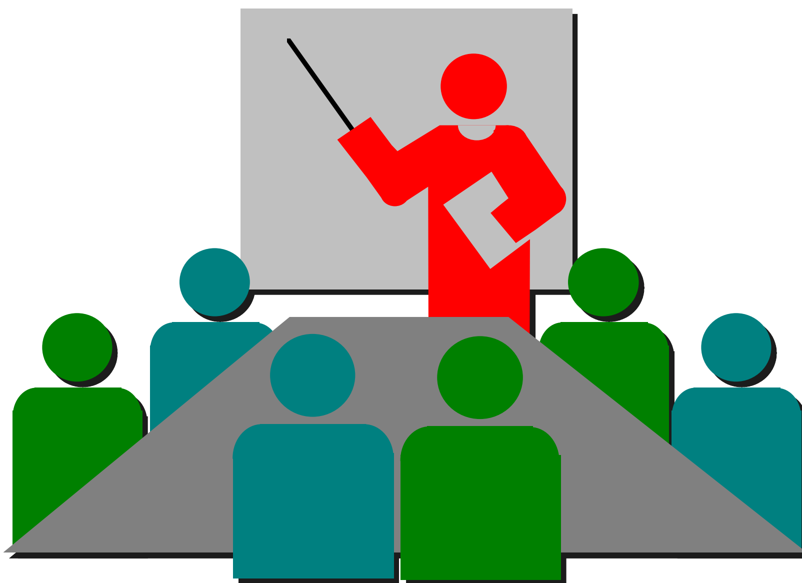
第2章：
白盒测试用例设计方法

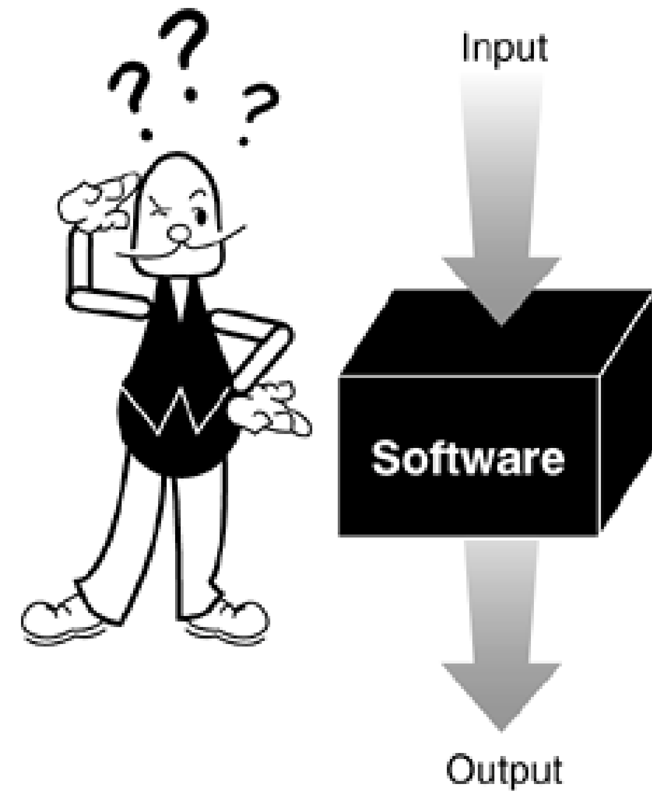


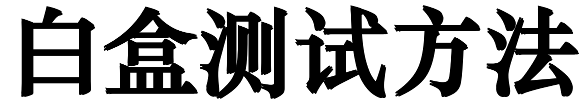
- 测试用例由测试输入数据以及与之对应的输出结果组成。
- 测试用例设计的好坏直接决定了测试的效果和结果。所以说在软件测试活动中最关键的步骤就是设计有效的测试用例。
- 测试用例可以针对黑盒测试设计用例，也可以针对白盒测试设计用例，我们今天开始讲针对白盒测试的用例设计方法。

回顾：黑盒测试

黑盒测试

又称为功能性测试或数据驱动测试。



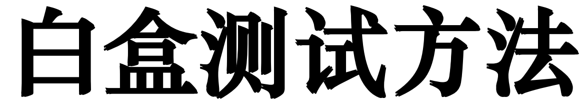


白盒测试方法

- 为什么要进行白盒测试?
- **如果所有软件错误的根源都可以追溯到某个唯一原因，那么问题就简单了。然而，事实上一个bug 常常是由多个因素共同导致的，如下图所示。**

x = 0

y = 5 / x

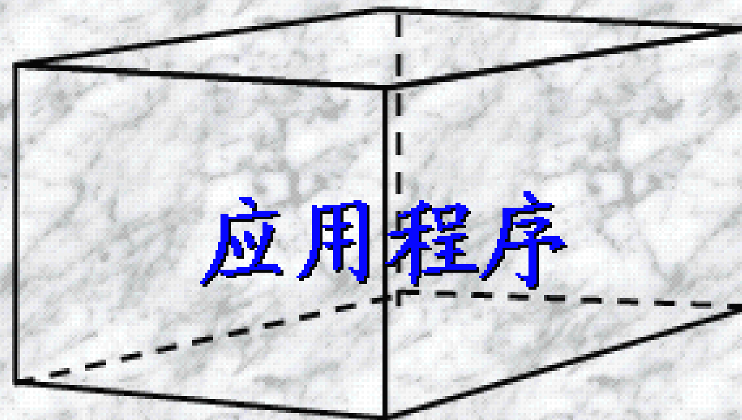


白盒测试方法

假设此时开发工作已结束，程序送交到测试组，没有人知道代码中有一个潜在的被**0** 除的错误。若测试组采用的测试用例的执行路径没有同时经过**x=0**和**y=5/x**进行测试，显然测试工作似乎非常完善，测试用例覆盖了所有执行语句，也没有被 **0** 除的错误发生。

白盒测试方法

- 白盒测试将被测程序看作一个**打开的盒子**，测试者**能够看到被测源程序**，可以分析被测程序的内部结构，此时测试的焦点集中在根据其内部结构设计测试用例。
- 白盒测试方法又称为 **结构测试** 或 **逻辑驱动测试**

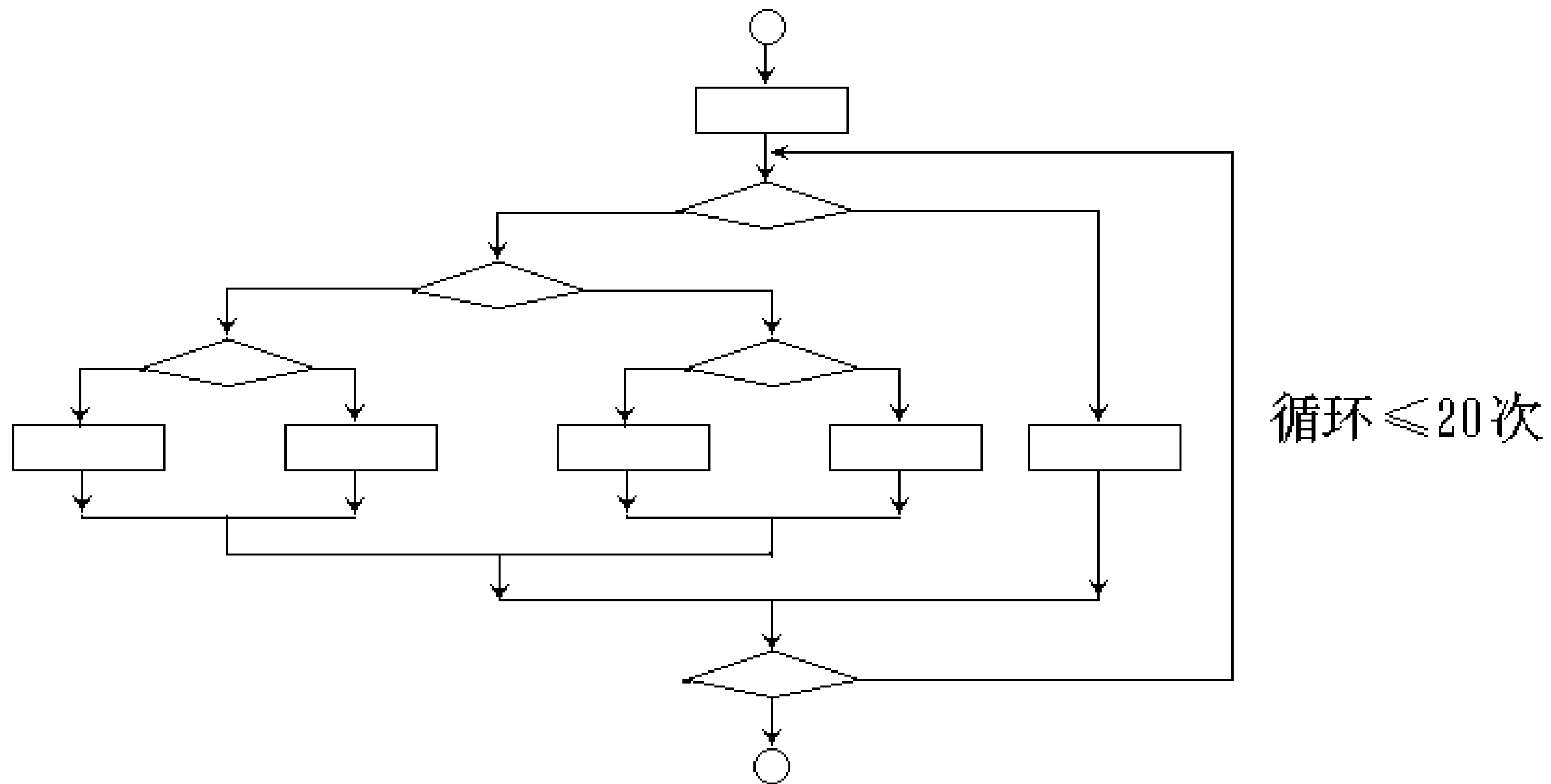


白盒测试需要完全了解程序结构和处理过程，它按照程序内部逻辑测试程序，检验程序中每条通路是否按预定要求正确工作。也被称为程序员测试。

白盒测试

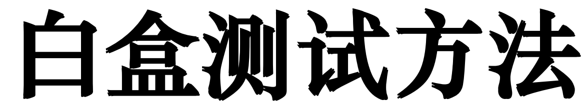
- 程序的结构形式是白盒测试的主要依据。
- 程序结构主要用 流程图 N-S图来表示
- 程序的执行路径数目庞大，让程序的所有路径都执行一次,全面的白盒测试将产生百分之百正确的程序，但实际上是不可能的

- 对一个具有多重选择和循环嵌套的程序，不同的路径数目可能是天文数字。给出一个小程序的流程图，它包括了一个执行20次的循环。
- 包含的不同执行路径数达5^{20}条，对每一条路径进行测试需要1毫秒，假定一年工作365 × 24小时，要想把所有路径测试完，需3170年。



白盒测试方法

- 白盒测试主要是检查程序的内部结构、逻辑、循环和路径。常用测试用例设计方法有：
  - 逻辑覆盖法（逻辑驱动测试）
  - 基本路径测试方法
- 此外还有：
  - 循环测试
  - 程序插桩
  - Z路径覆盖



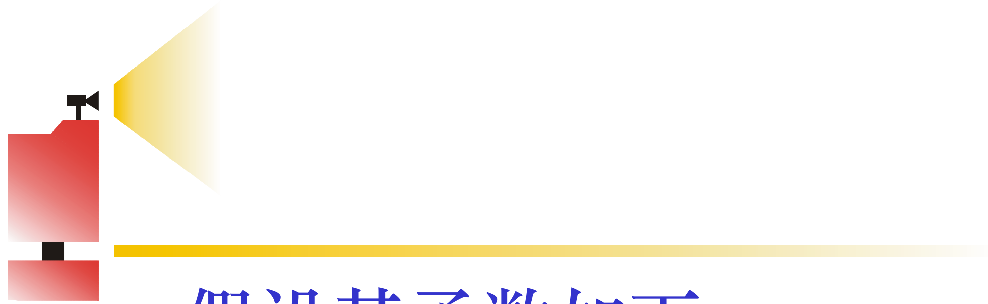
白盒测试方法

- A. 逻辑覆盖法
- B. 基本路径测试法
- C. 循环测试
- D. 程序插桩技术

A。逻辑覆盖法

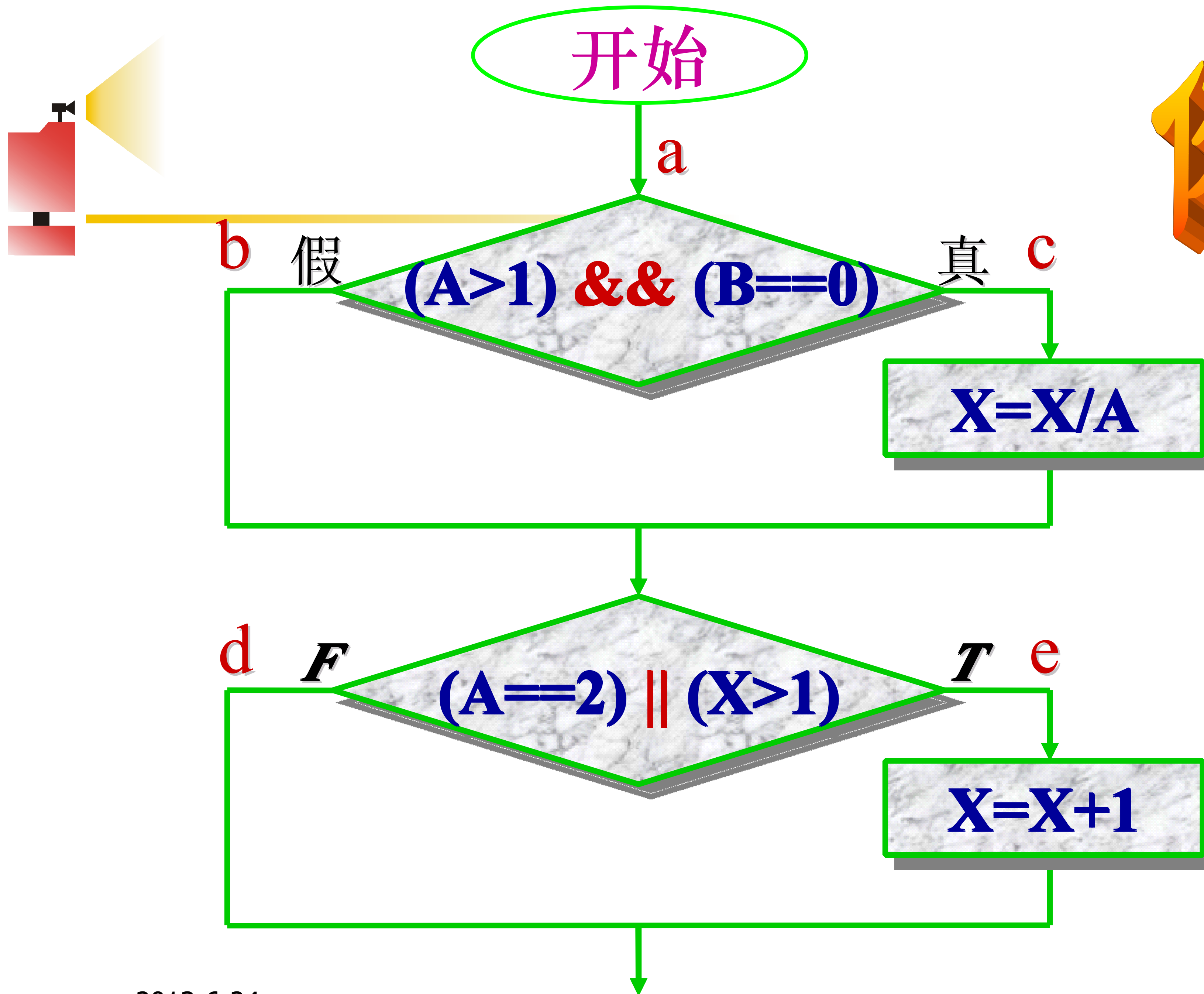
主要是测试覆盖率，以程序内在逻辑结构为基础的测试。包括以下6种类型：

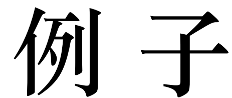
- 语句覆盖
- 判定覆盖
- 条件覆盖
- 判定－条件覆盖
- 条件组合覆盖
- 路径覆盖。



- 假设某函数如下：

```
func ( )
{ //…
  if(A>1  && B==0)  X=X/A;
  if(A==2 || X>1)   X=X+1;
// …
}
```


开始
a
b
假
(A>1) && (B==0)
真
c
X=X/A
d
F
(A==2) || (X>1)
T
e
X=X+1
例



例 子

L1： (a → c → e)

= {(A>1) and (B=0)} and

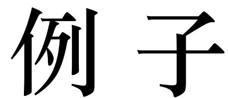
{(A=2) or (X/A>1)}

= (A>1) and (B=0) and (A=2) or

(A>1) and (B=0) and (X/A>1)

= (A=2) and (B=0) or

(A>1) and (B=0) and (X/A>1)



例 子

L2: (a→ b → d)

= not{(A>1) and (B=0)} and

not{(A=2) or (X>1)}

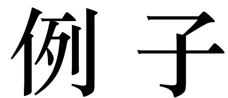
= { not (A>1) or not (B=0) } and

{ not (A=2) and not (X>1) }

= not (A>1) and not (A=2) and not (X>1)

or

not (B=0) and not (A=2) and not (X>1)



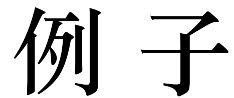
例 子

L3： (a→ b→ e)

= not {(A>1) and (B=0)} and {(A=2) or (X>1)}

= { not (A>1) or not (B=0)} and {(A=2) or (X>1)}

= not (A>1) and (A=2) or
not (A>1) and (X>1) or
not (B=0) and (A=2) or
not (B=0) and (X>1)



例 子

L4: (a→ c → d)

= {(A>1) and (B=0)} and

not {(A=2) or (X/A>1)}

= (A>1) and (B=0) and not (A=2) and

not (X/A>1)

语句覆盖

- 原理：如果语句中有错误，仅靠观察不执行可能发现不了.
- 语句覆盖就是设计若干个测试用例，运行被测程序，使得每一条可执行语句至少执行一次。
- 在例图中，正好所有的可执行语句都在路径L1上，所以选择路径 L1设计测试用例，就可以覆盖所有的可执行语句。

语句覆盖

- 规定测试用例的设计格式如下:

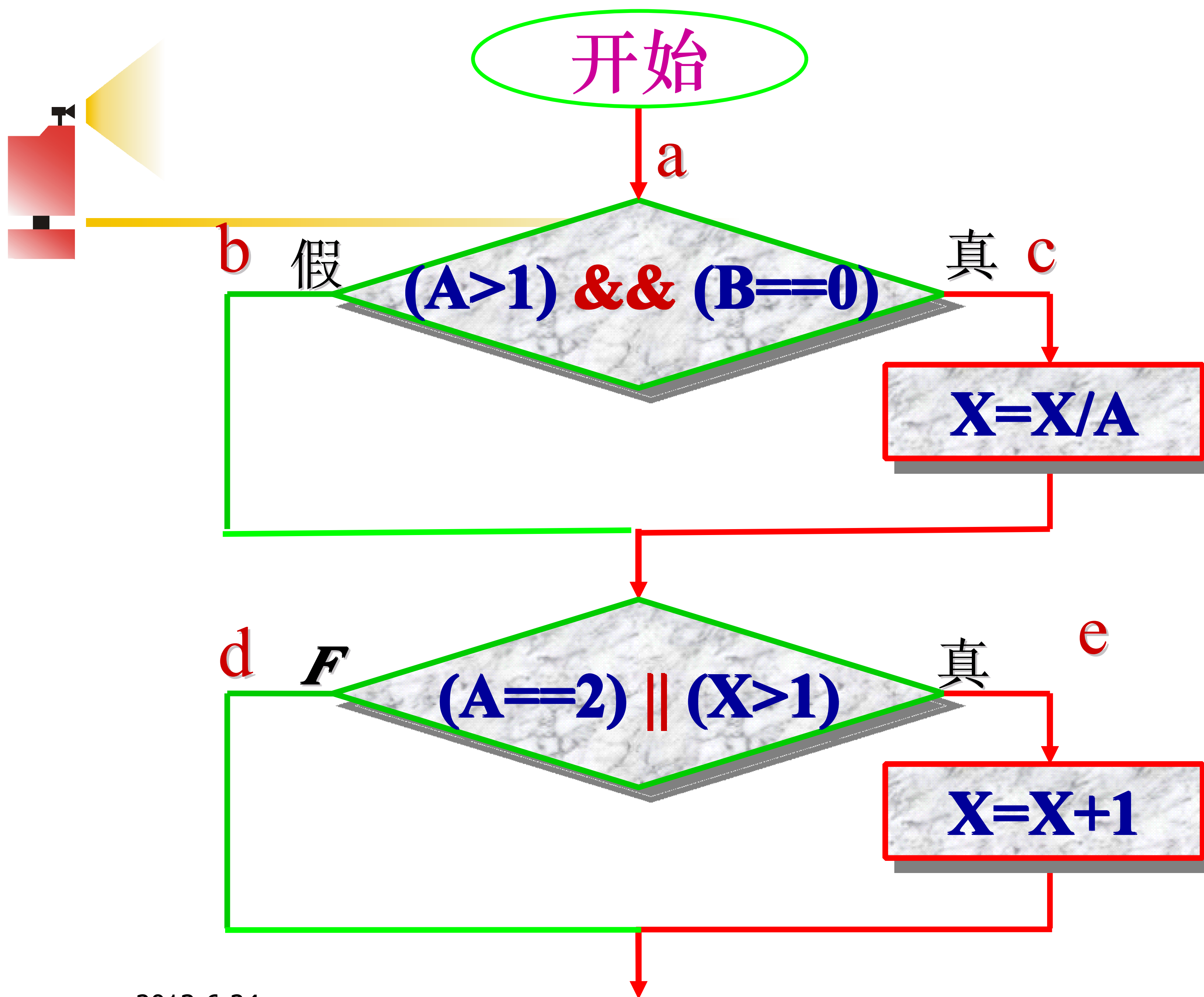
【输入的(A, B, X)，输出的(A, B, X)】

- 为图例设计满足**语句覆盖**的测试用例是:

【(2, 0, 4)，(2, 0, 3)】

覆盖 *a-c-e*【L1】

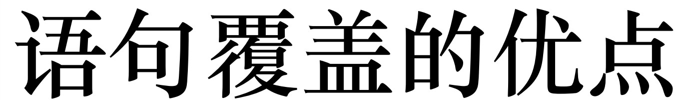
输入 A B X	预期结果 A B X



例

语句覆盖

- 语句覆盖率
  - 已执行的可执行语句占程序中可执行语句总数的百分比
- 复杂的程序不可能达到语句的完全覆盖
- 语句覆盖率越高越好



- 检查所有语句
- 结构简单的代码的测试效果较好
- 容易实现自动测试
- 代码覆盖率高
- 如果是程序块覆盖，则不涉及程序块中的源代码

语句覆盖不能检查出的错误

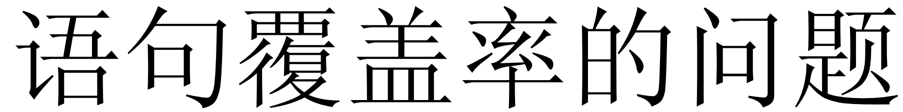
- 逻辑运算（&&、||）错误
  判定的第一个运算符“&&”错写成“||”，或第二个运算符“||”错写成“&&”,这时使用上述的测试用例仍然可以达到100%的语句覆盖.
- 循环语句错误
  - 循环次数错误
  - 跳出循环条件错误

语句覆盖不能检查出的错误

- 循环语句例子

```
for(i=0;i<10;i++)
{ statement;
}
while(x>3)
{ statement;
}
```

```
for(i=0; i<=10; i++)
{
        statement;
}
while(x>3&&x<7)
{
        statement;
}
```



语句覆盖率的问题

- **能达到很高的语句覆盖率**
- **语句覆盖率看似很高，却有严重缺陷:**
-

```
if(x!=1)
{
        statements;     }99句
        ……;
}
else
{
        statement;      }1句
}
```

测试用例:

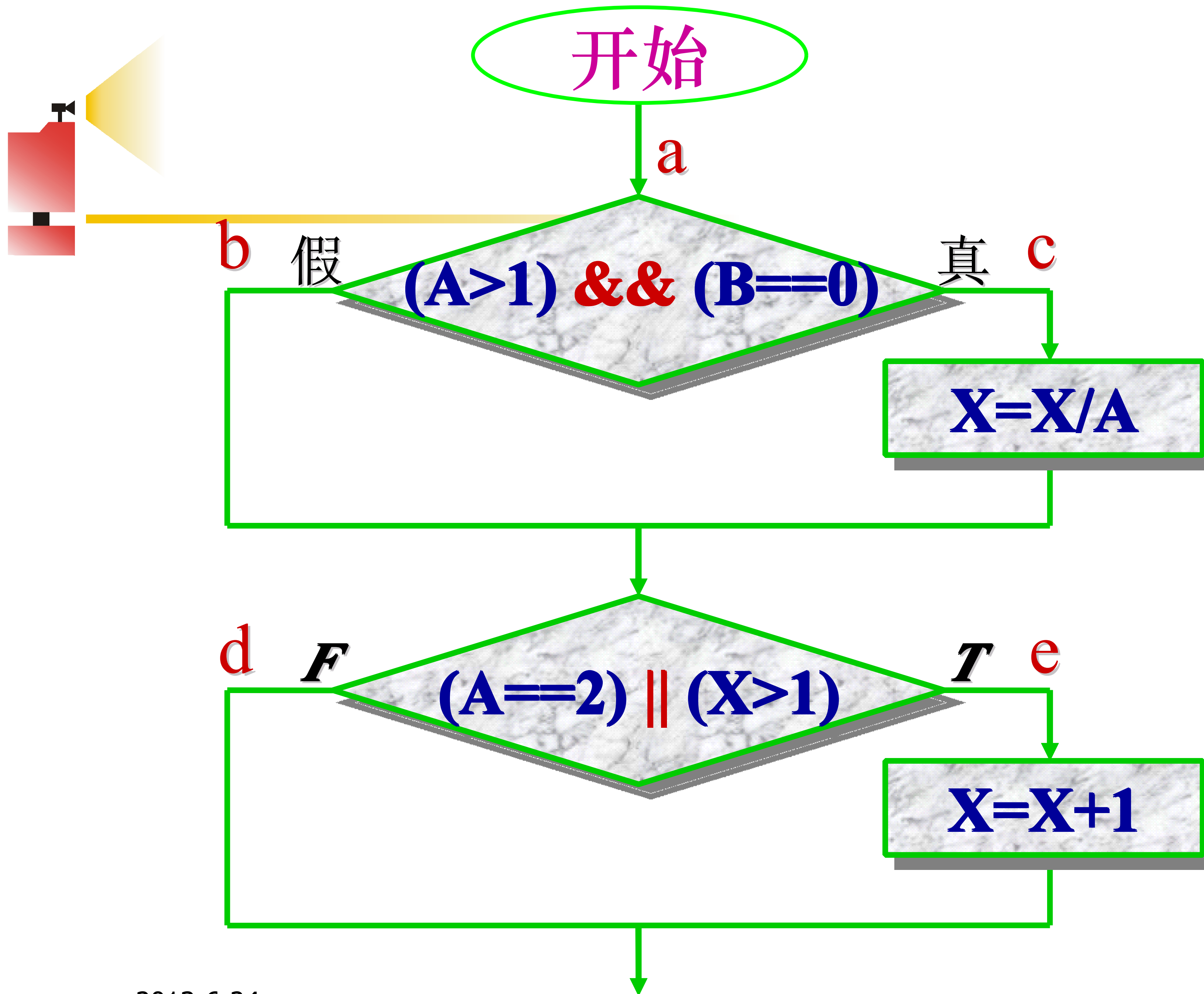
x = 2

语句覆盖率99%

50%的分支没有达到

判定覆盖 Decision Coverage

- 判定覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。
- 判定覆盖又称为分支覆盖。
- 对于图例，如果选择路径L1和L2，就可得满足要求的测试用例：



例

判定覆盖

- 【(2, 0, 4)，(2, 0, 3)】
  覆盖 ace【L1】
  【(1, 1, 1)，(1, 1, 1)】
  覆盖 abd【L2】

(A==2) **and** (B==0) **or**

(A>1) **and** (B==0) **and** (X/A>1)

判定覆盖

not (A>1) and not (A==2) and not (X>1)

or

not (B==0) and **not (A==2)** and **not (X>1)**

判定覆盖

- 如果选择路径L3和L4，还可得另一组可用的测试用例：

  【(2, 1, 1)，(2, 1, 2)】覆盖 abe【L3】

  【(3, 0, 3)，(3, 1, 1)】覆盖 acd【L4】

---

not (A>1) and (X>1) or not (B==0) and (A==2) or not (B==0) and (X>1)

(A>1) and (B==0) and not (A==2) and not (X/A>1)

条件覆盖Condition Coverage

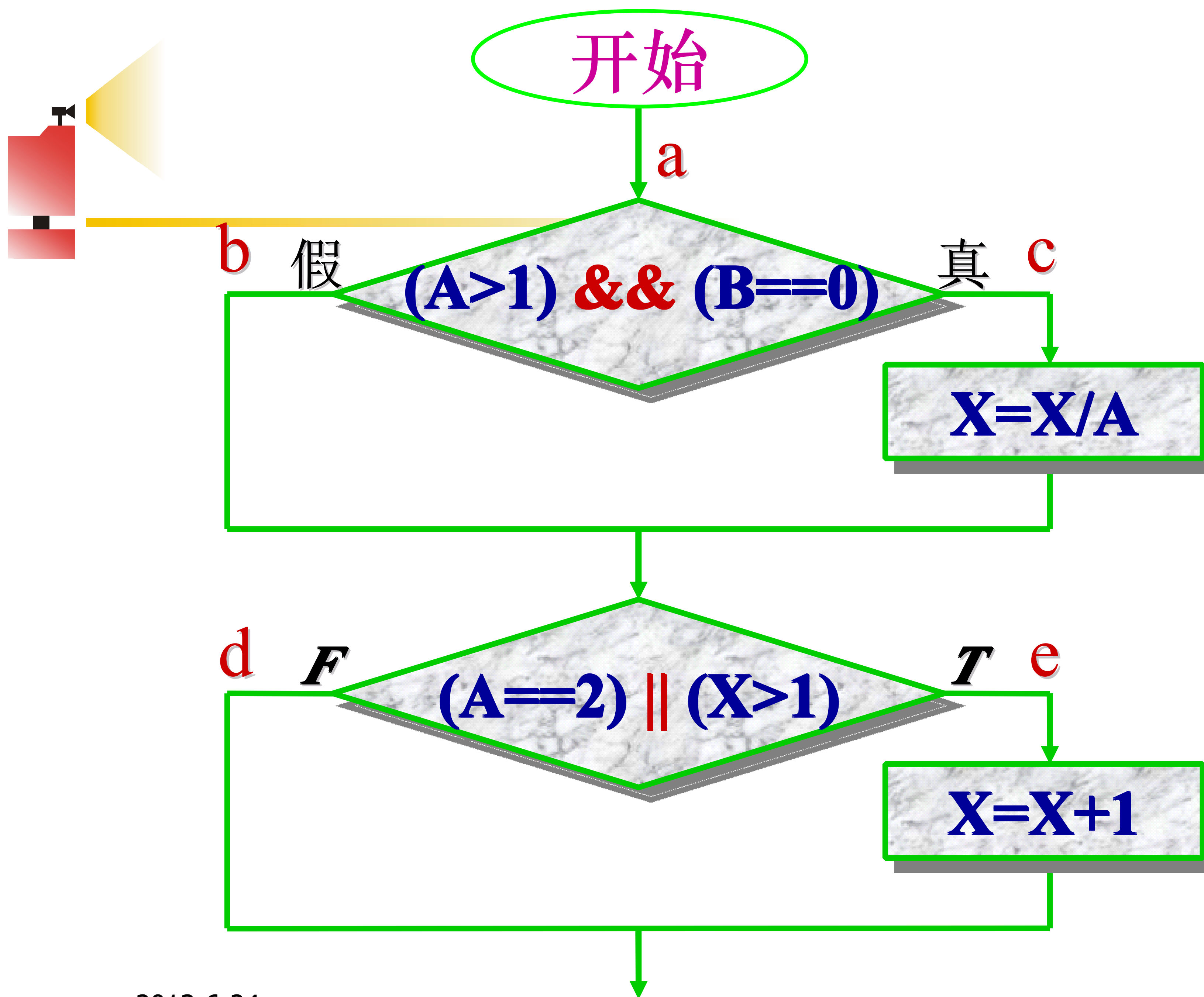
- 在设计程序中，一个判定语句是由多个条件组合而成的复合判定，判定**(a)&&(b||c)**包含了三个条件：**a,b**和**c**。为了更彻底的实现逻辑覆盖，可以采用条件覆盖。

条件覆盖Condition Coverage

- 条件覆盖就是设计若干个测试用例，运行被测程序，使得程序中每个判断的每个条件的可能取值至少执行一次。
- 在图例中，我们事先可对所有条件的取值加以标记。

条件覆盖 Condition Coverage

- 例如：对于第一个判断，
  - 条件 $A>1$ 取真为 T_1，取假为 $\overline{T_1}$
    条件 $B=0$ 取真为 T_2，取假为 $\overline{T_2}$
- 对于第二个判断:
  - 条件 $A=2$ 取真为 T_3，取假为 $\overline{T_3}$
    条件 $X>1$ 取真为 T_4，取假为 $\overline{T_4}$


开始
a
b
假
(A>1) && (B==0)
真
c
X=X/A
d
F
(A==2) || (X>1)
T
e
X=X+1
例

条件覆盖Condition Coverage

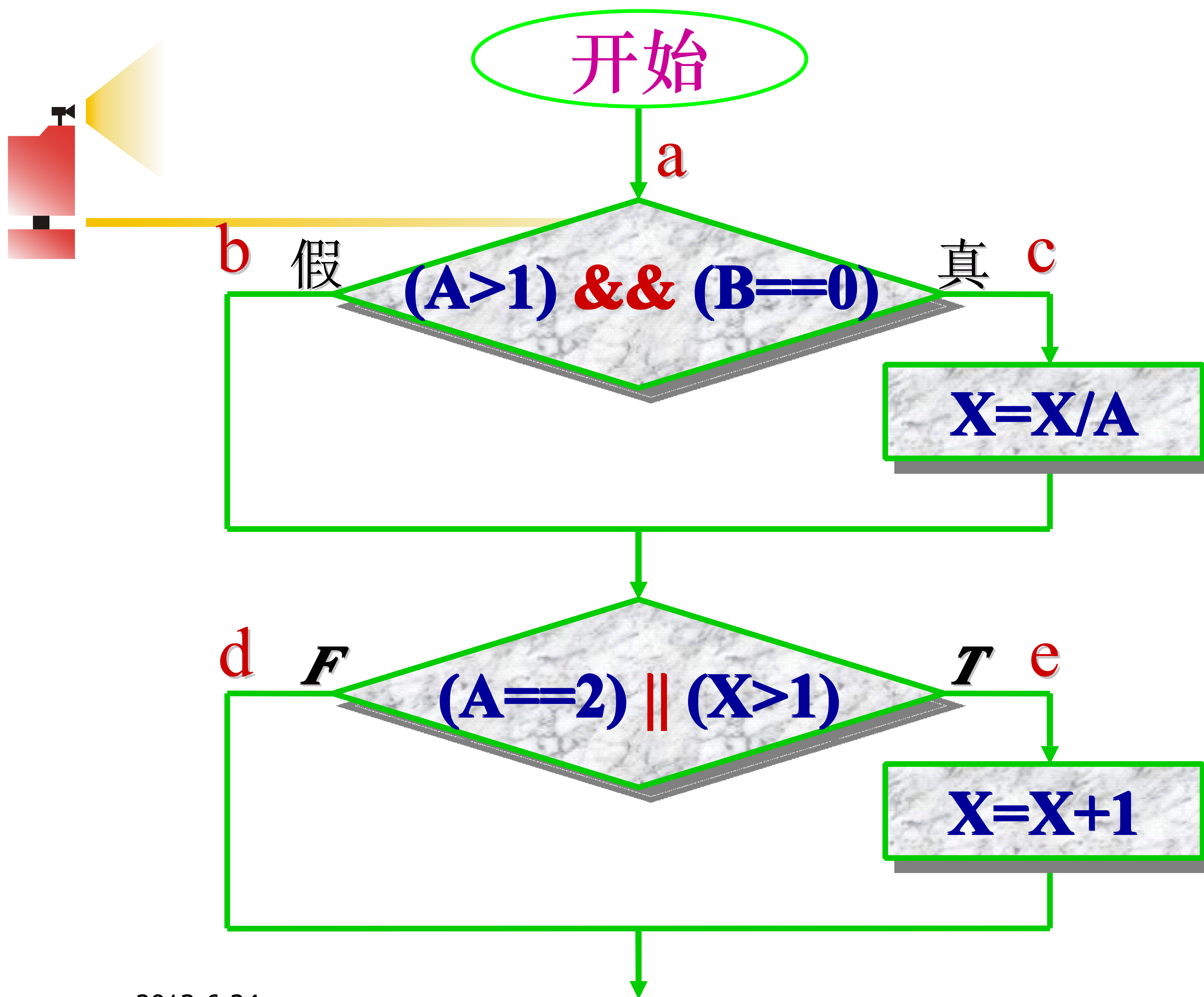
测试用例	覆盖分支	条件取值
【(2, 0, 4)，(2, 0, 3)】	L1(c, e)	$T_1T_2T_3T_4$
【(1, 0, 1)，(1, 0, 1)】	L2(b, d)	$\overline{T_1}T_2\overline{T_3}\,\overline{T_4}$
【(2, 1, 1)，(2, 1, 2)】	L3(b, e)	$T_1\overline{T_2}T_3\overline{T_4}$

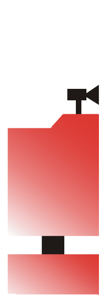
条件覆盖

或　测 试 用 例	覆盖分支	条件取值
【(1, 0, 3)，(1, 0, 4)】	L3(b, e)	$\overline{T_1}T_2\overline{T_3}T_4$
【(2, 1, 1)，(2, 1, 2)】	L3(b, e)	$T_1\overline{T_2}T_3\overline{T_4}$

判定－条件覆盖 CDC

- 判定/条件覆盖实际上是将判定覆盖和条件覆盖结合起来的一种方法，
- 就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判定的可能结果也至少出现一次。
- 设计测试用例覆盖4个条件的8种取值以及4个判定分支。


开始
a
b
假
(A>1) && (B==0)
真
c
X=X/A
d
F
(A==2) || (X>1)
T
e
X=X+1
例



判定－条件覆盖CDC

测 试 用 例	覆盖分支	条件取值
【(2, 0, 4)，(2, 0, 3)】	L1(c, e)	$T_1T_2T_3T_4$
【(1, 1, 1)，(1, 1, 1)】	L2(b, d)	$\overline{T_1}\,\overline{T_2}\,\overline{T_3}\,\overline{T_4}$

(A=2) and (B=0) or

(A>1) and (B=0) and (X/A>1)

not (A>1) and not (A=2) and not (X>1)

or

not (B=0) and not (A=2) and not (X>1)

判定—条件覆盖CDC

- 分析：从表面上看，判定/条件覆盖测试了各个判定中的所有条件的取值，但实际上，编译器在检查含有多个条件的逻辑表达式时，某些情况下的某些条件将会被其它条件所掩盖。因此，判定/条件覆盖也不一定能够完全检查出逻辑表达式中的错误。

判定－条件覆盖CDC

- 例如：对于第一个判定(A>1)&&(B==0)来说，必须A>1和B==0这两个条件同时满足才能确定该判定为真。如果A>1为假，则编译器将不再检查B==0这个条件，那么即使这个条件有错也无法被发现。对于第二个判定(A==2)||(X>1)来说，若条件A==2满足，就认为该判定为真，这时将不会再检查X>1，那么同样也无法发现这个条件中的错误。

- 条件组合覆盖就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次。

条件组合覆盖

记 ① $A > 1, B = 0$ 作 T_1T_2

② $A > 1, B \neq 0$ 作 $T_1\overline{T_2}$

③ $A \ngtr 1, B = 0$ 作 $\overline{T_1}T_2$

④ $A \ngtr 1, B \neq 0$ 作 $\overline{T_1}\,\overline{T_2}$

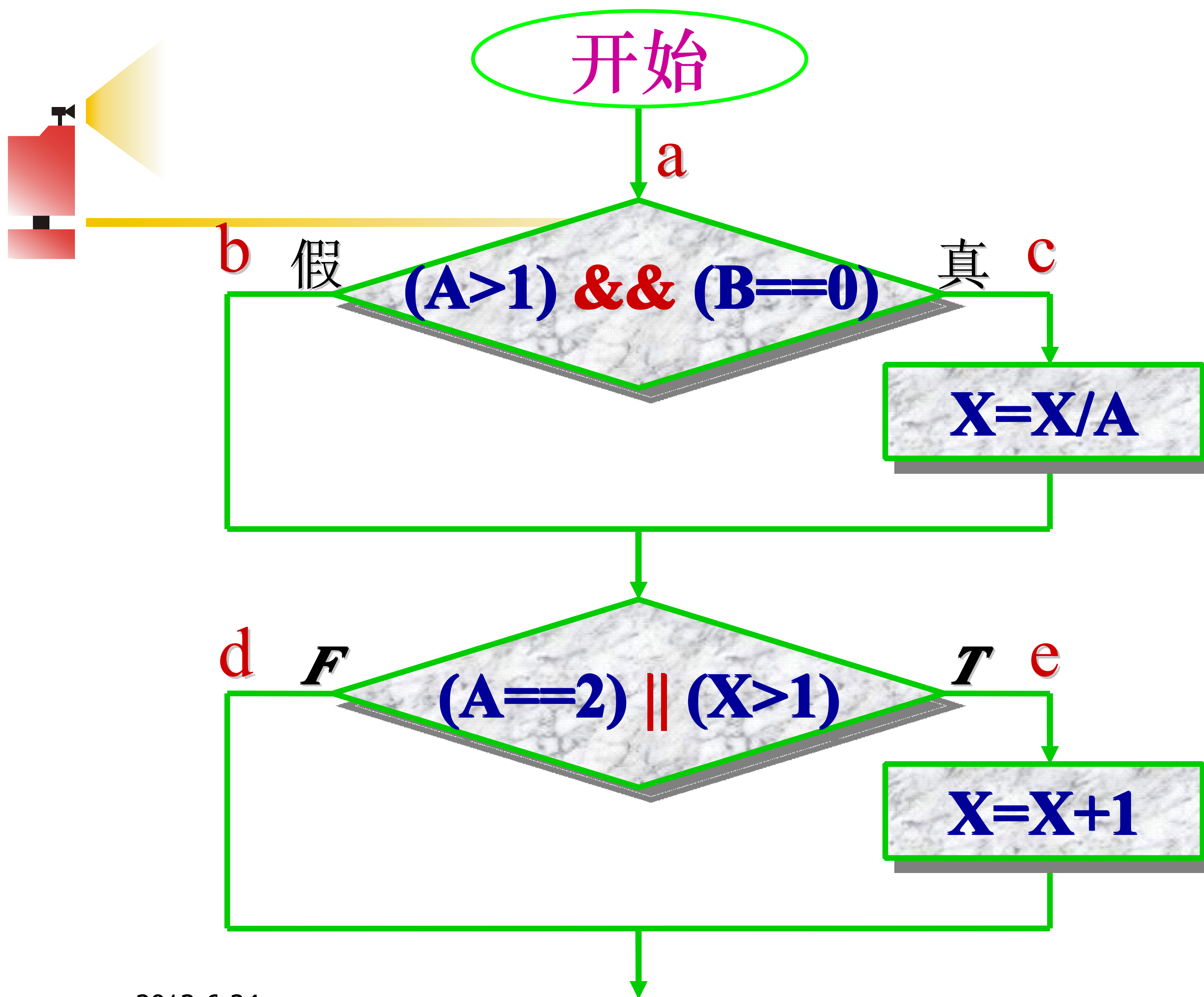
---

⑤ $A = 2, X > 1$ 作 T_3T_4

⑥ $A = 2, X \ngtr 1$ 作 $T_3\overline{T_4}$

⑦ $A \neq 2, X > 1$ 作 $\overline{T_3}T_4$

⑧ $A \neq 2, X \ngtr 1$ 作 $\overline{T_3}\,\overline{T_4}$


开始
a
b
假
(A>1) && (B==0)
真
c
X=X/A
d
F
(A==2) || (X>1)
T
e
X=X+1
例

条件组合覆盖

测 试 用 例 **覆盖条件 覆盖组合**

【(2, 0, 4), (2, 0, 3)】(L1) $T_1T_2T_3T_4$ ①, ⑤

【(2, 1, 1), (2, 1, 2)】(L3) $T_1\overline{T_2}T_3\overline{T_4}$ ②, ⑥

【(1, 0, 3), (1, 0, 4)】(L3) $\overline{T_1}T_2\overline{T_3}T_4$ ③, ⑦

【(1, 1, 1), (1, 1, 1)】(L2) $\overline{T_1}\,\overline{T_2}\,\overline{T_3}\,\overline{T_4}$ ④, ⑧

- 分析：上面这组测试用例覆盖了所有8种条件取值的组合，覆盖了所有判定的真假分支，但是却丢失了一条路径L4。

路径覆盖

- 路径覆盖就是设计足够的测试用例，**覆盖程序中所有可能的路径。**

测 试 用 例	通过路径	覆盖条件
【(2, 0, 4), (2, 0, 3)】	ace (L1)	$T_1T_2T_3T_4$
【(1, 1, 1), (1, 1, 1)】	abd (L2)	$\overline{T_1}\,\overline{T_2}\,\overline{T_3}\,\overline{T_4}$
【(1, 1, 2), (1, 1, 3)】	abe (L3)	$\overline{T_1}\,\overline{T_2}\,\overline{T_3}T_4$
【(3, 0, 3), (3, 0, 1)】	acd (L4)	$T_1T_2\overline{T_3}\,\overline{T_4}$

路径覆盖

- 分析：

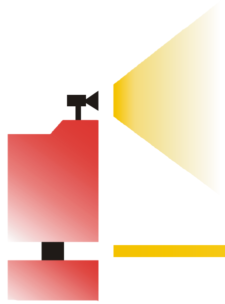
  虽然前面一组测试用例满足了路径覆盖，但并没有覆盖程序中所有的条件组合，即满足路径覆盖的测试用例并不一定满足组合覆盖。

路径覆盖

- 说明：
  - 对于比较简单的小程序，实现路径覆盖是可能做到的。但如果程序中出现较多判断和较多循环，可能的路径数目将会急剧增长，要在测试中覆盖所有的路径是无法实现的。为了解决这个难题，只有把覆盖路径数量压缩到一定的限度内，如程序中的循环体只执行一次。

路径覆盖

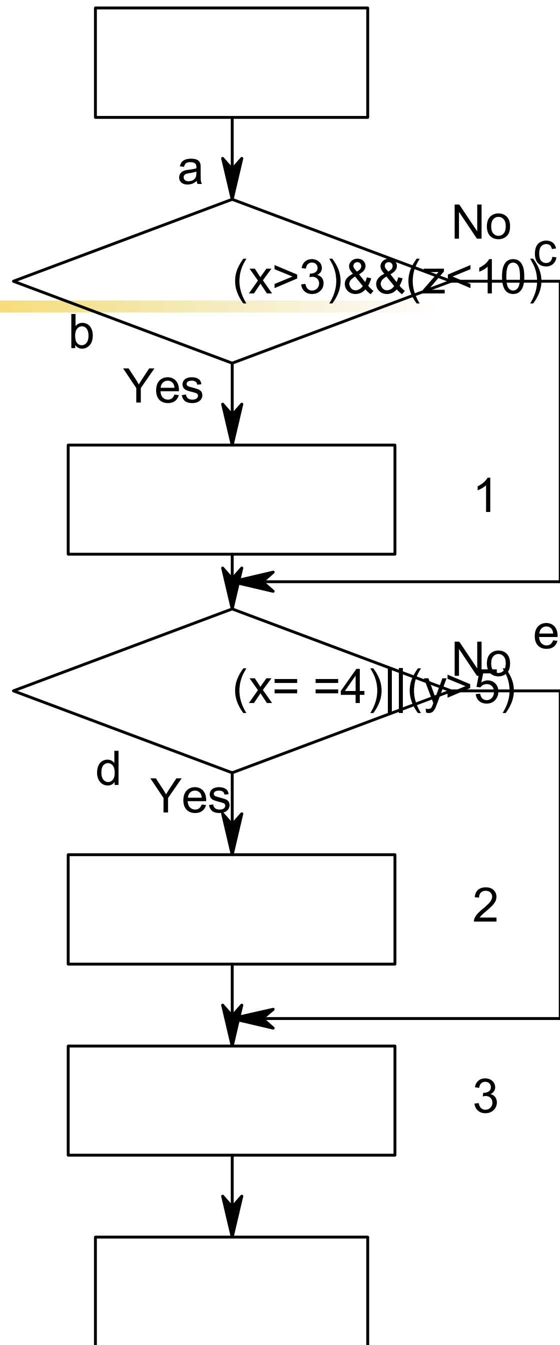
- 在实际测试中，即使对于路径数很有限的程序已经做到路径覆盖，仍然不能保证被测试程序的正确性，还需要采用其他测试方法进行补充。



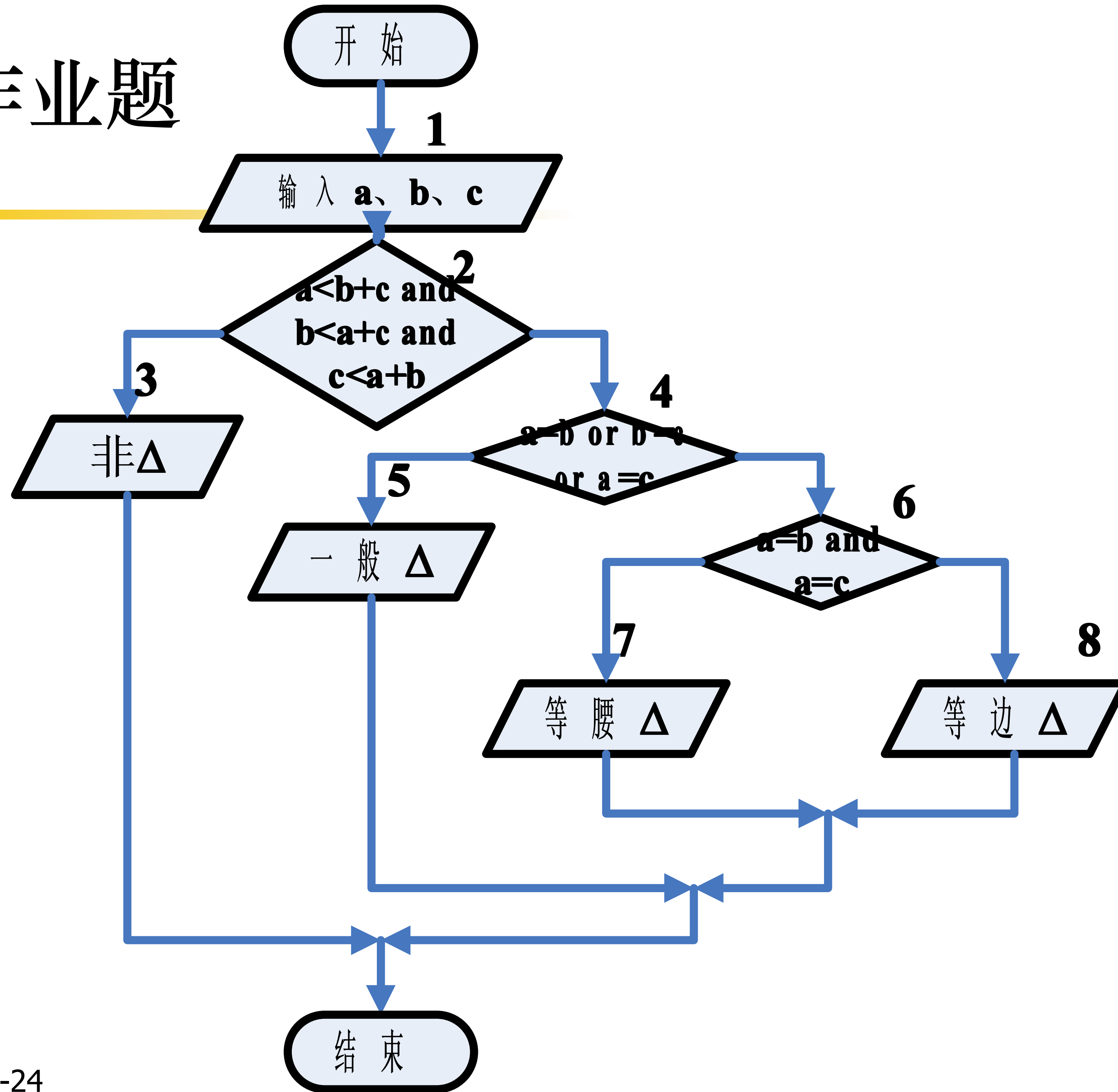
练习题

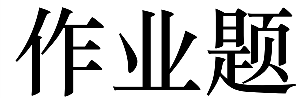
```
void DoWork (int x, int y, int z)
{   int  k=0,j=0;
    if ( (x>3) && (z<10) )
    {   k=x*y-1;      //语句块1
        j=sqrt(k);
    }
    if ( (x= =4) || (y>5) )
   {   j=x*y+10;   //语句块2
   }
   j=j%3;             //语句块3
}
```

流程图



作业题





作业题

- 试做出左边三角形问题的语句覆盖,条件覆盖,判定覆盖,判定－条件覆盖、组合条件覆盖的测试用例.并注明满足覆盖的条件.eg:
- a b c
- 3 4 5 T1T2T3F4F5F6