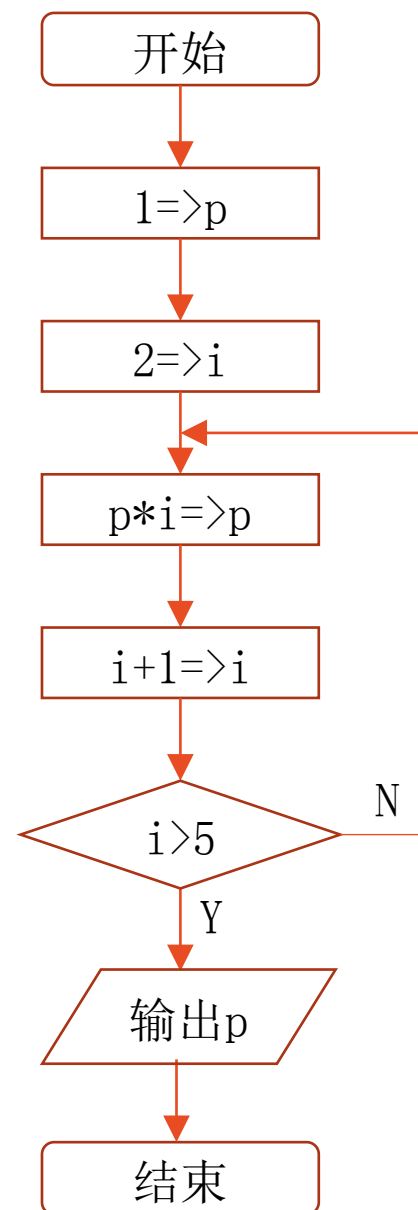
求 $1 \times 2 \times 3 \times 4 \times 5$ 。



P: 表示被乘数

i: 表示乘数

10:29:26

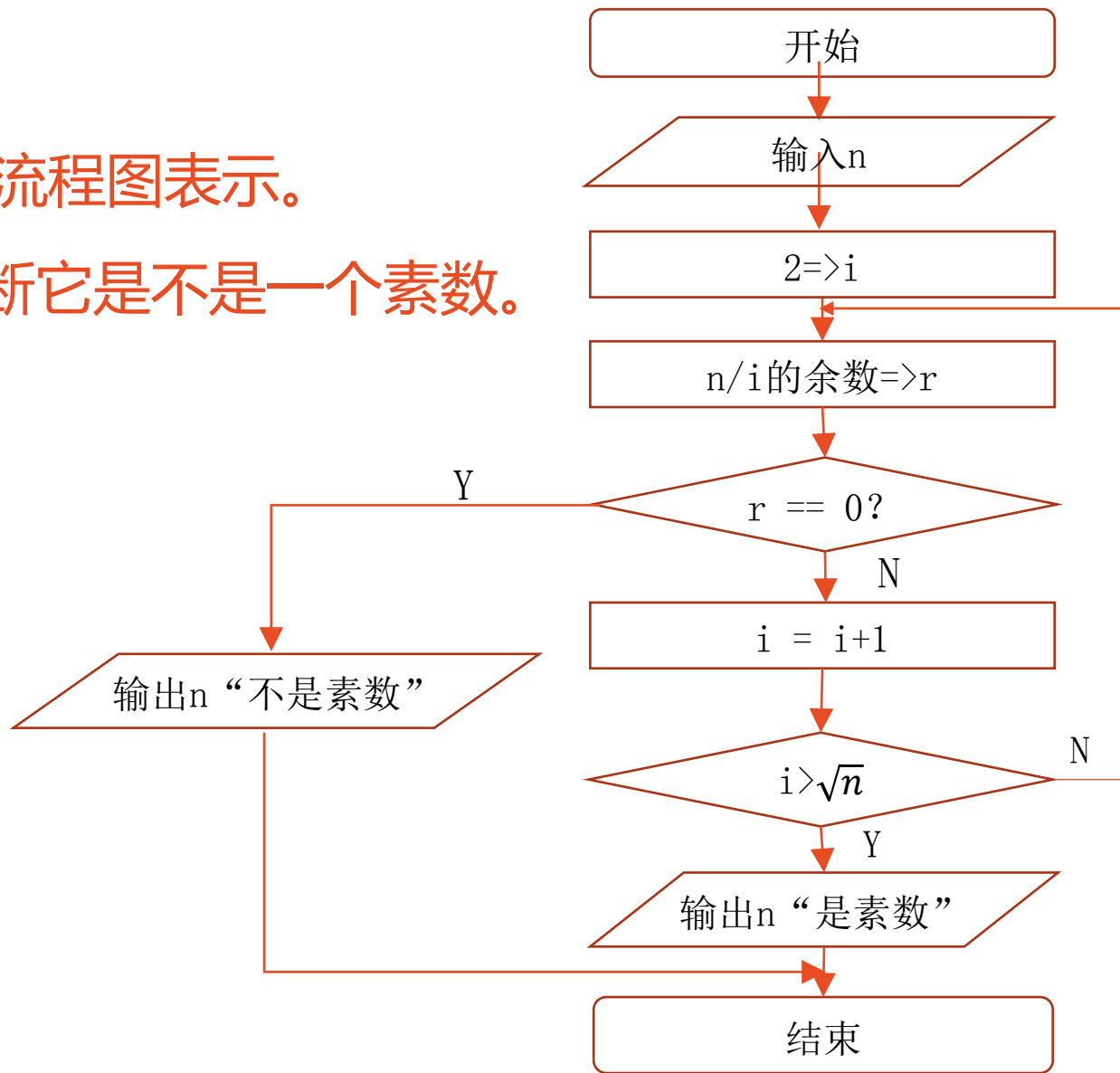
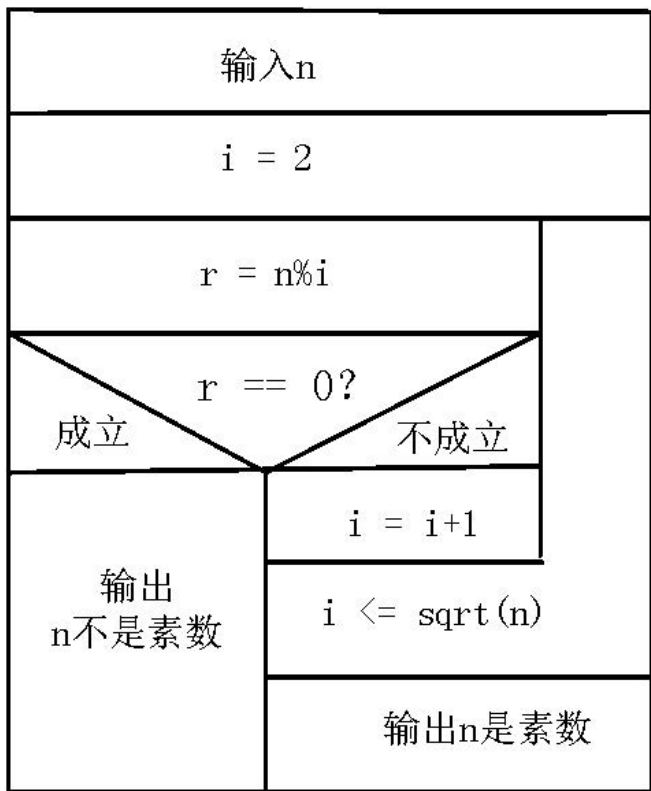


25

传统流程图与N-S流程图

【例2.10】例2.5判断素数的算法用流程图表示。

对一个大于或等于3的正整数，判断它是不是一个素数。

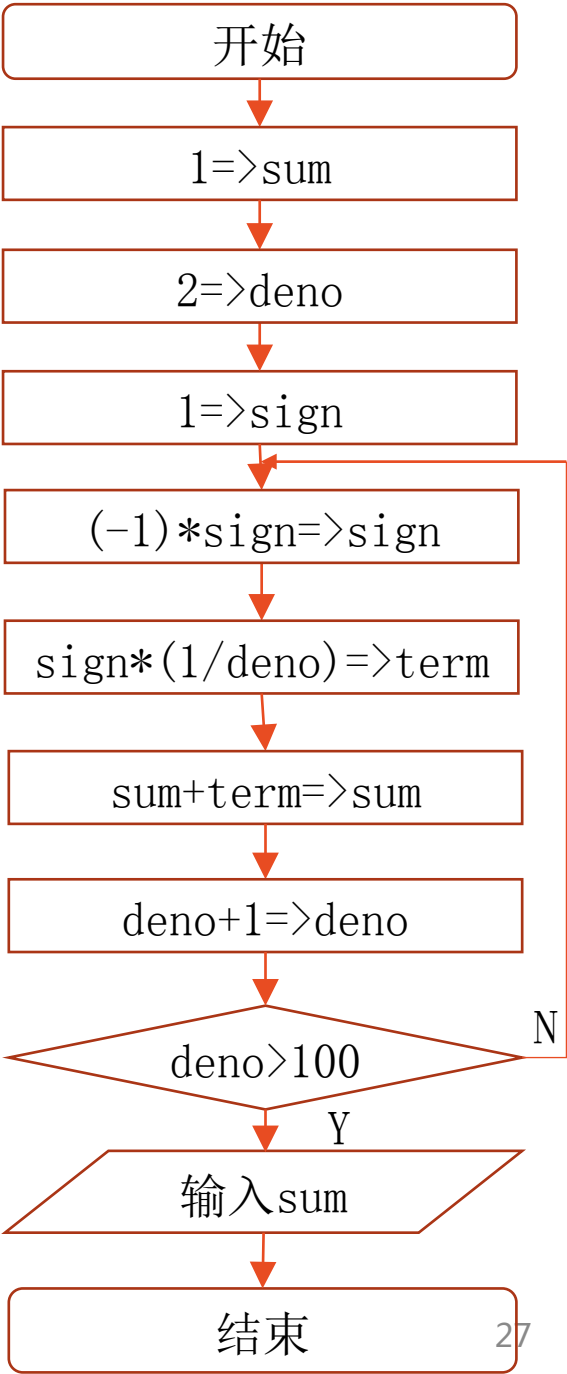
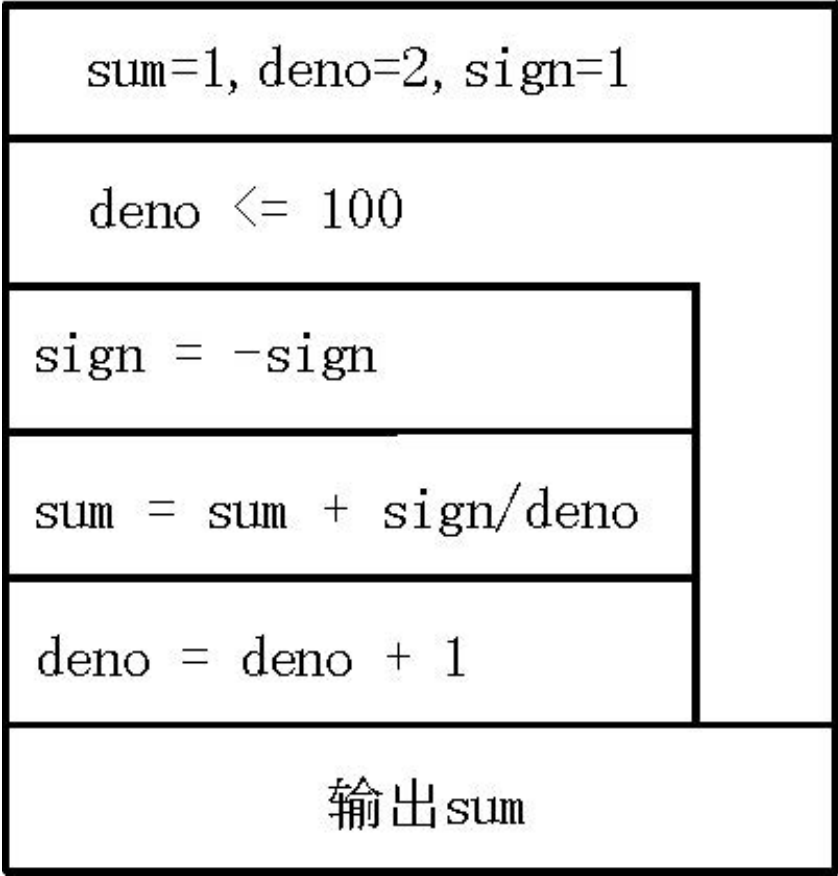


传统流程图与N-S流程图

【例2.9】将例2.4的算法用流程图表示。

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
sum: 表示当前项的累加和
deno: 表示当前项的分母





用伪代码表示算法

伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法。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。



算法的流程图表示举例

【例2.16】求5!，用伪代码表示。

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

begin (算法开始)

$1 \Rightarrow p$

$2 \Rightarrow i$

while $i \leq 5$

{ $p * i \Rightarrow p$

$i + 1 \Rightarrow i$

}

print p

end (算法结束)

伪代码

算法的流程图表示举例

【例2.17】求 $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + \frac{1}{99} - \frac{1}{100}$ ，用伪代码表示。

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

```
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

用计算机语言表示算法

【例2.18】将例2.16表示的算法（求5!）用C语言表示。

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

```
#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;
}
```

用计算机语言表示算法

【例2.18】将例2.17表示的算法求 $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots + \frac{1}{99} - \frac{1}{100}$ 的值用C语言表示。

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

```
#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;
}
```


结构化程序设计方法

