



地球与空间科学系
DEPARTMENT OF EARTH AND SPACE SCIENCES

C程序设计基础

Introduction to C programming Lecture 2: Algorithms

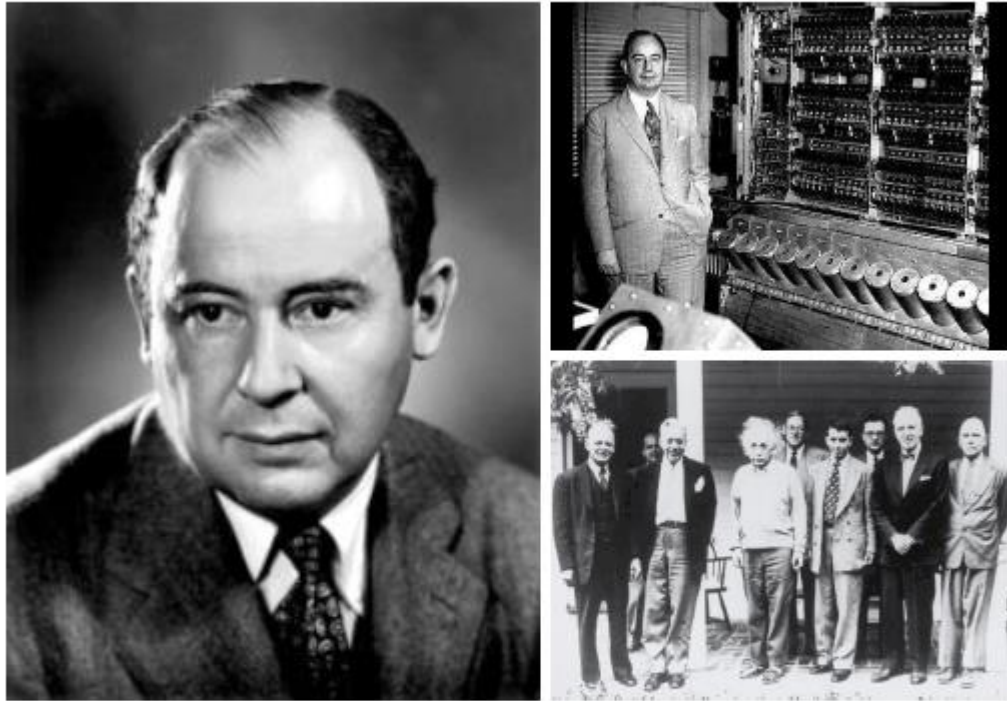
张振国 zhangzg@sustech.edu.cn

南方科技大学/理学院/地球与空间科学系

加课
Y/N?

Review on L1

Von Neumann architecture



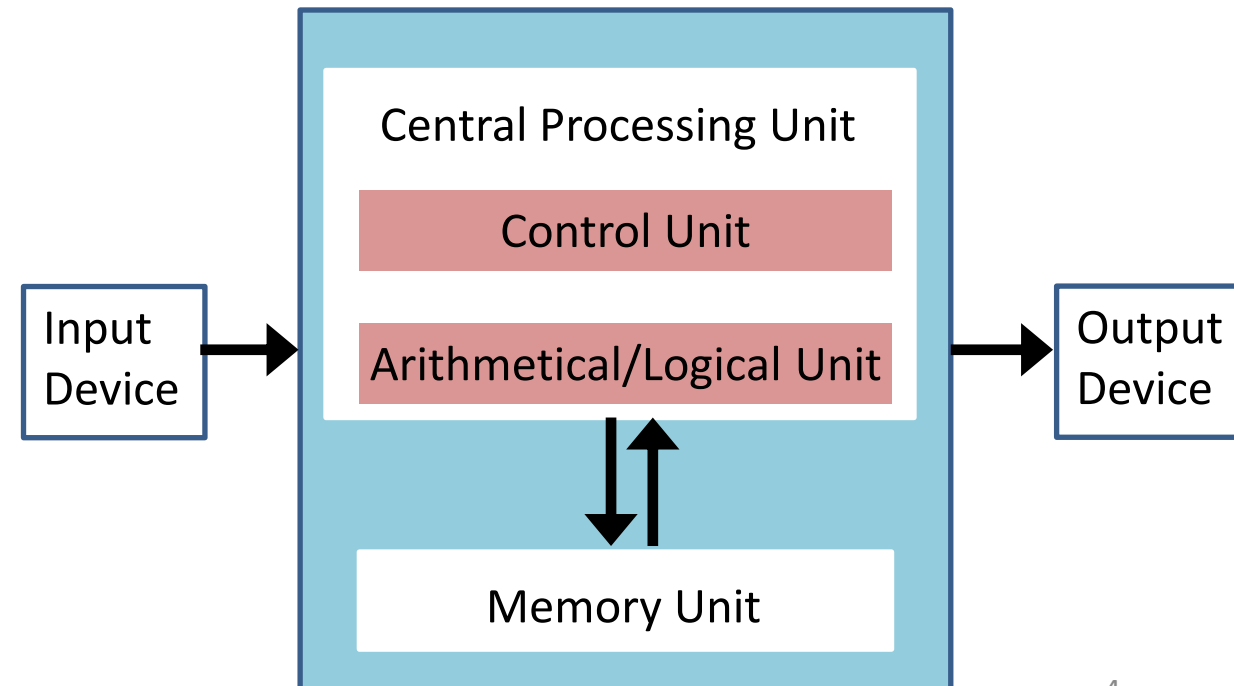
John von Neumann (1903-1957)

Hungarian-American mathematician physicist
Founder of modern computer architecture

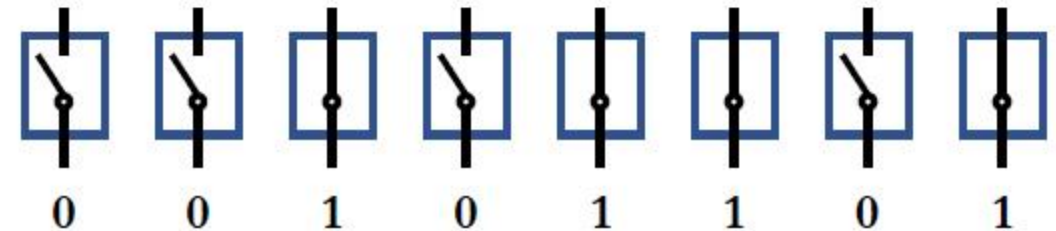
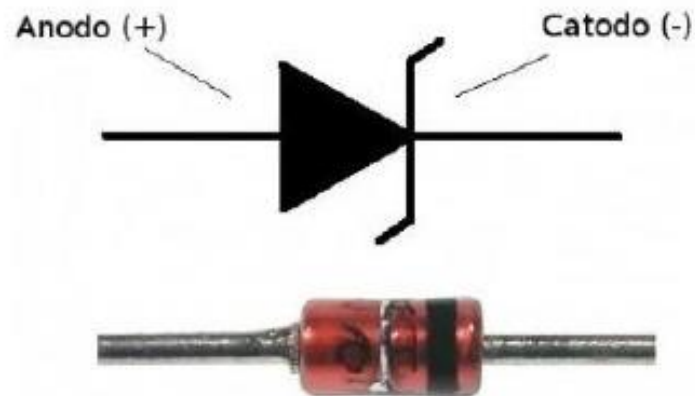
Machine is programmable

Von Neumann architecture (1946)

冯·诺依曼架构



- Human languages are complex;
- Machine speaks binary language: 0 and 1;
- A computer is nothing but a vast collection of electronic switches to store information.

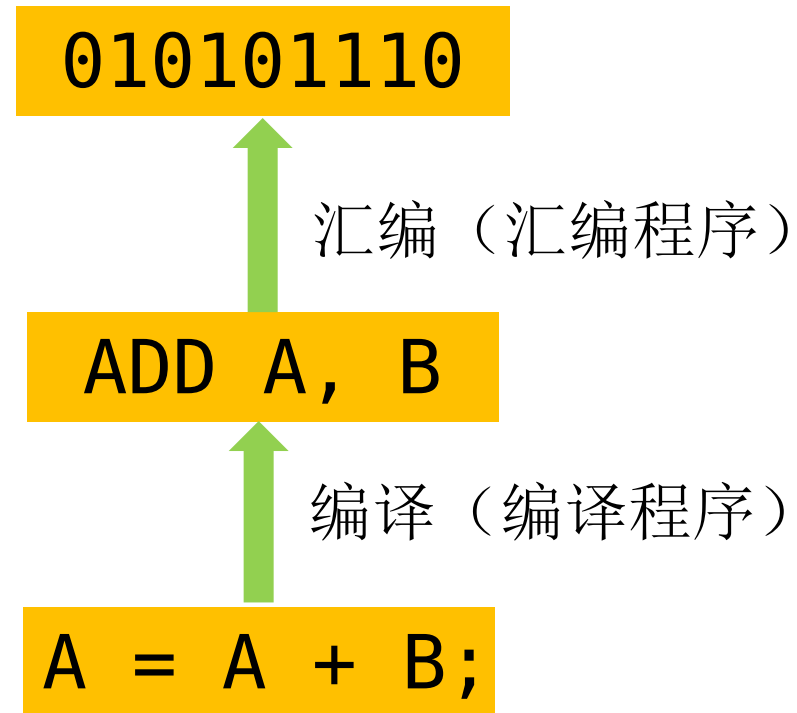


We need a language not
a tool.



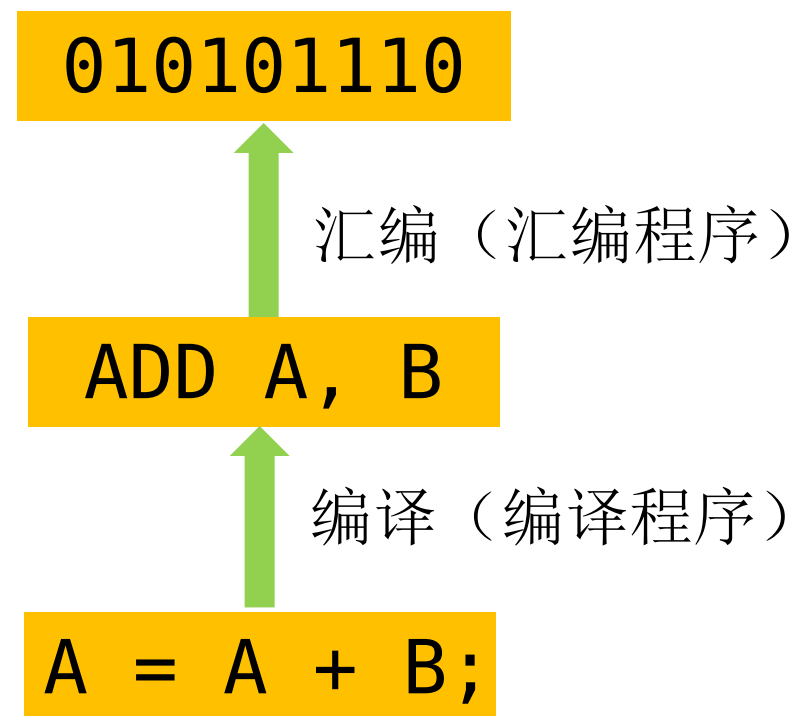
Programming languages

- Machine language
- Symbolic language
- High-level language



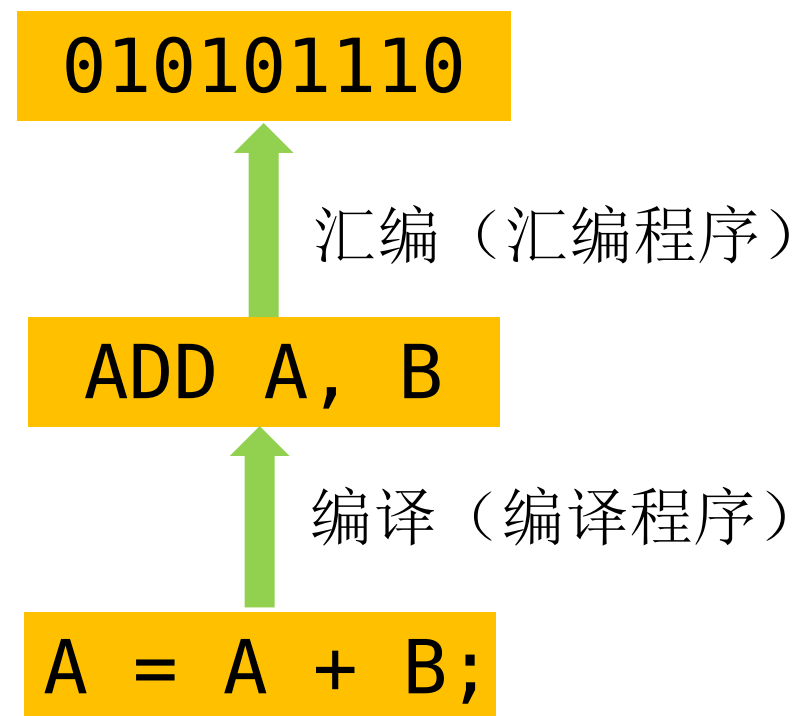
Programming languages

- High-level language
 - Unstructured language
 - 典例: 早期BASIC、COBOL、FORTRAN等语言
 - 特点: 不能反映结构程序设计的思想
 - 缺点: 程序的逻辑比较复杂



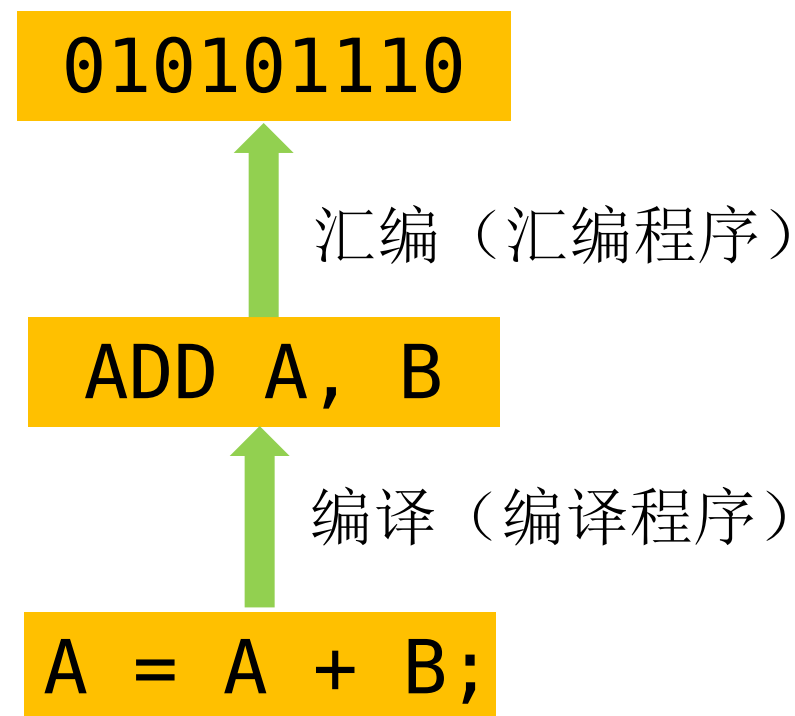
Programming languages

- High-level language
 - Unstructured language
 - Structured language
 - 规定程序必须由具有良好特性的基本结构（顺序、选择、循环结构）
 - 特点: 清晰易读和逻辑严密
 - QBASIC, FORTRAN, C

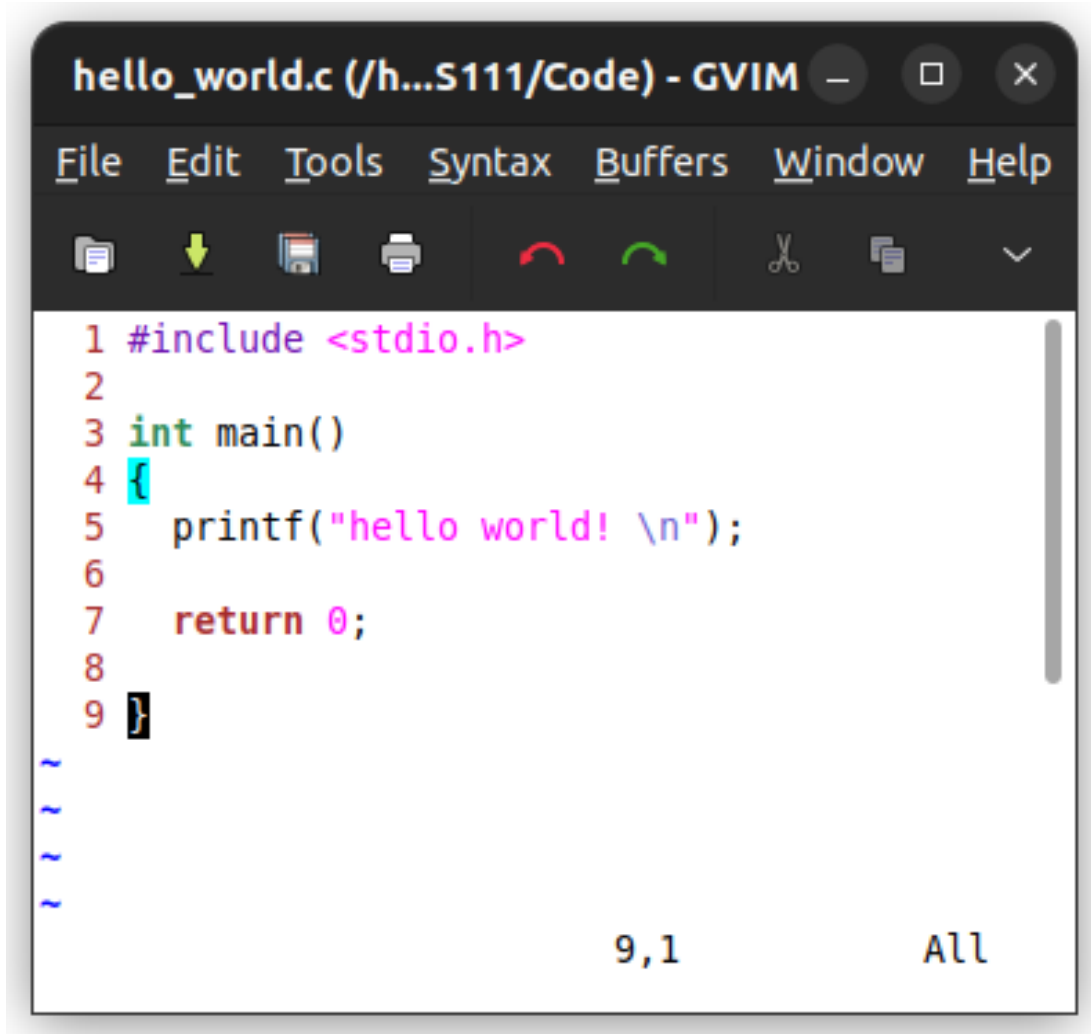


Programming languages

- High-level language
 - Unstructured language
 - Structured language
 - Object-Oriented Language
 - C++, C#, Visual Basic, Java
 - 处理规模较大的问题
- 基于过程的语言
- 面向对象的语言



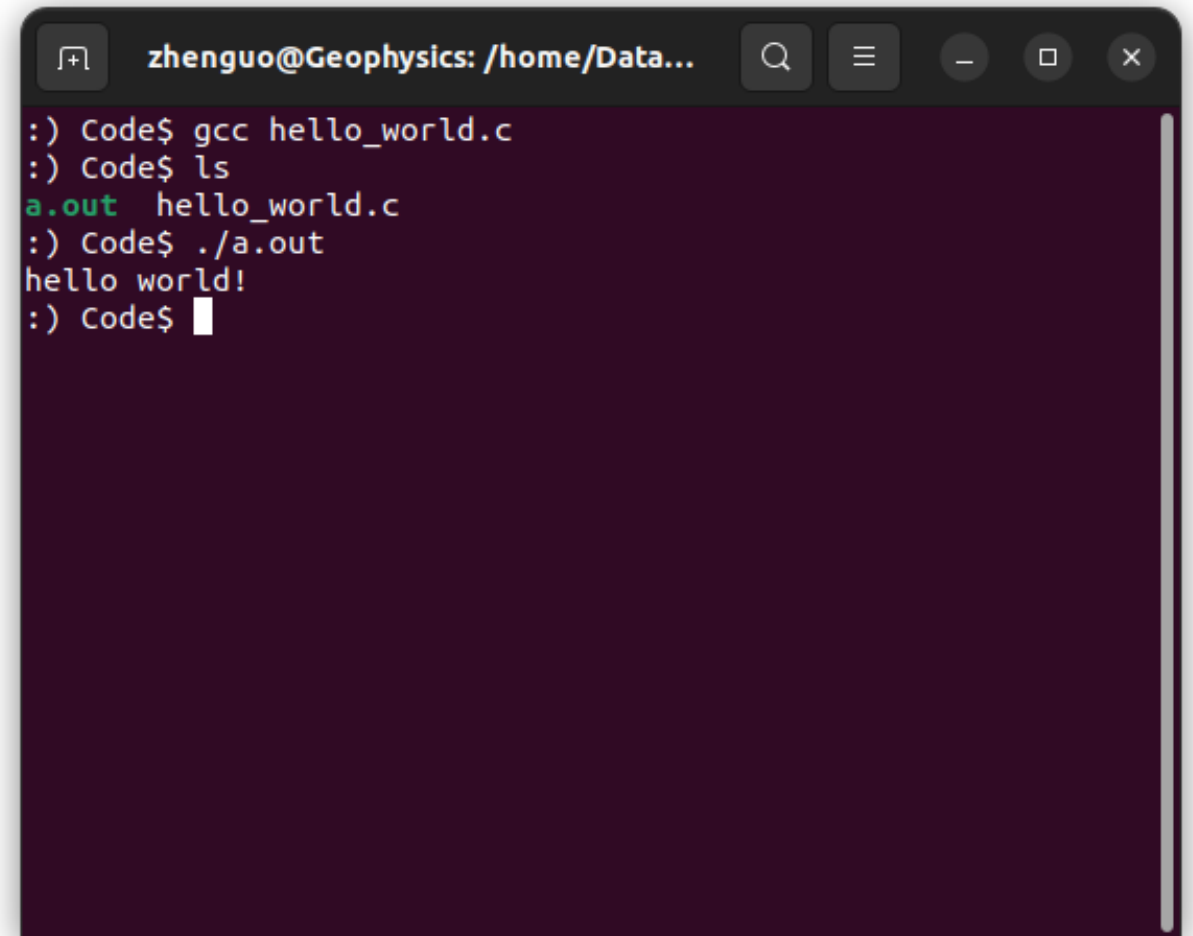
A first example



The screenshot shows a GVIM editor window titled "hello_world.c (/h...S111/Code) - GVIM". The menu bar includes File, Edit, Tools, Syntax, Buffers, Window, and Help. Below the menu is a toolbar with icons for file operations and editing. The main text area contains the following C code:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello world! \n");
6
7     return 0;
8
9 }
```

At the bottom right of the editor, the status bar displays "9,1" and "All".



The screenshot shows a terminal window titled "zhenguo@Geophysics: /home/Data...". The terminal displays the following commands and output:

```
:~ Code$ gcc hello_world.c
:~ Code$ ls
a.out hello_world.c
:~ Code$ ./a.out
hello world!
:~ Code$
```

简单的C语言程序介绍

```
#include <stdio.h>
```

```
int main(void )
```

```
{
```

```
    printf ("Hello world!\n");
```

```
    return 0;
```

```
}
```

```
/*文件包含*/
```

```
/*主函数 */
```

```
/*函数体开始*/
```

```
/*输出语句*/
```

```
/*函数体结束*/
```

说明： main-主函数名， int-函数类型

- 每个C程序必须有一个主函数main
- { }是函数开始和结束的标志, 不可省
- 每个C语句以分号结束
- 使用标准库函数时应在程序开头一行写:

```
#include <stdio.h>
```

说明： 输出一行信息: sum is 579

例 求两数之和

```
#include <stdio.h>
void main( )      /*求两数之和*/
{
    int a, b, sum;  /*声明，定义变量为整型*/
    /*以下3行为C语句 */
    a=123; b=456;
    sum=a+b;
    printf(" sum is %d\n" , sum);
}
```

说明： `/*.....*/`表示注释。注释只是给人看的, 对编译和运行不起作用。所以可以用汉字或英文字符表示, 可以出现在一行中的最右侧, 也可以单独成为一行。//注释后续行内容

例 求3个数中较大者。

```
#include <stdio.h>
void main( )    /* 主函数*/
{
    int max(int x, int y); / 对被调用函数max的声明 */
    int a, b, c;      /*定义变量a、b、c */
    scanf(" %d, %d" , &a, &b); /*输入变量a和b的值*/
    c=max(a, b); /*调用max函数, 将得到的值赋给c */
    printf(" max=%d\\n" , c); /*输出c的值*/
}
```

说明：本程序包括main和被调用函数max两个函数。max函数的作用是将x和y中较大者的值赋给变量z。return语句将z的值返回给主调函数main。

- 程序运行情况如下：
- 8, 5 ↙ (输入8和5赋给a和b)
- max=8 (输出c的值)

```
int max(int x, int y)
{
    int z;
    if (x>y) z=x;
    else z=y;
    return (z);
}
```

Homework1

Good examples

%.2f Represents a floating-point number output to **2** decimal places

(5) Print an equation multiplying two float numbers

```
int main(void)
{
    float a, b;
    printf("请输入第一个数:");
    scanf_s("%f", &a);

    printf("请输入第二个数: ");
    scanf_s("%f", &b);

    printf(" %.2f * %.2f = %.2f\n", a, b, a * b);
}
```



```
Microsoft Visual Studio 调试  x + -
请输入第一个数:2.2
请输入第二个数:3.3
2.20 * 3.30 = 7.26
C:\Users\29927\source\repos\Hello\x64\Debug\Hello.exe (进程 11588)已退出。代码为 0。
按任意键关闭此窗口。 . . .]
```

Comments are good coding practice

```
#include <stdio.h>

int main() {
    /*求两数之乘*/
    float a, b, multiplication; /*声明, 定义变量为浮点型*/
    /*以下3行为C语句*/
    a = 1.23;
    b = 4.56;
    multiplication = a * b;
    printf("multiplication is %f\n", multiplication);
    return 0;
}
```

Problems in the assignment

(5) Print an equation multiplying two float numbers

```
5, #include<stdio.h>

int main()
{
    int a=2,b=1;
    int c=a*b;
    printf("The result is %d.\n",c);
    return 0;
}
```

Visual Studio Debug Console: × + ▾

The result is 2.
请按任意键继续. . . |

```
#include<stdio.h>

double multiply(double x, double y){
    return x*y;
}

int main(){
    double result;
    double a = 1.5;
    double b = 2.5;
    result = multiply(a,b);
    printf("%f * %f = %f\n",a,b,result);
    return 0;
}
```

Problems in the assignment

(6) Briefly describe how a C program is executed (in 200 words, bonus)

1. Edit: Enter the source program and save it as a .c file.
2. **Compile:** Translate the source program into the object file .obj.
3. **Link:** Link the object file and runtime library, convert to an executable program .exe.
4. Run: Run the .exe file to obtain the running result.

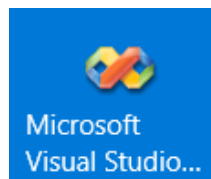
Problems in the assignment: Tips

- Submit a **PDF** assignment and put all the content in **one** file.
- Take screenshots: windows shortcut to **win+shift+s**.
- A full score of 60 is required **after the submission deadline**.

program = algorithm + data
程序 算法 数据



QQ.exe



Visual Studio.exe



iexplore.exe



winamp.exe

data

- 整数 1
- 实数 1.0
- 复数 (1.0, 0.0)
- 逻辑数 true
- 字符 "A"
- ...



010101110

程序的灵魂——算法

● 本章要点

- 算法的概念
- 算法的表示
- 结构化程序设计方法

主要内容

2.1 算法的概念

2.2 简单算法举例

2.3 算法的特性

2.4 怎样表示一个算法

2.5 结构化程序设计方法

一个程序应包括两个方面的内容：

- 对数据的描述：数据结构(data structure)
- 对操作的描述：算法(algorithm)

著名计算机科学家沃思提出一个公式：

数据结构 + 算法 = 程序

完整的程序设计应该是：

数据结构 + 算法 + 程序设计方法 + 语言工具

2.1 算法的概念

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，可有不同的解题方法和步骤

例： 求 $\sum_{n=1}^{100} n$

- 方法1： 1+2, +3, +4, 一直加到100 加99次
- 方法2： $100+(1+99)+(2+98)+\dots+(49+51)+50$
 $= 100 + 49 \times 100 + 50$ 加51次

2.1 算法的概念

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。希望方法简单，运算步骤少。

计算机算法可分为两大类别：

- **数值运算算法**：求数值解，例如求方程的根、求函数的定积分等。
- **非数值运算**：包括的面十分广泛，最常见的是用于事务管理领域，例如图书检索、人事管理、行车调度管理等。（多数）

2.2 简单算法举例

例2.1: 求 $1 \times 2 \times 3 \times 4 \times 5$

步骤1: 先求 1×2 , 得到结果2

步骤2: 将步骤1得到的乘积2再乘以3, 得到结果6

步骤3: 将6再乘以4, 得24

步骤4: 将24再乘以5, 得120

如果要求 $1 \times 2 \times \cdots \times 1000$, 则要写999个步骤

太繁琐

可以设两个变量：一个变量代表被乘数，一个变量代表乘数。不另设变量存放乘积结果，而直接将每一步骤的乘积放在被乘数变量中。设 p 为被乘数， i 为乘数。用循环算法来求结果，算法可改写：

S1：使 $p=1$ 。

S2：使 $i=2$ 。

S3：使 $p \times i$ ，乘积仍放在变量 p 中，可表示为： $p \times i$

S4：使 i 的值加1，即 $i+1$ 。

S5：如果 i 不大于5，返回重新执行步骤S3以及其后的步骤S4和S5；否则，算法结束。最后得到 p 的值就是 $5!$ 的值。

如果题目改为：求 $1 \times 3 \times 5 \times \cdots \times 11$ 算法只需作很少的
改动：

算法简练

S1: $1 \rightarrow p$

S2: $3 \rightarrow i$

S3: $p \times i \rightarrow p$

S4: $i+2 \rightarrow i$

S5: 若 $i \leq 11$ ，返回S3。否则，结束。

用这种方法表示的算法具有通用性、灵活性。S3到S5组成一个循环，在实现算法时要反复多次执行S3，S4，S5等步骤，直到某一时刻，执行S5步骤时经过判断，乘数*i*已超过规定的数值而不返回S3步骤为止。此时算法结束，变量*p*的值就是所求结果。

S1: $1 \rightarrow p$

S2: $3 \rightarrow i$

S3: $p \times i \rightarrow p$

S4: $i+2 \rightarrow i$

S5: 若 $i \leq 11$ ，返回S3。否则，结束。

例2.2 有50个学生，要求将他们之中成绩在80分以上者打印出来。设 n 表示学号， n_1 代表第一个学生学号， n_i 代表第 i 个学生学号。用 g 代表学生成绩， g_i 代表第 i 个学生成绩，算法表示如下：

S1: $1 \rightarrow i$

S2: 如果 $g_i \geq 80$ ，则打印，否则不打印。

S3: $i+1 \rightarrow i$

S4: 如果 $i \leq 50$ ，返回S2，继续执行。否则算法结束

变量 i 作为下标，用来控制序号(第几个学生，第几个成绩)。当 i 超过50时，表示 已对50个学生的成绩处理完毕，算法结束。

例2.3 判定2000~2500年中的每一年是否闰年，将结果输出。

分析：

闰年的条件是：

- (1) 能被4整除，但不能被100整除的年份都是闰年，如1996, 2004年是闰年；
- (2) 能被100整除，又能被400整除的年份是闰年。如1600, 2000年是闰年。

不符合这两个条件的年份不是闰年。

设 y 为被检测的年份，算法可表示如下：

S1: $2000 \rightarrow y$

S2: 若 y 不能被4整除，则输出 y “不是闰年”。然后转到S6。

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

S4: 若 y 能被100整除，又能被400整除，输出 y “是闰年”，否则输出 “不是闰年”。然后转到S6。

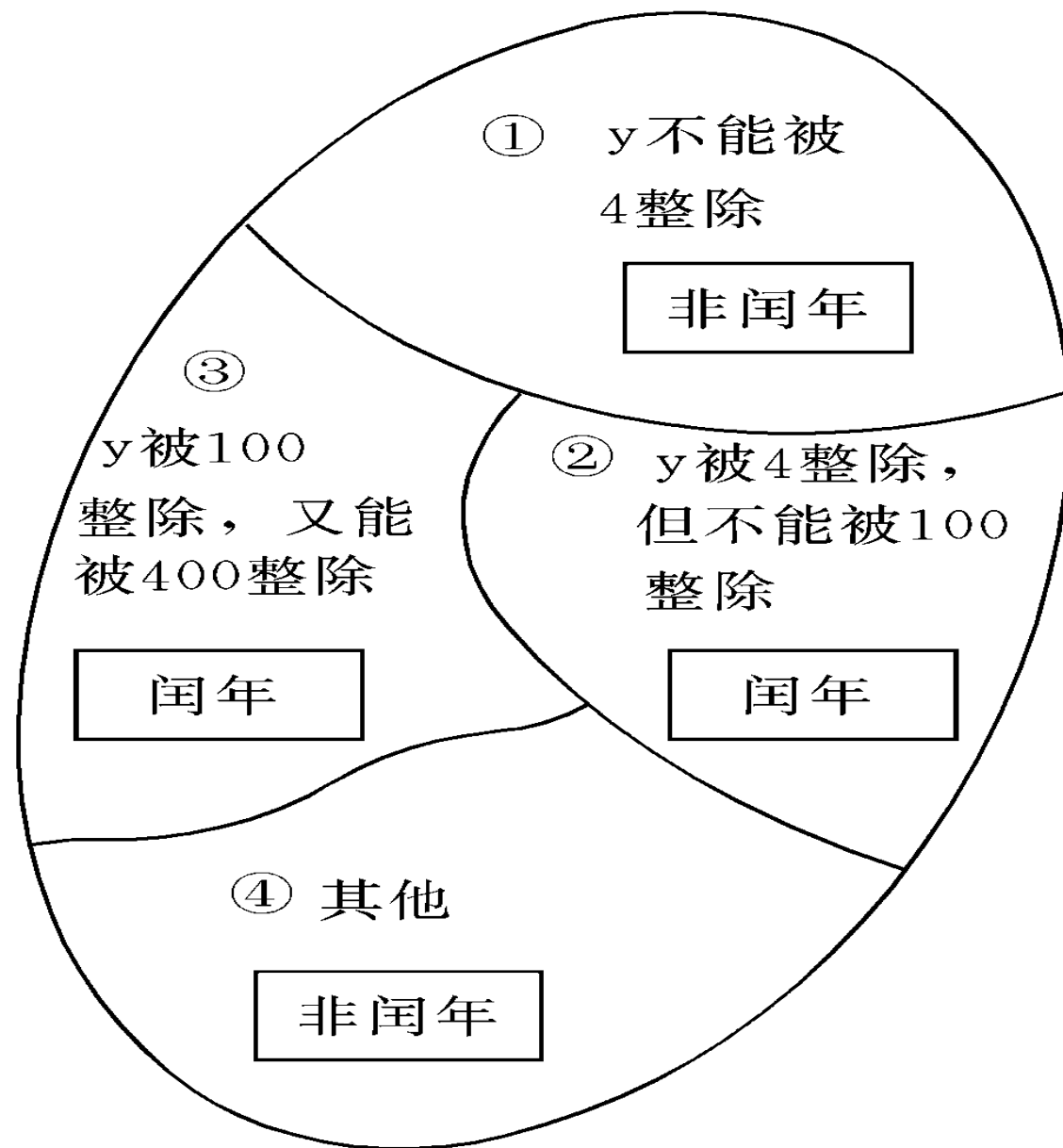
S5: 输出 y “不是闰年”。

S6: $y+1 \rightarrow y$

S7: 当 $y \leq 2500$ 时，转S2继续执行，如 $y > 2500$ ，算法停止。

◆ 以上算法中每做一步都分别分离出一些范围(已能判定为闰年或非闰年), 逐步缩小范围, 直至执行S5时, 只可能是非闰年。

◆ “其它” 包括能被4整除, 又能被100整除, 而不能被400整除的那些年份(如1990) 是非闰年。



例2.4 求

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

算法如下：

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。

单词作变量名，以使算法更易于理解：

sum表示累加和，deno是英文分母（denominator）缩写，sign代表数值的符号，term代表某一项。

反复执行S4到S8步骤，直到分母大于100为止。一共执行了99次循环，向sum累加入了99个分数。sum最后的值就是多项式的值。

例2.5 对一个大于或等于3的正整数，判断它是不是一个素数。

概念：所谓素数，是指除了1和该数本身之外，不能被其它任何整数整除的数。例如，13是素数。因为它不能被2，3，4，…，12整除。

分析：判断一个数 n ($n \geq 3$) 是否素数的方法：

将 n 作为被除数，将2到 $(n-1)$ 各个整数轮流作为除数，如果都不能被整除，则 n 为素数。

算法如下：

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，则打印 n “不是素数”，算法结束。否则执行S5

S5: $i+1 \rightarrow i$

S6: 如果 $i \leq n-1$ ，返回S3。否则打印 n “是素数”。然后结束。

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

2.3 算法的特性

一个算法应该具有以下特点：

- 有穷性：包含有限的操作步骤。
- 确定性：算法中的每一个步骤都应当是确定的。
- 有零个或多个输入：输入是指在执行算法时需要从外界取得必要的信息。
- 有一个或多个输出：算法的目的是为了求解，“解”就是输出。
- 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。

2.4 算法的表示

可以用不同的方法表示算法，常用的有：

- 自然语言
- 传统流程图
- 结构化流程图
- 伪代码
- PAD图

2.4.1 用自然语言表示算法

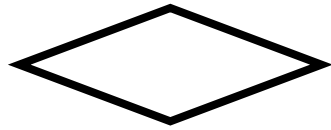
自然语言就是人们日常使用的语言，可以是汉语或英语或其它语言。用自然语言表示通俗易懂，但文字冗长，容易出现“歧义性”。自然语言表示的含义往往不大严格，要根据上下文才能判断其正确含义，描述包含分支和循环的算法时也不很方便。因此，除了那些很简单的问题外，一般不用自然语言描述算法。

2.4.2 用流程图表示算法

美国国家标准化协会ANSI(American National Standard Institute)规定了一些常用的流程图符号：



起止框



判断框



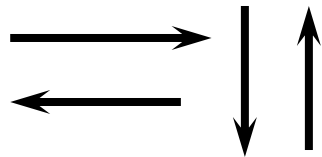
处理框



输入/输出框



注释框

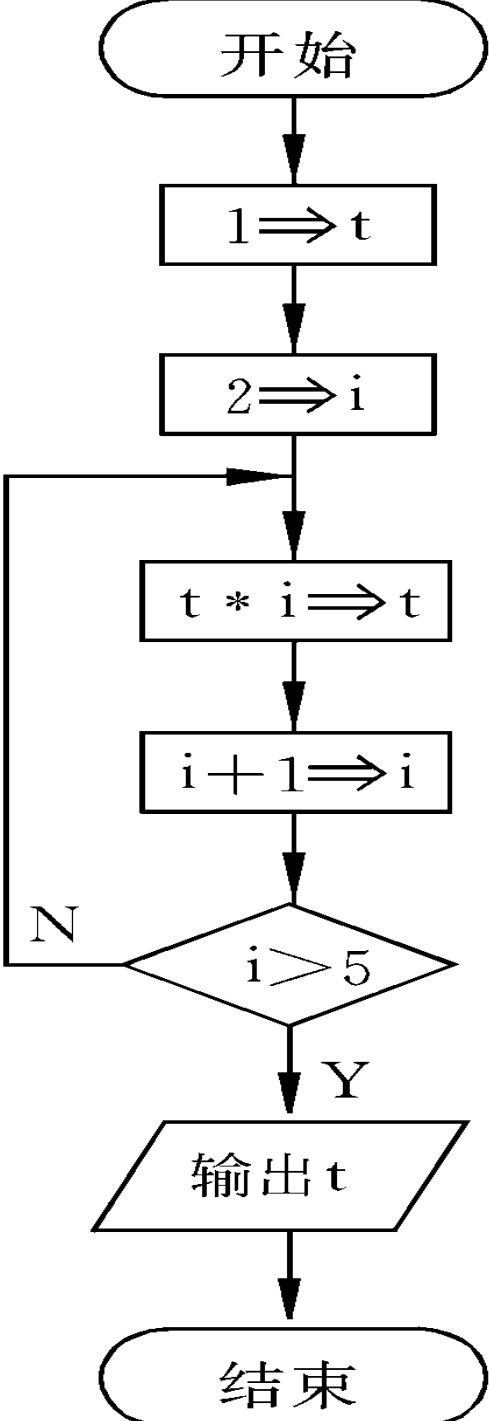
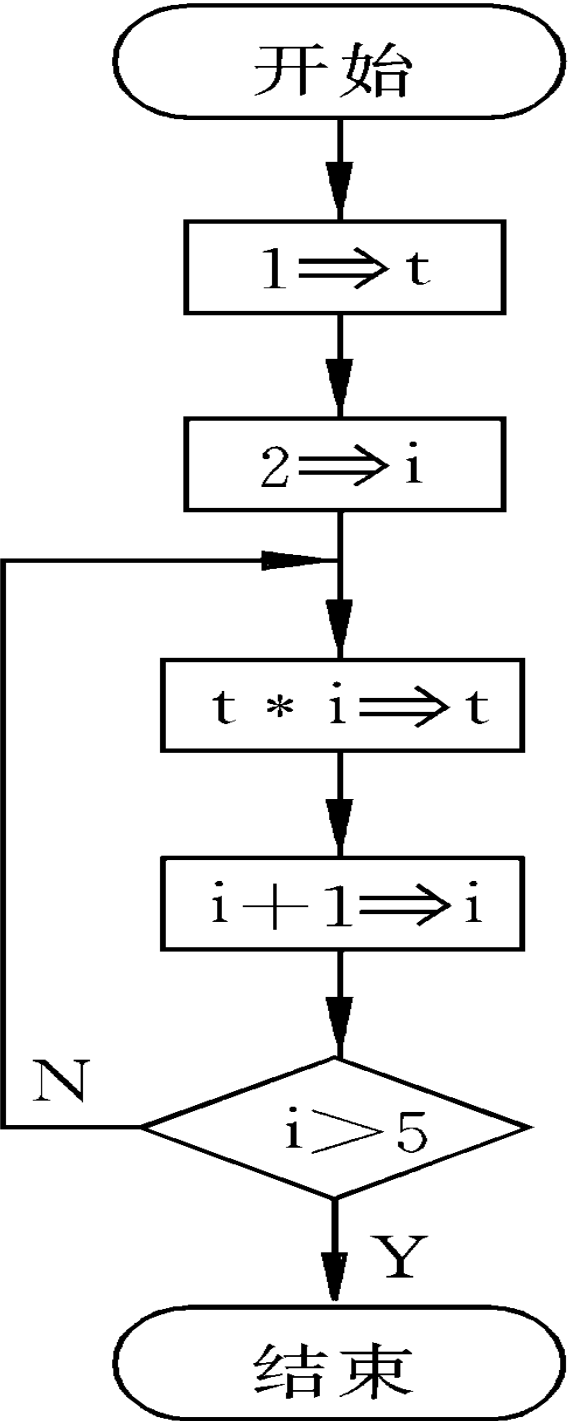


流向线

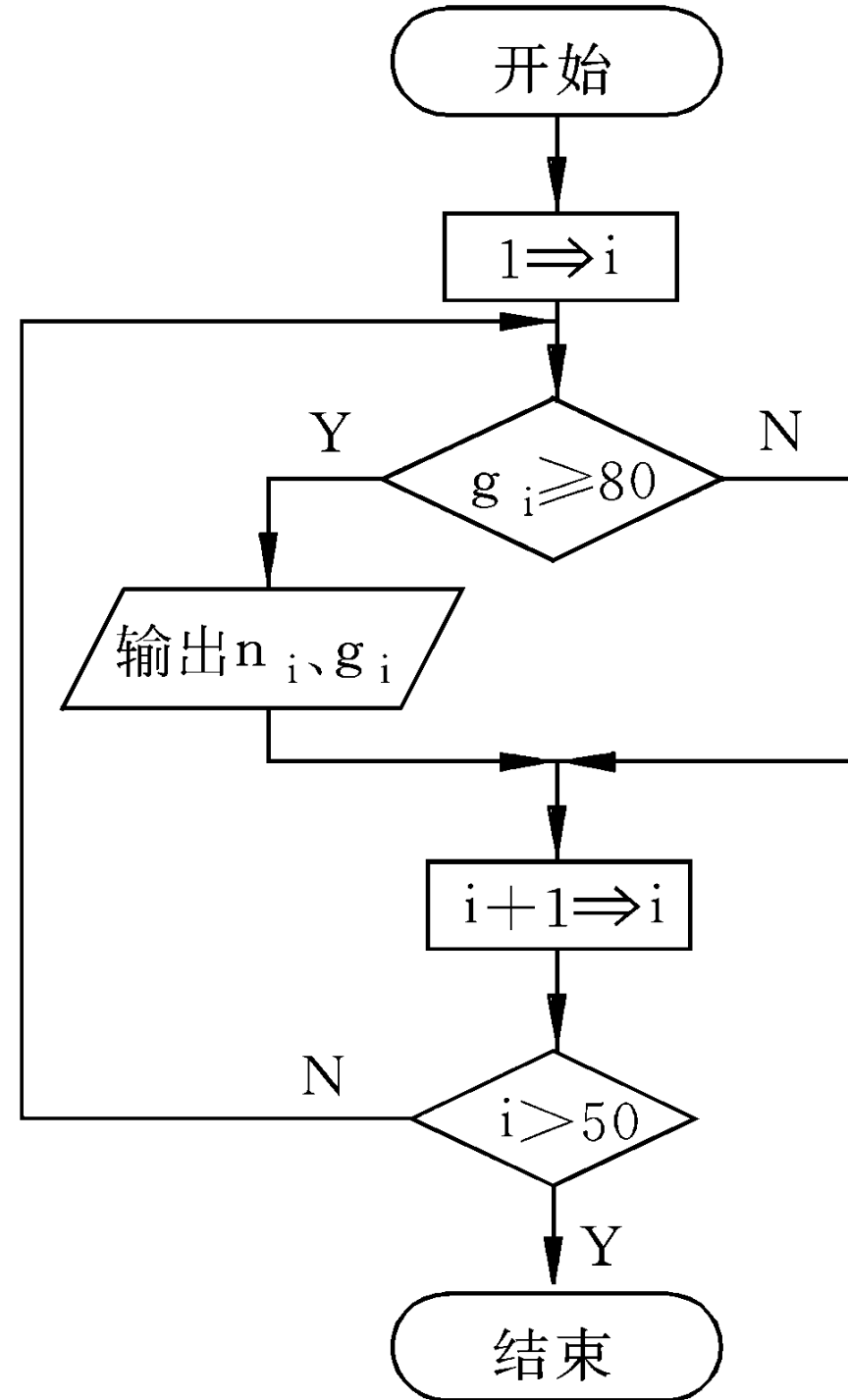
连接点

例2.6 将求5!的算法用流程图表示

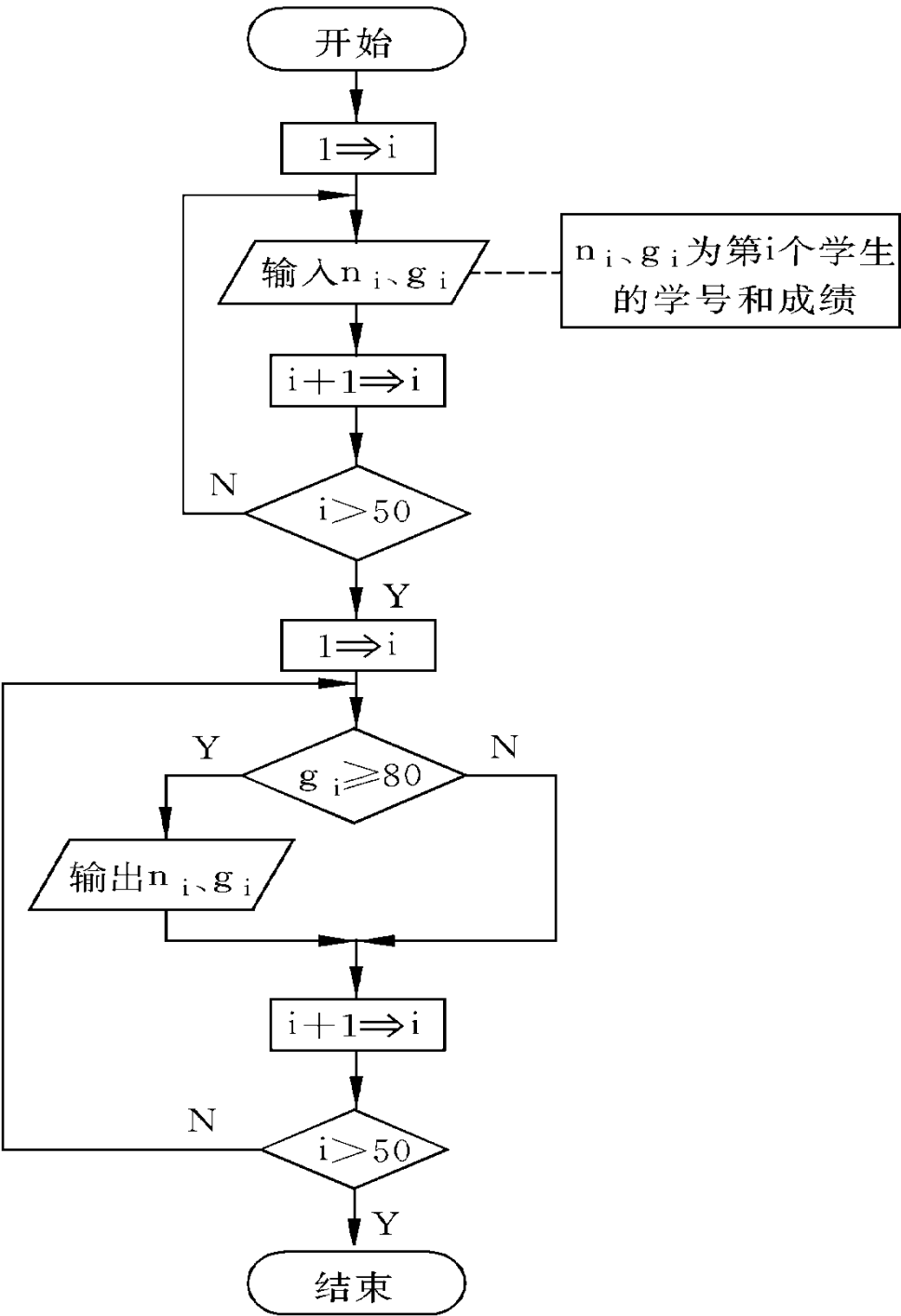
如果需要将最后结果打印出来，可在菱形框的下面加一个输出框。



例2.7 将例2.2的算法用流程图表示。打印50名学生中成绩在80分以上者的学号和成绩。

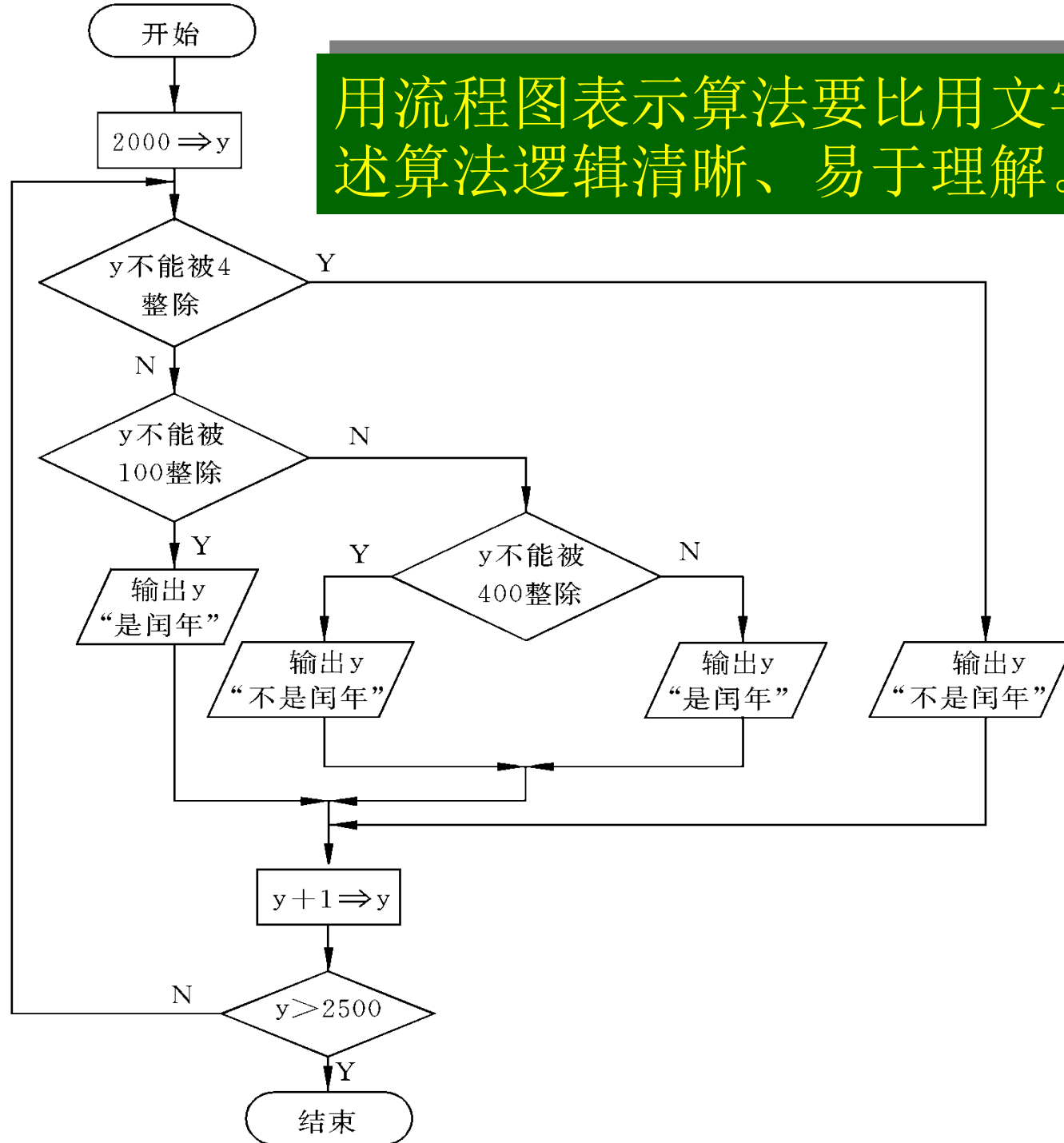


如果如果包括这个输入数据的部分，
流程图为



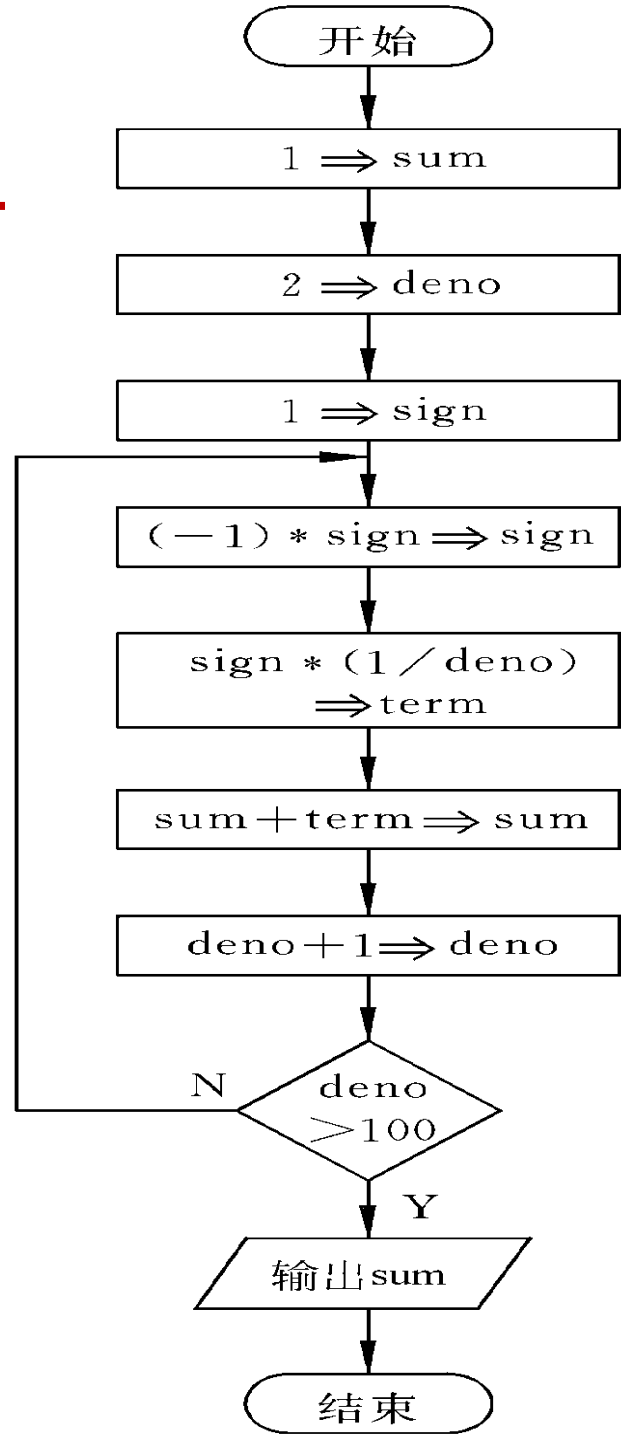
例2.8 将例2.3判定闰年的算法用流程图表示

用流程图表示算法要比用文字描述算法逻辑清晰、易于理解。

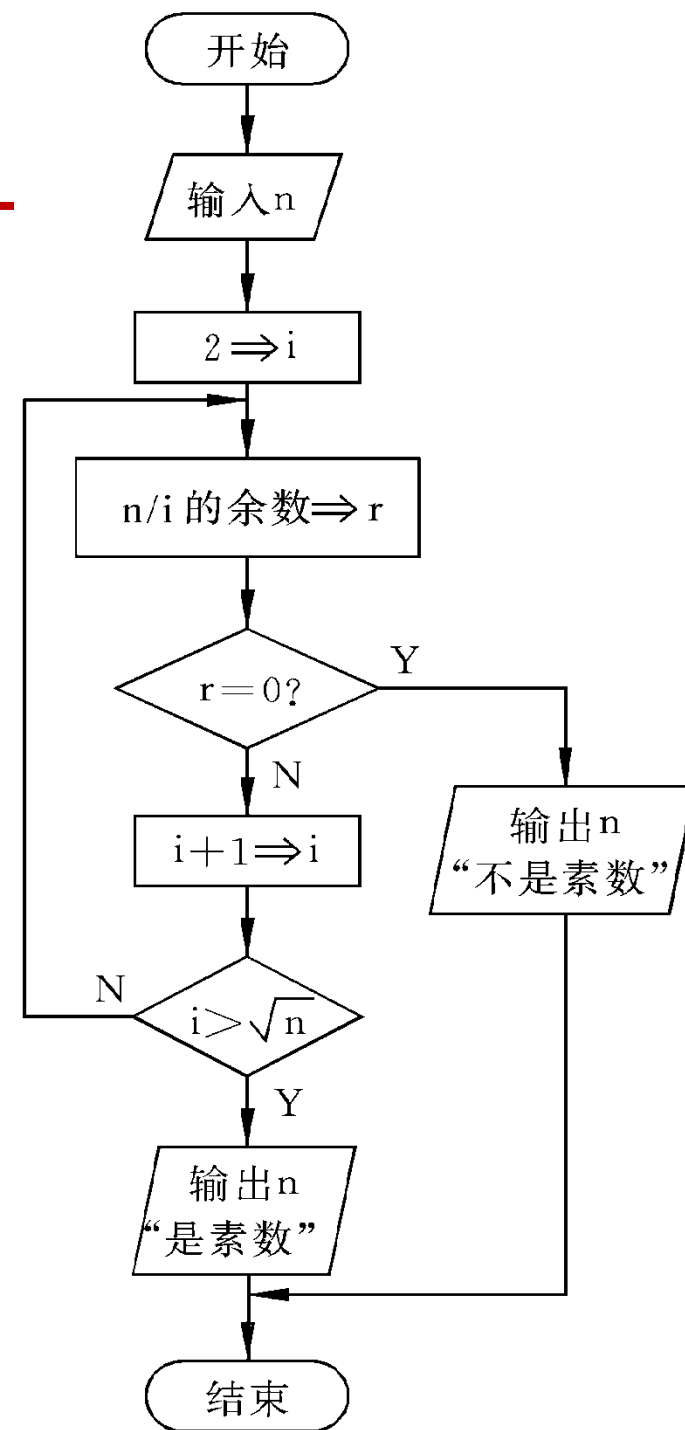


例2.9 将例2.4的算法用流程图表示

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



例2.10 将例2.5判断素数的算法用流程图表示



小结：

- 流程图是表示算法的较好的工具。一个流程图包括以下几部分：
 - (1) 表示相应操作的框；
 - (2) 带箭头的流程线；
 - (3) 框内外必要的文字说明。

2.4.3 三种基本结构和改进的流程图

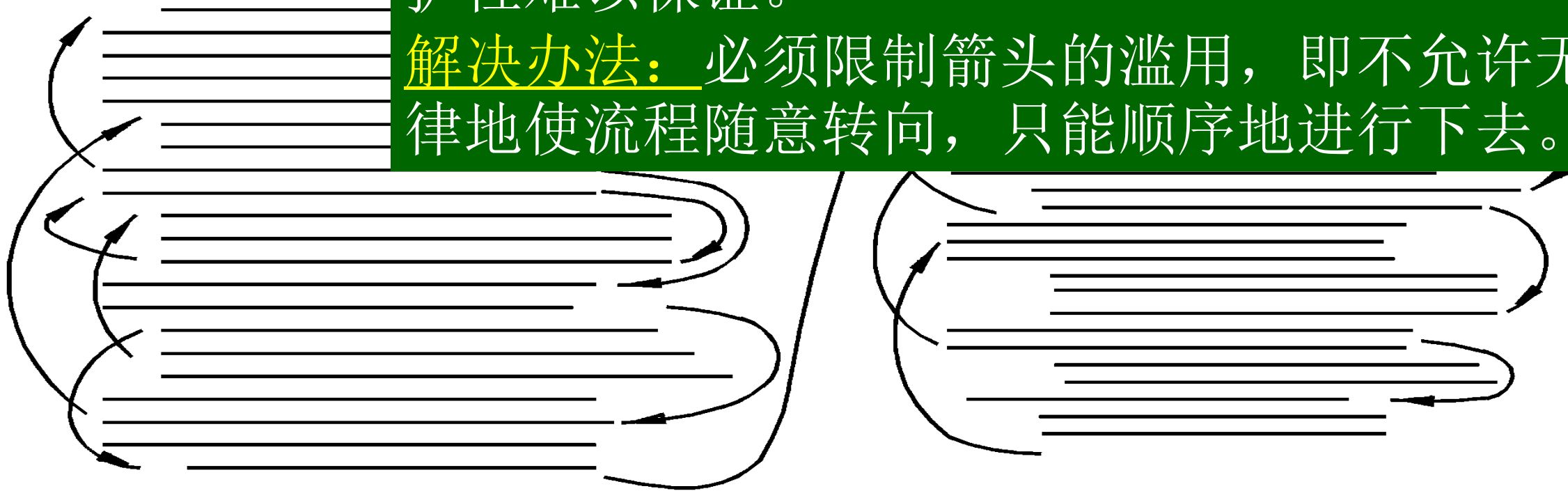
1. 传统流程图的弊端

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，使用者可以毫不受限制地使流程随意地转向，使流程图变得毫无规律，阅读者要花很大精力去追踪流程，使人难以理解算法的逻辑。如图：

传统流程图的流

缺点：难以阅读、修改，使算法的可靠性和可维护性难以保证。

解决办法：必须限制箭头的滥用，即不允许无规律地使流程随意转向，只能顺序地进行下去。



这种如同乱麻一样的算法称为BS型算法，意为一碗面条 (A Bowl of Spaghetti)，乱无头绪。

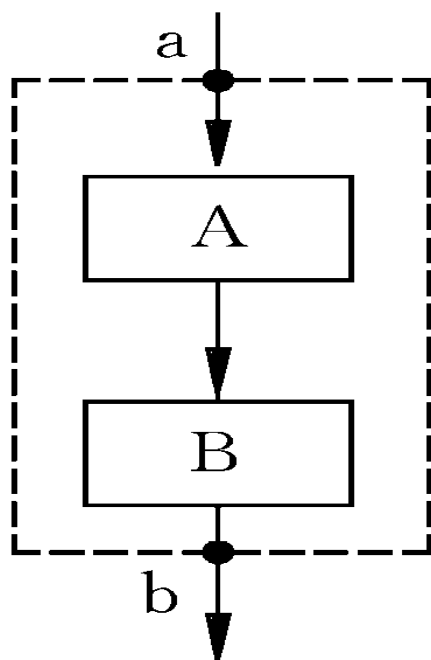
2. 三种基本结构

Bohra和Jacopini提出了以下三种基本结构：

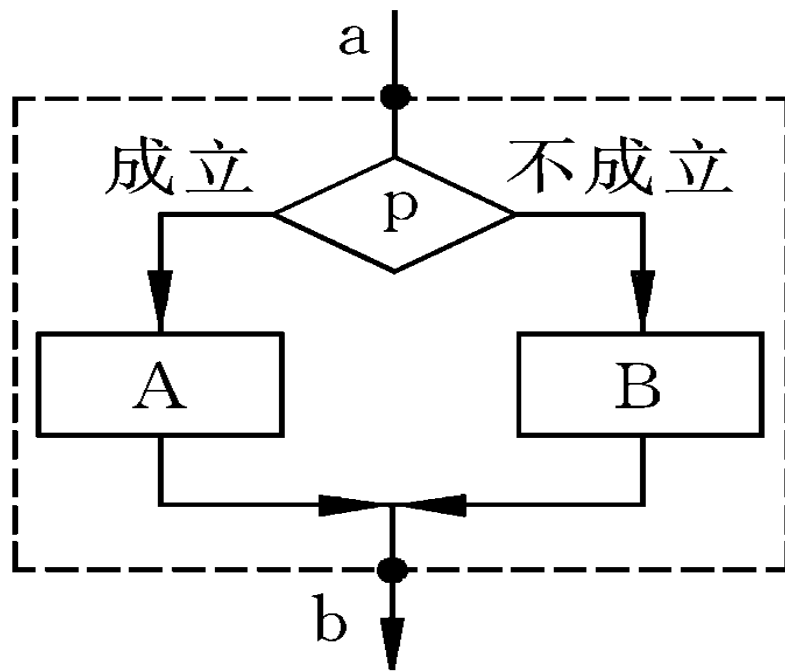
顺序结构、选择结构、循环结构

用这三种基本结构作为表示一个良好算法的基本单元。

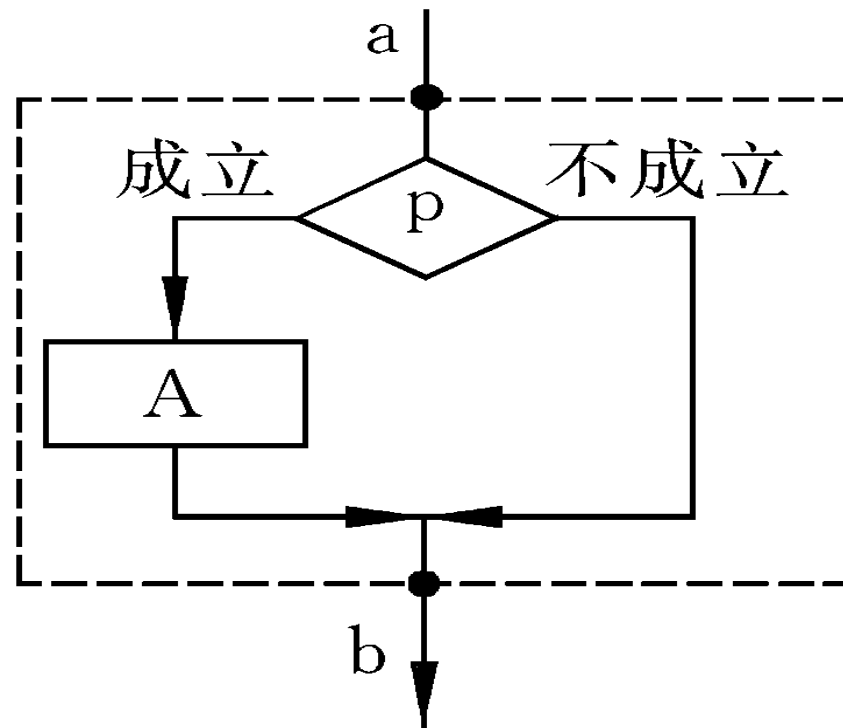
三种基本结构的图示：



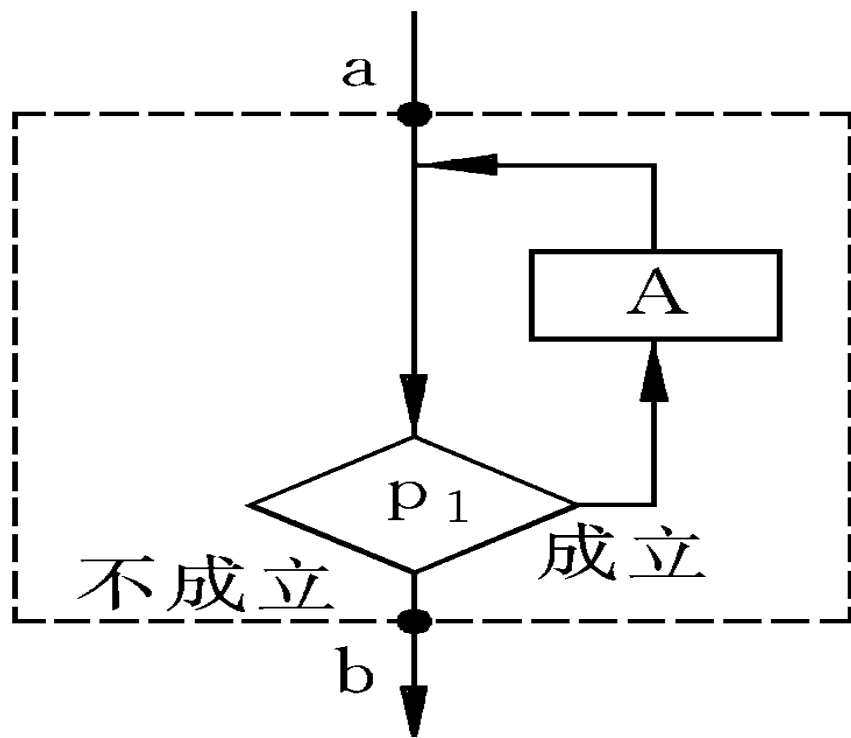
顺序结构



选择结构

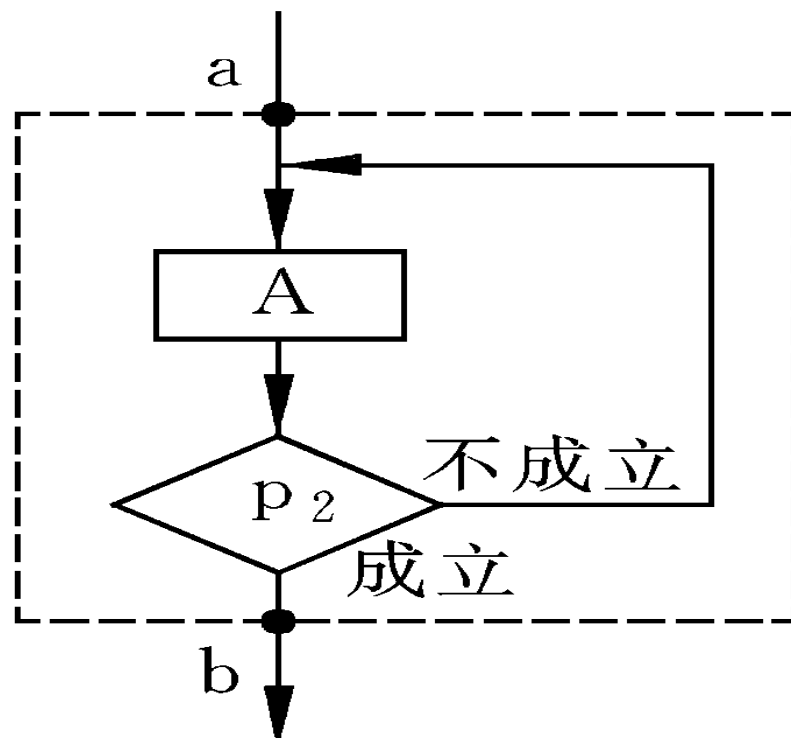


循环结构的图示：



(a)

当型(While型)循环结构



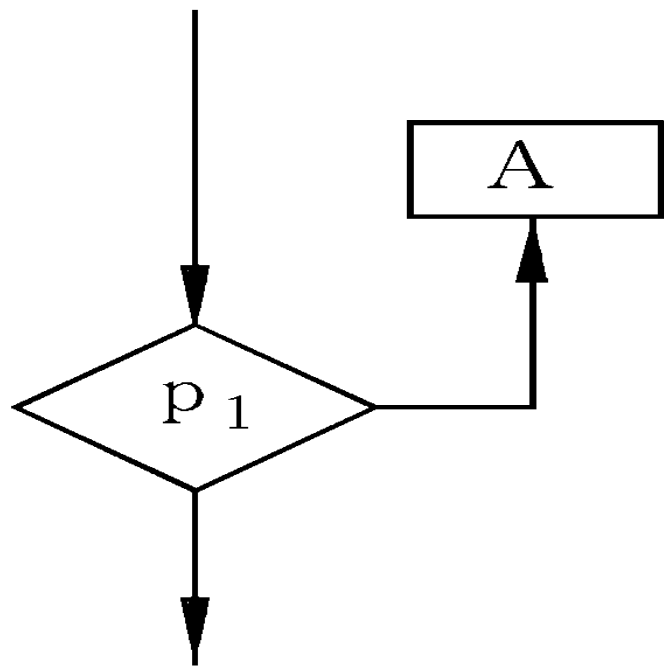
(b)

直到型(Until型)循环

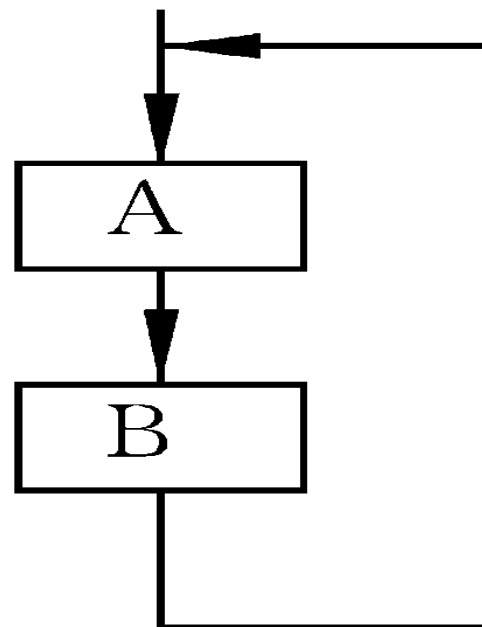
三种基本结构的共同特点：

- (1) 只有一个入口。
- (2) 只有一个出口。（请注意：一个菱形判断框有两个出口，而一个选择结构只有一个出口。不要将菱形框的出口和选择结构的出口混淆。）
- (3) 结构内的每一部分都有机会被执行到。
- (4) 结构内不存在“死循环”（无终止的循环）。

不正确的流程表示:



图中没有一条从入口到出口的路径通过A框



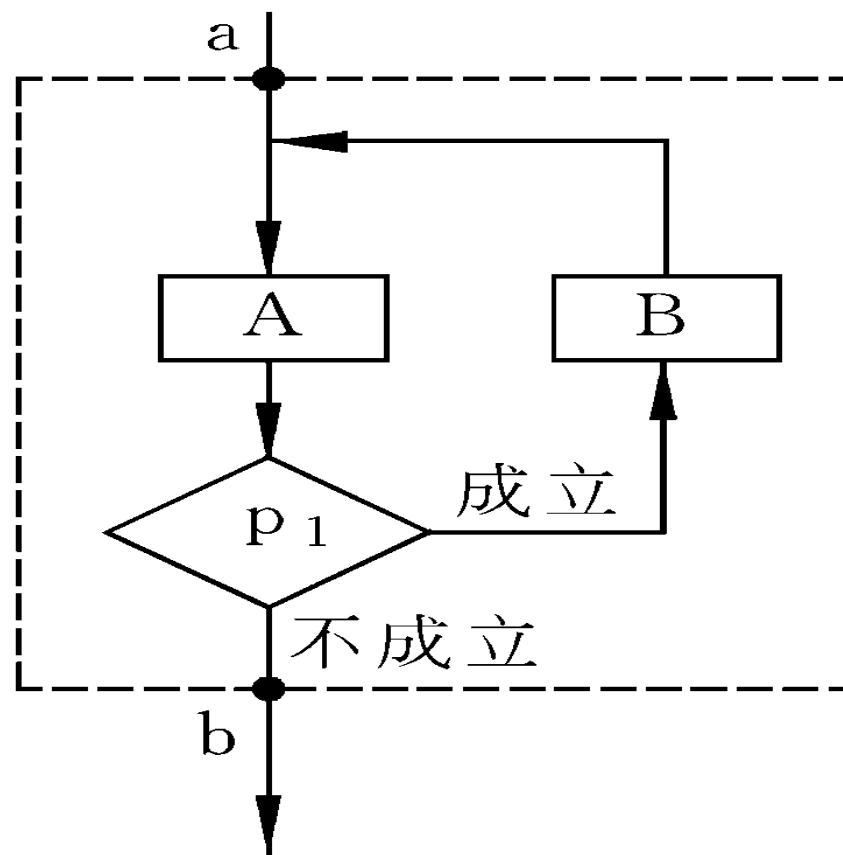
流程内的死循环

小结：

- 由三种基本结构顺序组成的算法结构，可以解决任何复杂的问题。
- 由基本结构所构成的算法属于“结构化”的算法，它不存在无规律的转向，只在本基本结构内才允许存在分支和向前或向后的跳转。

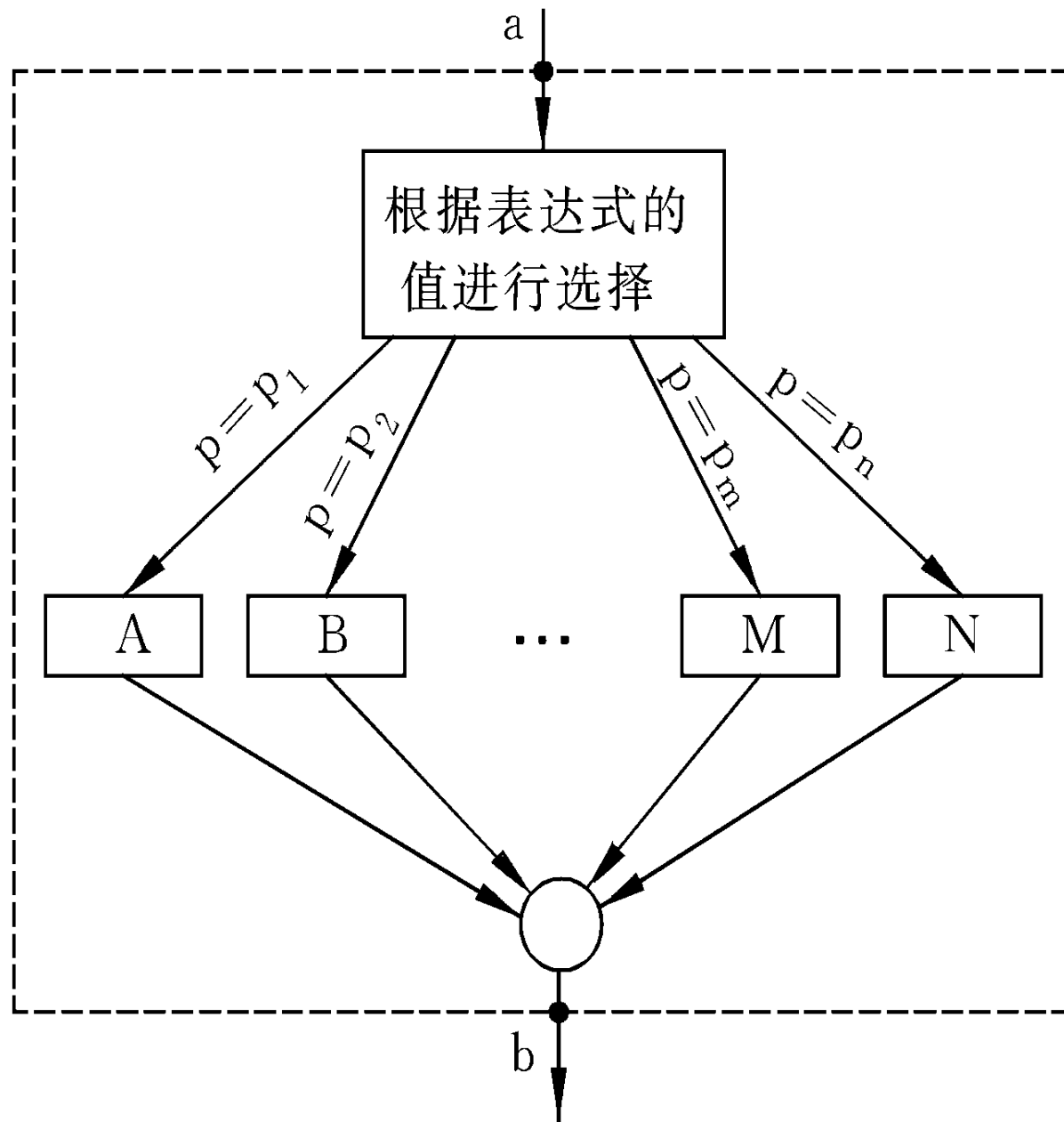
扩展:

- 只要具有上述四个特点的都可以作为基本结构。可以自己定义基本结构，并由这些基本结构组成结构化程序。



此图符合基本结构的特点

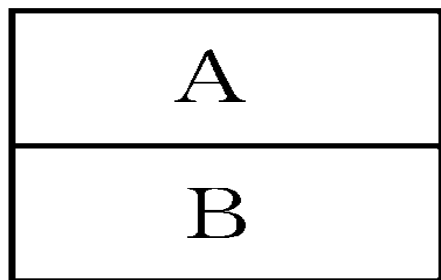
- ✓ 这是一个多分支选择结构，根据表达式的值决定执行路线。虚线框内的结构是一个入口一个出口，并且有上述全部四个特点。
- ✓ 由此构成的算法结构也是结构化的算法。
- ✓ 可以认为这是由三种基本结构所派生出来的。



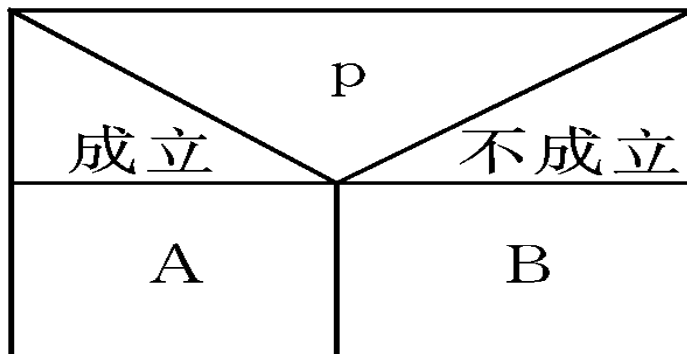
2.4.4 用N-S流程图表示算法

1973年美国学者I. Nassi和B. Shneiderman提出了一种新的流程图形式。在这种流程图中，完全去掉了带箭头的流程线。全部算法写在一个矩形框内，在该框内还可以包含其它的从属于它的框，或者说，由一些基本的框组成一个大的框。这种流程图又称N--S结构化流程图。

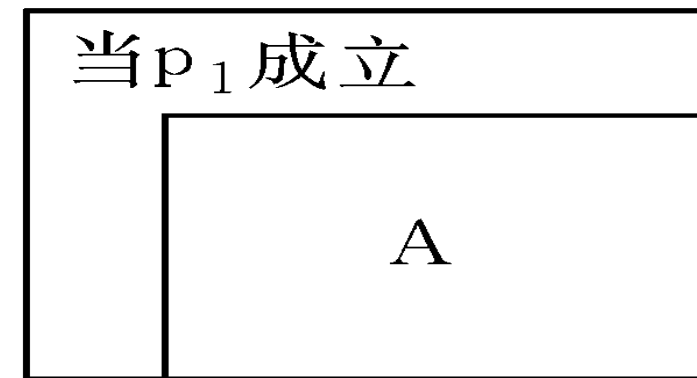
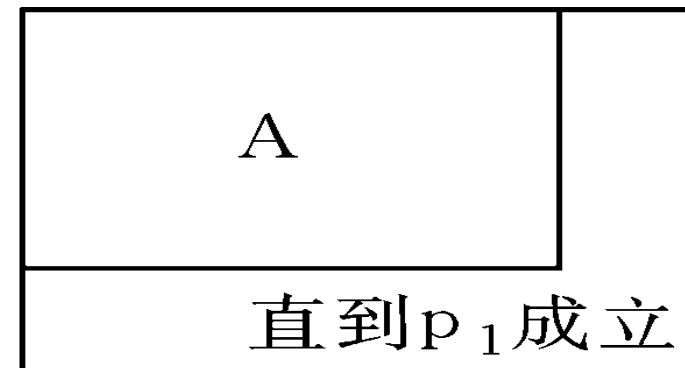
N-S流程图用以下的流程图符号：



(1)顺序结构

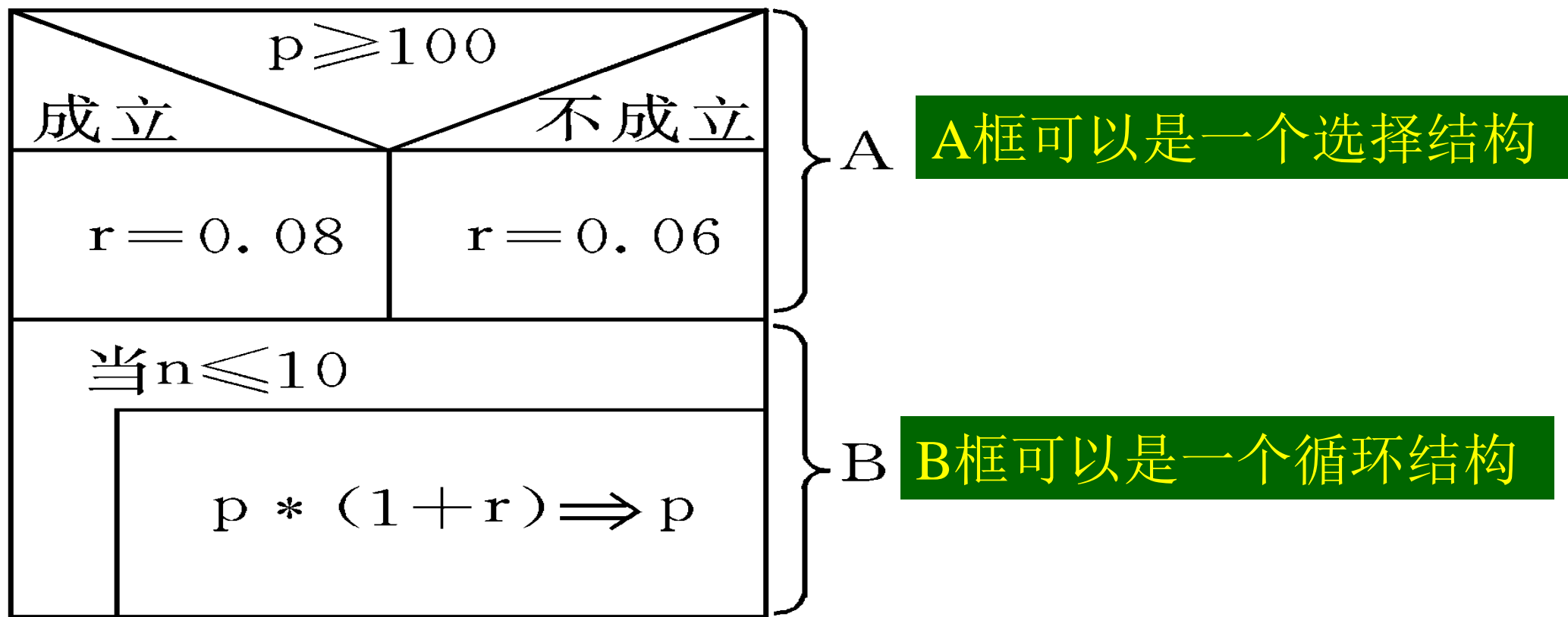


(2)选择结构

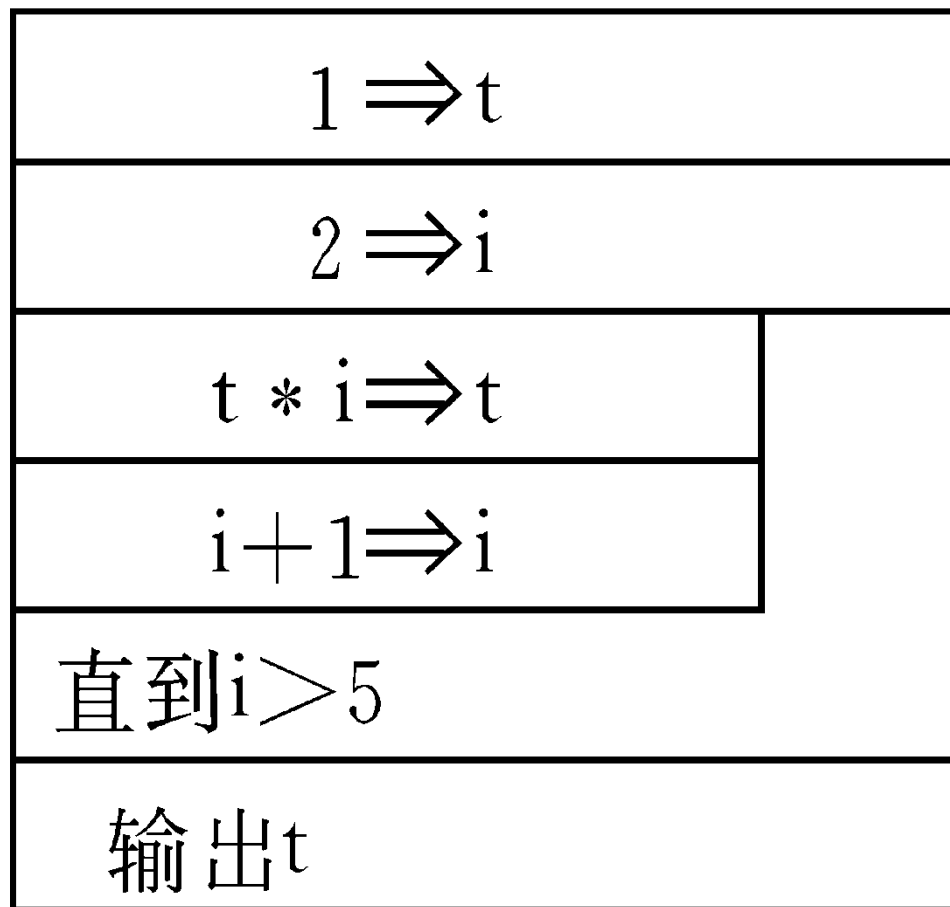


(3)循环结构

◆用三种N-S流程图中的基本框，可以组成复杂的N-S流程图。图中的A框或B框，可以是一个简单的操作，也可以是三个基本结构之一。

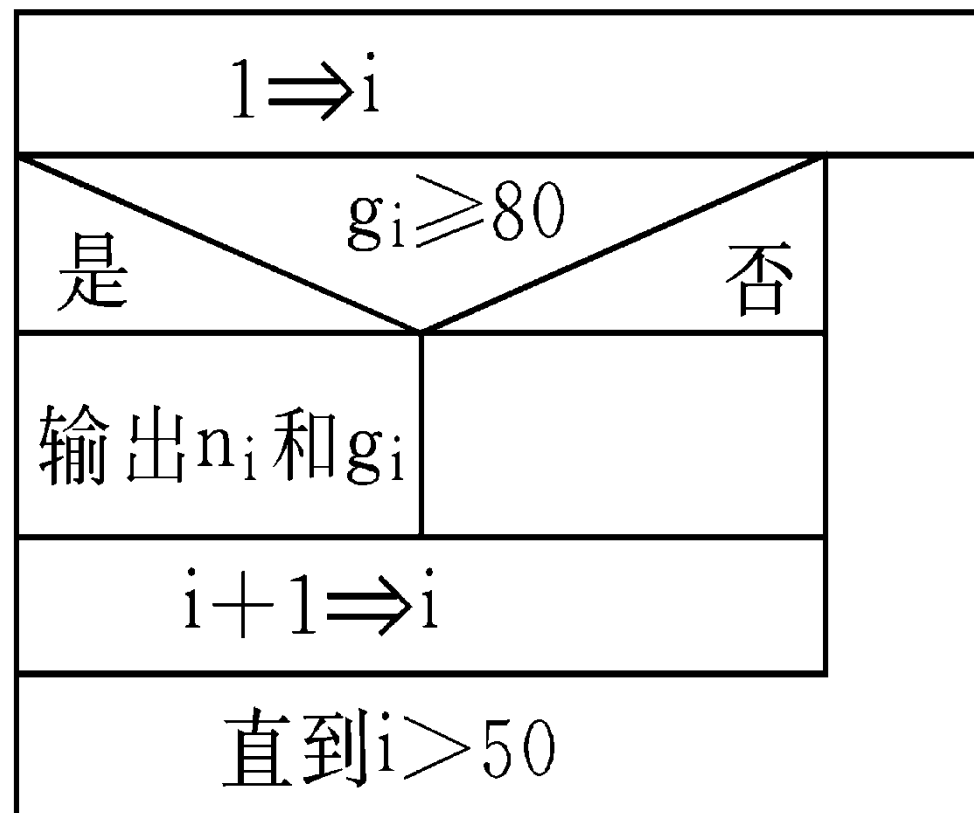


例2.11 将例2.1的求
5!算法用N-S图表示



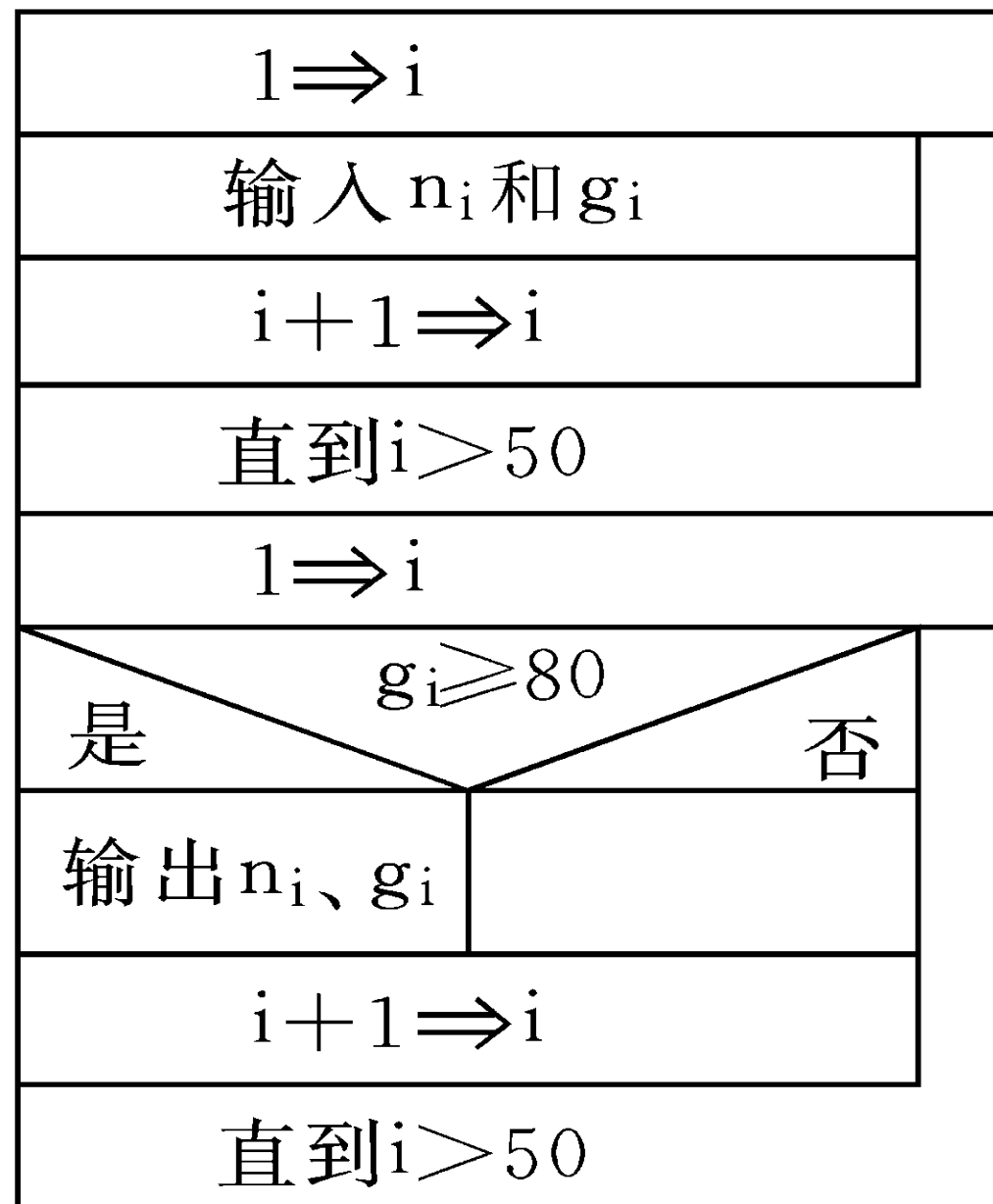
例2.12 将例2.2的算法用N-S图表示。
(打印50名学生中成绩高于80分的学号和成绩)

没有输入数据

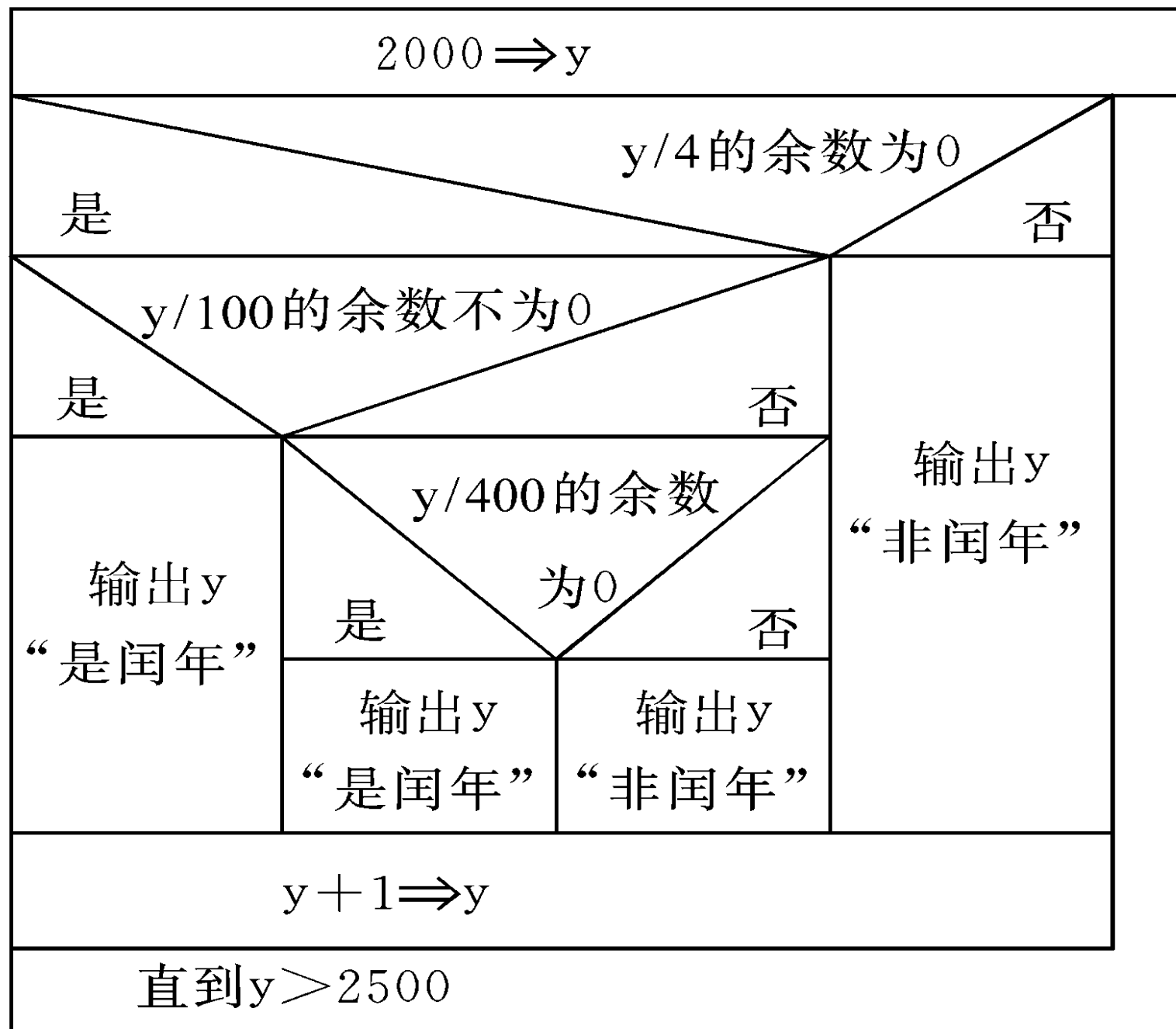


例2.12 将例2.2的算法用N-S图表示。
(打印50名学生中成绩高于80分的学号和成绩)

有输入数据



例2.13
将例2.3判定闰
年的算法用N-S
图表示



例2.14 将例2.4的算法用N-S图表示

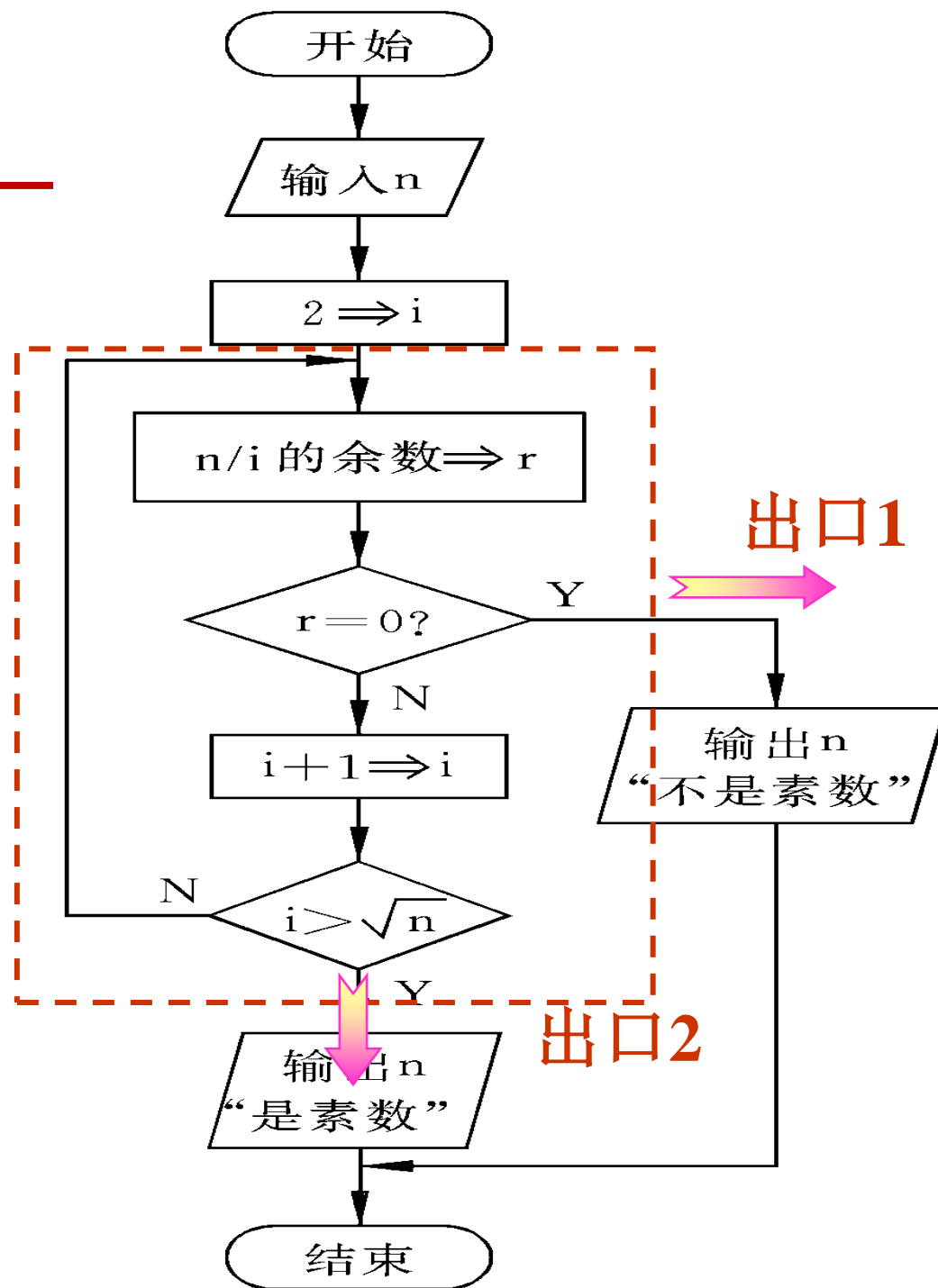
$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots - \frac{1}{99} + \frac{1}{100}$$

$1 \Rightarrow \text{sum}$	
$2 \Rightarrow \text{deno}$	
$1 \Rightarrow \text{sign}$	
$(-1) * \text{sign} \Rightarrow \text{sign}$	
$\text{sign} * 1 / \text{deno} \Rightarrow \text{term}$	
$\text{sum} + \text{term} \Rightarrow \text{sum}$	
$\text{deno} + 1 \Rightarrow \text{deno}$	
直到 $\text{deno} > 100$	
输出 sum	

例2.15 将例2.5判别素数的算法用N-S流程图表示。

传统流程图分析：

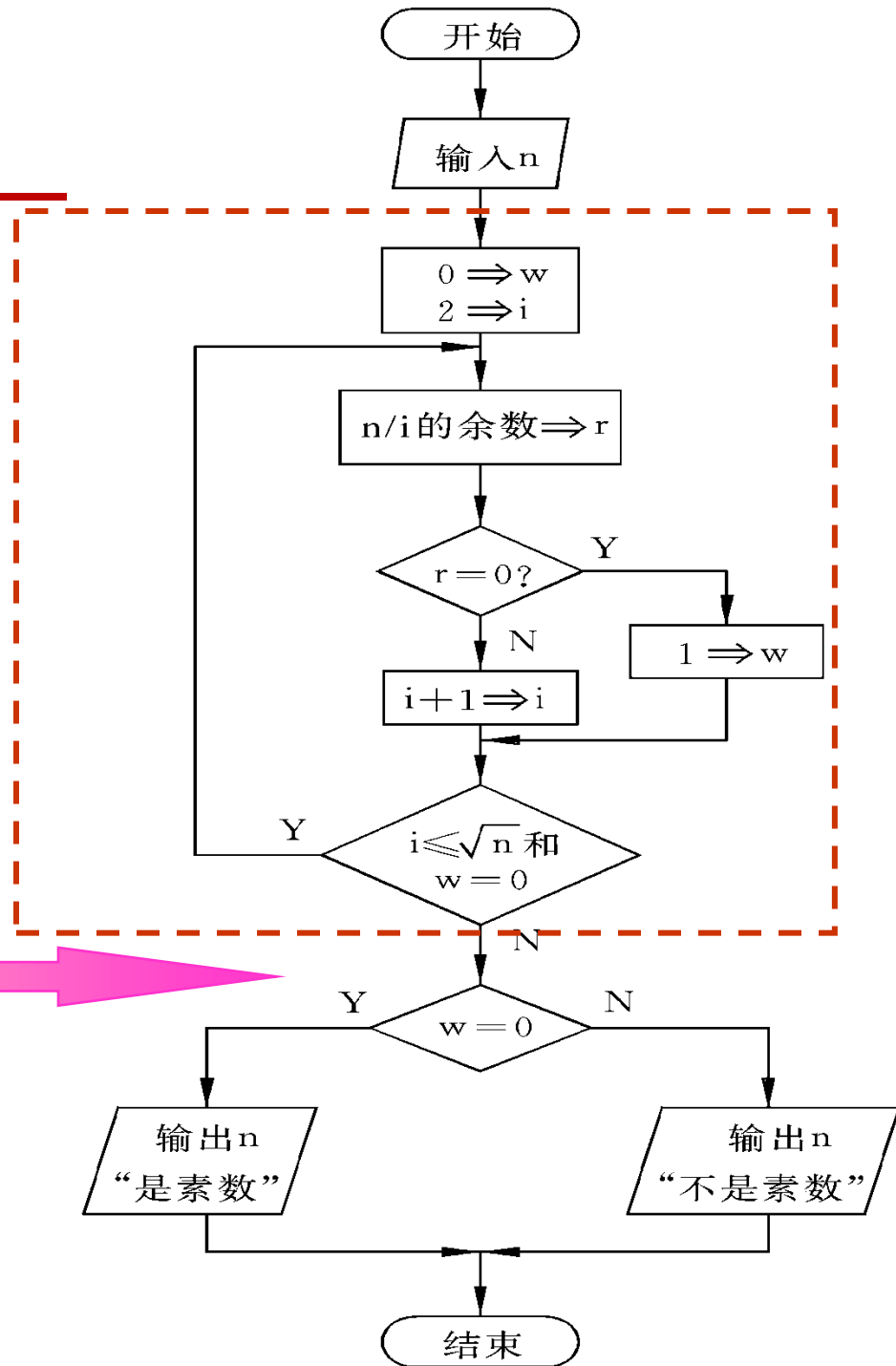
此图不符合基本结构特点！由于不能分解为三种基本结构，就无法直接用N--S流程图的三种基本结构的符号来表示。因此，应当先作必要的变换。



例2.15 将例2.5判别素数的
算法用N-S流程图表示。

传统流程图变换为：

一个出口



用N-S流程图表示：

输入n	
$0 \Rightarrow w$	
$2 \Rightarrow i$	
n / i 的余数 $\Rightarrow r$	
$r = 0$	
是	否
$1 \Rightarrow w$	$i + 1 \Rightarrow i$
直到 $i > \sqrt{n}$ 或 $w \neq 0$	
$w = 0$	
是	否
输出n“是素数”	输出n“不是素数”

N-S图表示算法的优点

- 比文字描述直观、形象、易于理解；比传统流程图紧凑易画。尤其是它废除了流程线，整个算法结构是由各个基本结构按顺序组成的，N-S流程图中的上下顺序就是执行时的顺序。用N-S图表示的算法都是结构化的算法，因为它不可能出现流程无规律的跳转，而只能自上而下地顺序执行。

小结

- 一个结构化的算法是由一些基本结构顺序组成的。在基本结构之间不存在向前或向后的跳转，流程的转移只存在于一个基本结构范围之内(如循环中流程的跳转)；一个非结构化的算法可以用一个等价的结构化算法代替，其功能不变。如果一个算法不能分解为若干个基本结构，则它必然不是一个结构化的算法。

2.4.5 用伪代码表示算法

- 概念：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。
- 特点：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，也比较好懂，也便于向计算机语言算法(即程序)过渡。
- 用处：适用于设计过程中需要反复修改时的流程描述。

例：“打印 x 的绝对值”的算法可以用伪代码表示为：

IF x is positive THEN

print x

ELSE

print $-x$

也可以用汉字伪代码表示：

若 x 为正

打印 x

否则

打印 $-x$

也可以中英文混用，如：

IF x 为正

print x

ELSE

print $-x$

例2.16 求5!。用伪代码表示算法：

也可以写成以下形式：

BEGIN {算法开始}

1→t

2 → i

while i≤5

{t×i →t

i+1 → i }

print t

END {算法结束}

开始

置t的初值为1

置i的初值为2

当i≤5，执行下面操作：

使t=t×i

使i=i+1

{循环体到此结束}

输出t的值

结束

例2.17 输出50个学生中成绩
高于80分者的学号和成绩。

用伪代码表示算法：

```
BEGIN { 算法开始 }  
    1 → i  
    while i ≤ 50  
        { input ni and gi  
          i+1 → i }  
    1 → i  
    while i ≤ 50  
        { if gi ≥ 80 print ni and gi  
          i+1 → i }  
    END { 算法结束 }
```


2.4.6 用计算机语言表示算法

- 概念：用计算机实现算法。计算机是无法识别流程图和伪代码的。只有用计算机语言编写的程序才能被计算机执行。因此在用流程图或伪代码描述出一个算法后，还要将它转换成计算机语言程序。
- 特点：用计算机语言表示算法必须严格遵循所用的语言的语法规则，这是和伪代码不同的。
- 用处：要完成一件工作，包括设计算法和实现算法两个部分。设计算法的目的是为了实现算法。

例 2.20 将例2.16表示的算法
(求5!) 用C语言表示。

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

-
- 应当强调说明：写出了C程序，仍然只是描述了算法，并未实现算法。只有运行程序才是实现算法。应该说，用计算机语言表示的算法是计算机能够执行的算法。

2.5 结构化程序设计方法

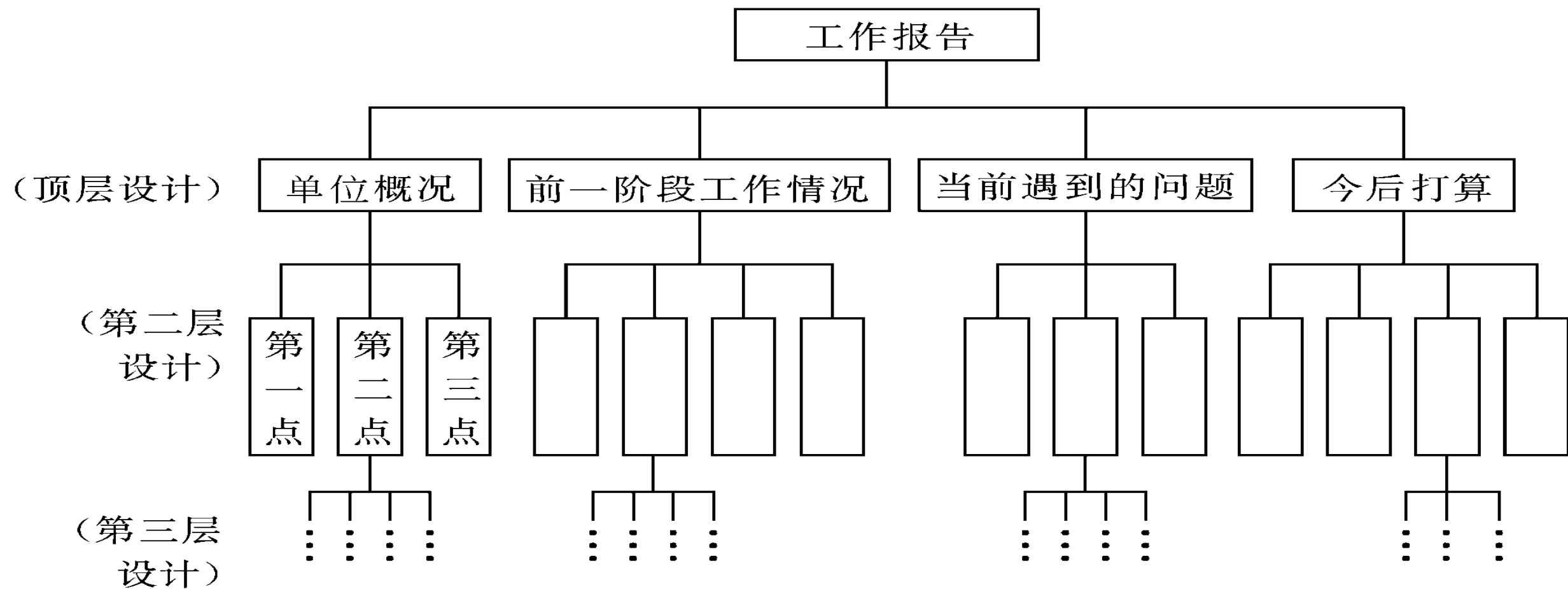
- 一个结构化程序就是用高级语言表示的结构化算法。用三种基本结构组成的程序必然是结构化的程序，这种程序便于编写、便于阅读、便于修改和维护。
- 结构化程序设计强调程序设计风格和程序结构的规范化，提倡清晰的结构。
- 结构化程序设计方法的基本思路是：把一个复杂问题的求解过程分阶段进行，每个阶段处理的问题都控制在人们容易理解和处理的范围内。

采取以下方法来保证得到结构化的程序：

- 自顶向下；
- 逐步细化；
- 模块化设计；
- 结构化编码。

两种不同的方法：

- 自顶向下，逐步细化；
- 自下而上，逐步积累。



用这种方法逐步分解，直到作者认为可以直接将各小段表达为文字语句为止。这种方法就叫做“自顶向下，逐步细化”。

自顶向下，逐步细化方法的优点：

考虑周全，结构清晰，层次分明，作者容易写，读者容易看。如果发现某一部分中有一段内容不妥，需要修改，只需找出该部分修改有关段落即可，与其它部分无关。我们提倡用这种方法设计程序。这就是用工程的方法设计程序。

模块设计的方法：

- 模块化设计的思想实际上是一种“分而治之”的思想，把一个大任务分为若干个子任务，每一个子任务就相对简单了。
- 在拿到一个程序模块以后，根据程序模块的功能将它划分为若干个子模块，如果这些子模块的规模还嫌大，还再可以划分为更小的模块。这个过程采用自顶向下方法来实现。
- 子模块一般不超过50行。
- 划分子模块时应注意模块的独立性，即：使一个模块完成一项功能，耦合性愈少愈好。

总结

- 工欲善其事，必先利其器
- 至少熟练一种算法流程图，不必贪多求全