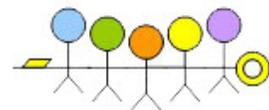




# 第三章 算法

## 第九讲



# 本章主要内容

- 熟悉算法的定义和算法的表示方法
- 掌握结构化程序设计方法



■ 数据结构 + 算法 = 程序

—— Nikiklaus Wirth

数据的类型  
和数据的组  
织形式

对数据  
的操作  
步骤

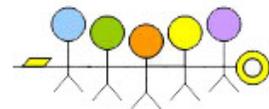


## 2.1 算法的概念

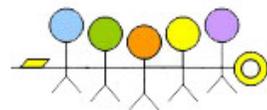
- 做事情要有步骤  决定成功与否
- 1、**算法**：为解决一个问题而采取的方法和步骤。
- 2、两大类别算法：**数值运算算法**和**非数值运算算法**。

↓  
求数值解，算法比较成熟。

↓  
用于事务管理，应用广泛。



- 3、算法特征：有穷性、确定性、有效性等。
- 4、算法的表示方法：可以归纳为两种——文字和图形。



## 2.2 算法的表示方法

- **方法1**：使用有序号的自然语言描述。
- **方法2**：使用流程图描述。
- **方法3**：使用**NS**图描述。
- **方法4**：使用伪码来描述。



## 方法1：使用有序号的自然语言描述。

- 通俗易懂，但容易出现“歧义性，表示的含义不严格。（参例**2.1—2.5**）

## 方法2：使用流程图描述。



起止框



或



流程线



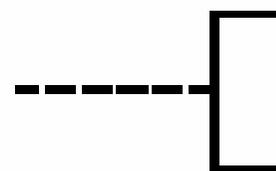
输入输出框



连接点



判断框



注释框



处理框



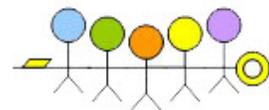
## 方法3：使用NS图描述。

- **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，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。
- 用**N-S**图表示的算法都是结构化的算法(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)。
- **N-S**图如同一个多层的盒子，又称盒图(**box diagram**)。



## 方法4：使用伪码来描述

- 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。
- **优点：**书写方便、格式紧凑，也比较好懂，便于向计算机语言算法(即程序)过渡。

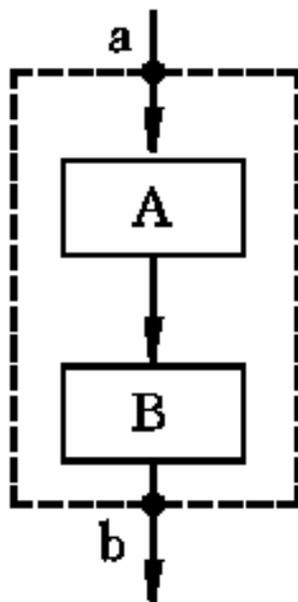


## 2.2 结构化程序设计方法

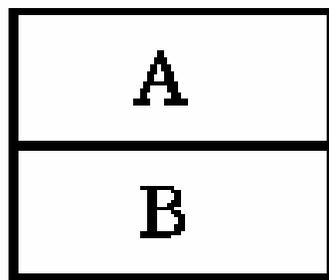
- 1966年，Bohra和Jacopini提出了以下三种基本结构，作为表示一个良好算法的基本单元。
- 程序设计的三种基本结构：
  - (1) 顺序结构
  - (2) 选择结构
  - (3) 循环结构，有两类循环结构：
    - ① 当型(While型)循环结构
    - ② 直到型(Until型)循环



## 顺序结构



流程图



NS图

如:

```
int a,b,c;
```

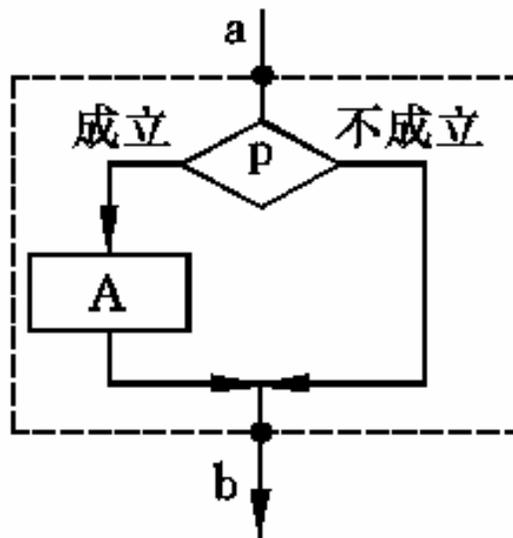
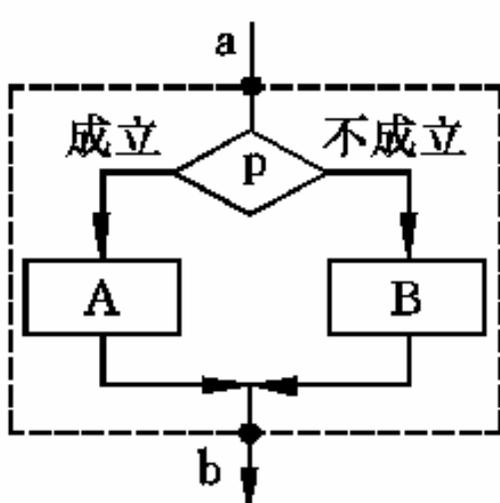
```
a = 1;
```

```
b = 2;
```

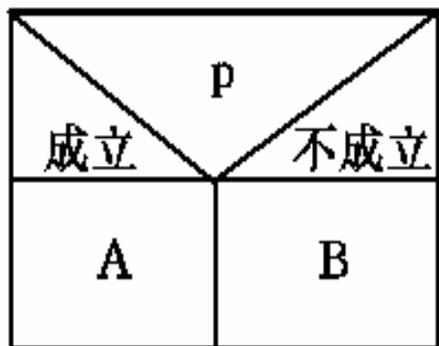
```
c = a+b;
```



# 选择结构



流程图



NS图



# 循环结构

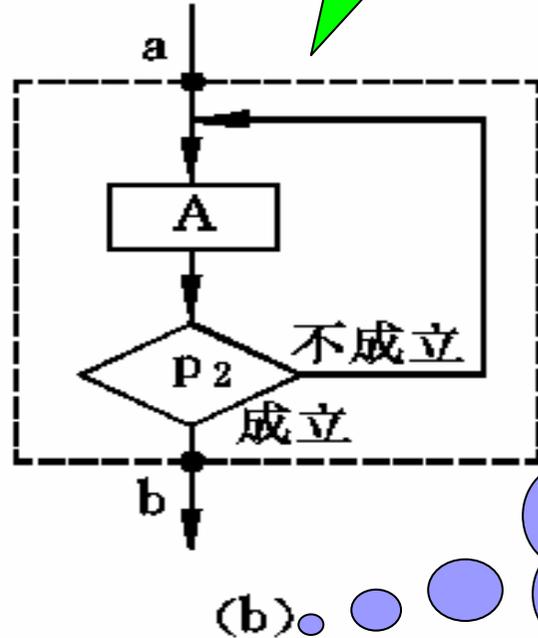
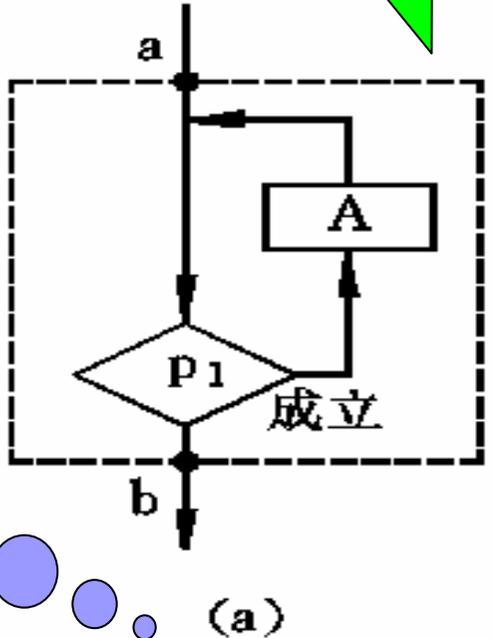
先判断, 再执行

先执行, 再判断

思考: 两种循环的区别

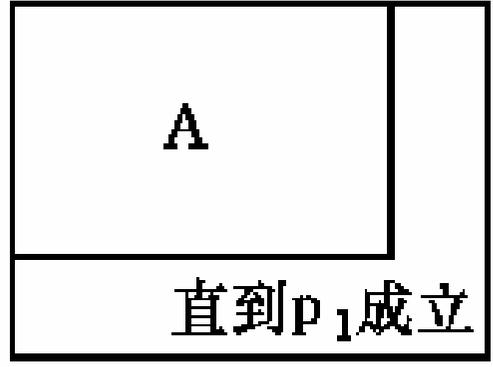
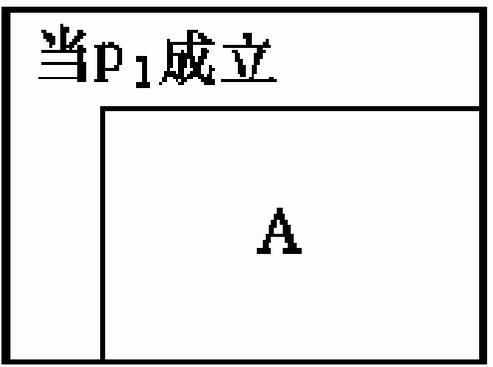
## 流程图

当型 (While型) 循环结构



直到型 (Until型) 循环结构

## NS图





```
int nSum,i;  
nSum = 0;  
i = 0;
```

```
while( i < 100)
```

```
{
```

```
    nSum += i;
```

```
    i++;
```

```
}
```

思考：如果i的  
初始值为100，  
两个程序的结果  
分别是多少？

```
int nSum,i;
```

```
nSum = 0;
```

```
i = 0;
```

```
do
```

```
{
```

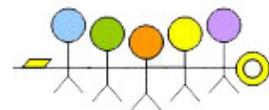
```
    nSum += i;
```

```
    ++i;
```

```
} while( i < 100)
```

当型(While型)循环结构

直到型(Until型)循环



- 以上三种基本结构的**共同特点**:
  - (1) 只有一个入口。
  - (2) 只有一个出口。
  - (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)。



- 由以上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。
- 每一个结构化的算法都是由一些基本结构顺序组成的；每个基本结构又可以包含其他的基本结构；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)。
- 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。



■ 例1、求 $1-1/2+1/3-1/4+\dots+1/99-1/100$ 。

➤ 使用自然语言描述，算法可以表示如下：

**S1:**  $1 \Rightarrow \text{sign}$

**S2:**  $1 \Rightarrow \text{sum}$

**S3:**  $2 \Rightarrow \text{deno}$

**S4:**  $(-1) \times \text{sign} \Rightarrow \text{sign}$

**S5:**  $\text{sign} \times (1/\text{deno}) \Rightarrow \text{term}$

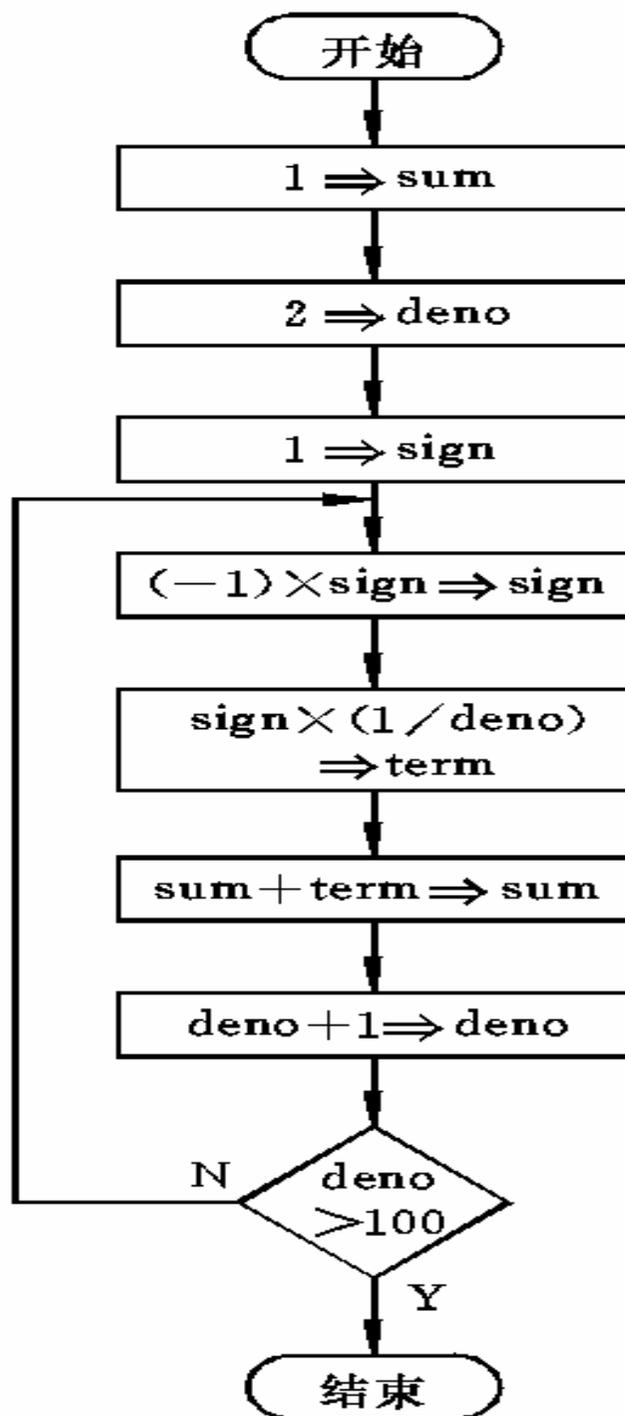
**S6:**  $\text{sum} + \text{term} \Rightarrow \text{sum}$

**S7:**  $\text{deno} + 1 \Rightarrow \text{deno}$

**S8:** 若 $\text{deno} \leq 100$ 返回**S4**；否则算法结束。



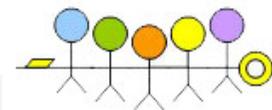
- 使用流程图描述，算法可以表示如下：





- 使用**NS**  
**图**描述，  
算法可以  
表示如下：

$1 \Rightarrow \text{sum}$
$2 \Rightarrow \text{deno}$
$1 \Rightarrow \text{sign}$
$(-1) \times \text{sign} \Rightarrow \text{sign}$
$\text{sign} \times \frac{1}{\text{deno}} \Rightarrow \text{term}$
$\text{sum} + \text{term} \Rightarrow \text{sum}$
$\text{deno} + 1 \Rightarrow \text{deno}$
直到 $\text{deno} > 100$
打印 $\text{sum}$



➤ 用伪代码表示的算法如下：

**BEGIN** (算法开始)

1=> sum

2=> deno

1=> sign

while deno <= 100

{

    (-1) × sign=>sign

    sign × 1/deno=>term

    sum+term=>sum

    deno+1=>deno

}

print sum

**END** (算法结束)



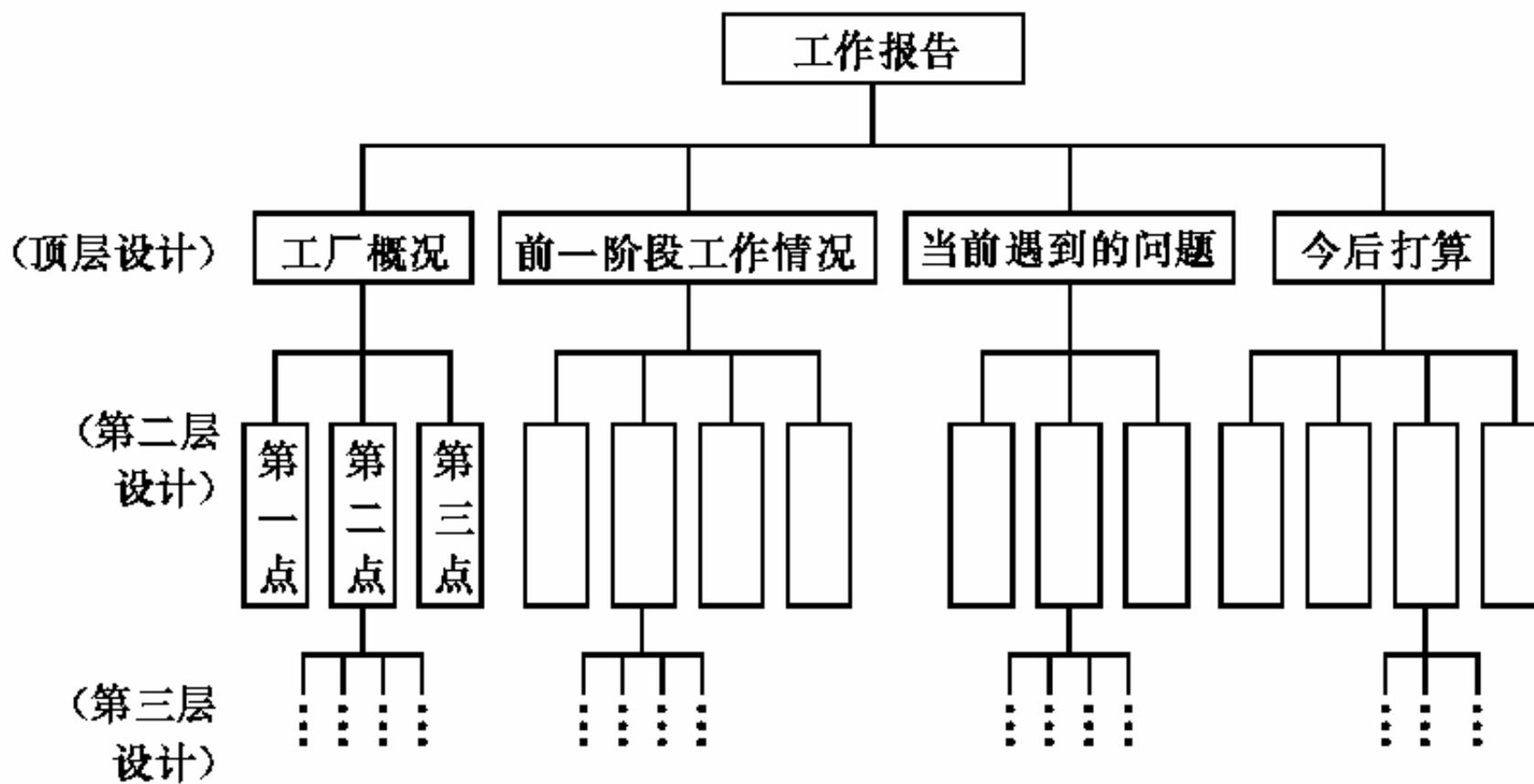
- 用三种基本结构组成的程序必然是结构化的程序。
- 结构化程序设计的优点：  
**易编、易读、易修改和易维护。**
- 减少了程序出错的机会，提高了程序的可靠性。

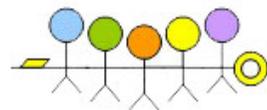


- 结构化程序设计强调程序设计风格和程序结构的规范化，提倡清晰的结构。
- 结构化程序设计方法的基本思路是，把一个复杂问题的求解过程分阶段进行，每个阶段处理的问题都控制在人们容易理解和处理的范围内。



- 结构化的程序的设计方法：
  - 自顶向下；
  - 逐步细化；
  - 模块化设计；
  - 结构化编码。
- 模块化设计的思想——“分而治之”。即把一个大的任务分为若干个相对简单的子任务。





## 2.3 结构化程序的设计过程

### ■ 设计过程:

- 1、确定算法。分析问题，建立相应的数学模型、选择公式，写出算法描述。
- 2、写出源程序。
- 3、上机调试。



■ 练习：100匹马，100块瓦，大马驮三，中马驮两，小马两个驮一块。问有多少匹马，多少块瓦？

➤ 编程过程：

1、分析阶段。

**BigHorses + MidHorses + SmlHorses=100**

**3\* BigHorses+2\* MidHorses +1/2\***

**SmlHorses=100**

其中**0<=BigHorses <34、 0<=MidHorses<=50、**

**0<= SmlHorses<=200**



- 如果直接解方程的话则无法计算，在这里可以使用枚举法来解决。

算法描述：

**BigHorses = 0**

当**BigHorses < 34**时

{

找出满足条件的  
**MidHorses**和  
**SmlHorses**的数

**BigHorses**加1

}

细化

**BigHorses = 0**

当**BigHorses < 34**时

{**MidHorses = 0**

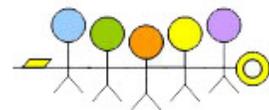
当**MidHorses <= 50**时

{找出满足条件**SmlHorses**的数

**MidHorses**加1}

**BigHorses**加1

}



再次细化

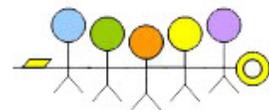
```
BigHorses = 0
当BigHorses<34时
{MidHorses = 0
  当MidHorses <= 50时
  {SmlHorses=100- BigHorses-
  MidHorses
  如果满足条件 $3 * \text{BigHorses} + 2 * \text{MidHorses} + 1/2 * \text{SmlHorses} = 100$ ，则输出。
  MidHorses加1}
  BigHorses加1
}
```



## 2、写出源码。

```
void main()
{ int BigHorse, MidHorse, SmlHorse;
  BigHorse = 0;
  while(BigHorse < 34)
  {   MidHorse = 0;
      while(MidHorse <= 50)
      {   SmlHorse = 100 - BigHorse - MidHorse;
          if( BigHorse * 3 + 2 * MidHorse + 1/2 *
              SmlHorse == 100 )

printf(“BigHorse=%d, MidHorse=%d, SmlHorse=%d\n”,
        BigHorse, MidHorse, SmlHorse);
          MidHorse++;
      }
      BigHorse++;
  }
}
```



### 3、调试运行。

```
BigHorse=2, MidHorse=30, SmallHorse=68  
BigHorse=5, MidHorse=25, SmallHorse=70  
BigHorse=8, MidHorse=20, SmallHorse=72  
BigHorse=11, MidHorse=15, SmallHorse=74  
BigHorse=14, MidHorse=10, SmallHorse=76  
BigHorse=17, MidHorse=5, SmallHorse=78  
BigHorse=20, MidHorse=0, SmallHorse=80
```