# 白盒测试

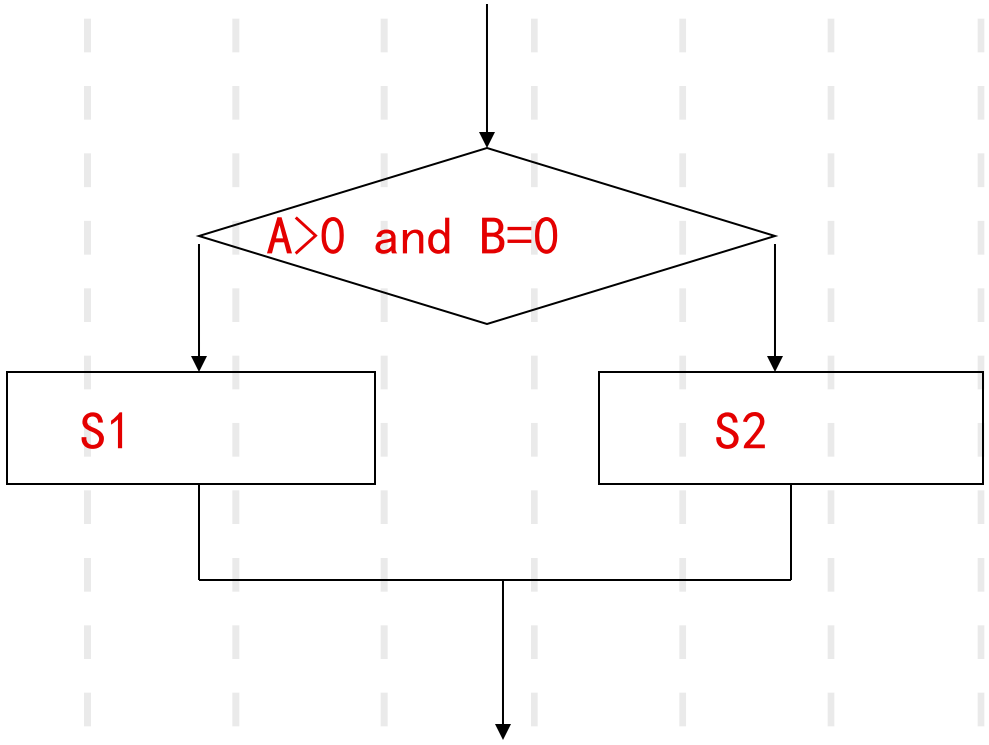
程序结构分析
- 结构流分析
- 数据流分析
- 控制流分析

逻辑覆盖：语句覆盖、判定覆盖、条件覆盖、判定—条件覆盖、路径覆盖等

程序插桩：方法简介、断言语句

其他白盒测试简介

如图显示某程序的逻辑结构。试为它设计足够的测试用例，分别实现对程序的判定覆盖、条件覆盖和条件组合覆盖。



| 覆盖种类 | 需满足的条件 | | 测试数据 | 期望结果 |
| --- | --- | --- | --- | --- |
| 判定覆盖 | A>1, B=0 | | A=2, B=0 | 执行S1 |
| | A>1, B≠0或<br>A≤1, B=0或<br>A≤1, B≠0 | | A=2, B=1或<br>A=1, B=0或<br>A=1, B=1 | 执行S2 |
| 条件覆盖 | 以下四种情况各出现一次 | | | |
| | A>1 | B=0 | A=2， B=0 | 执行S1 |
| | A≤1 | B≠0 | A=1， B=1 | 执行S2 |
| 条件组合覆盖 | A>1, B=0 | | A=2, B=0 | 执行S1 |
| | A>1, B≠0 | | A=2, B=1 | 执行S2 |
| | A≤1, B=0 | | A=1, B=0 | 执行S2 |
| | A≤1, B≠0 | | A=1, B=1 | 执行S2 |

# 最少测试用例计算

众所周知，结构化程序是由3种基本控制结构组成的：

顺序型－构成串行操作；

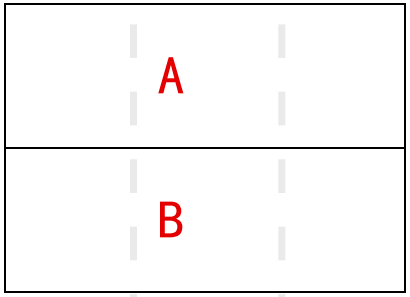
选择型－构成分支操作；

重复型－构成循环操作。

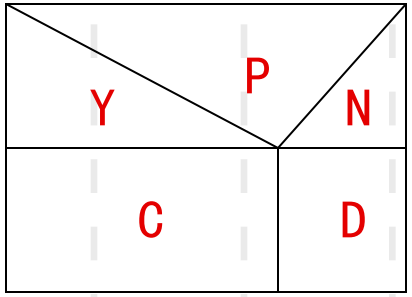
为了把问题简化，避免出现测试用例太多的组合，把构成循环操作的重复型结构用选择结构代替，只对循环体检验一次。这样，任一循环便改造成进入循环体或不进入循环体的分支操作了。

在做了以上简化循环的假设以后，对于一般的程序控制流，只要考虑选择型结构。

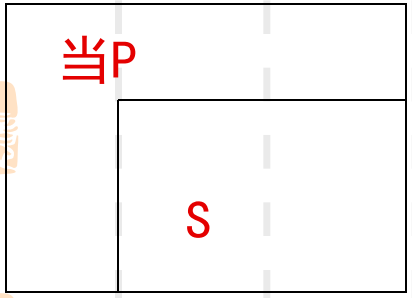
# N-S图表示的基本控制结构



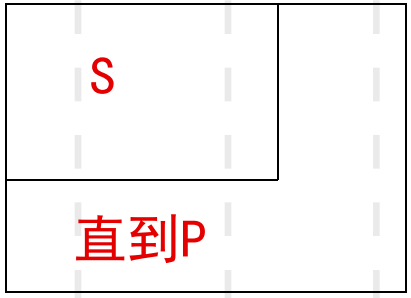

顺序型




选择型



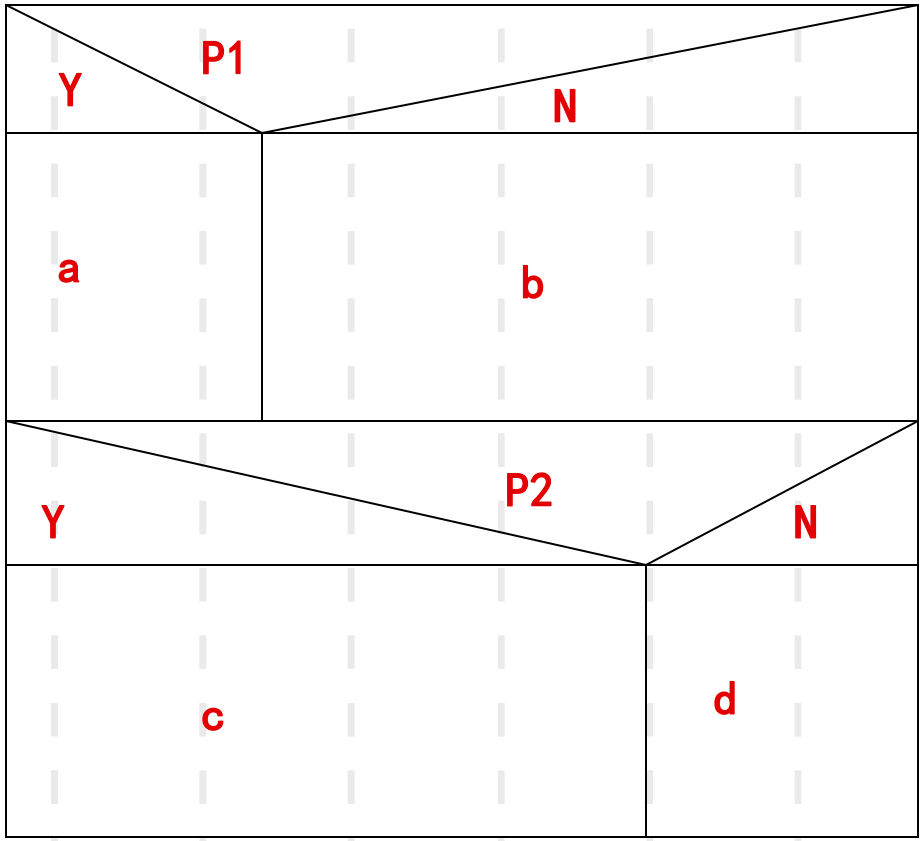

DOWHILE型




DOUNTIL型

# 最少测试用例计算

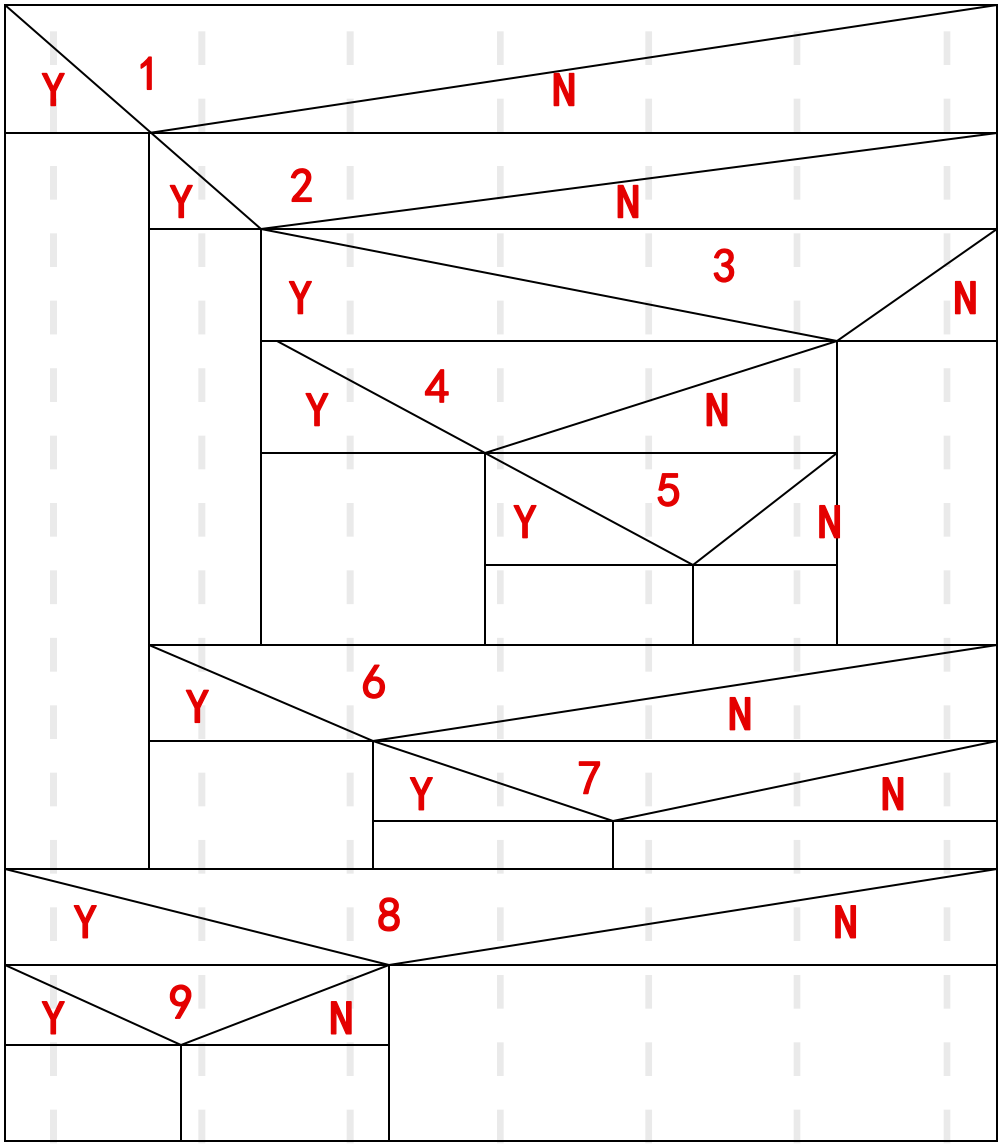
## 例1：两个串行分支结构的N-S图




最少测试用例

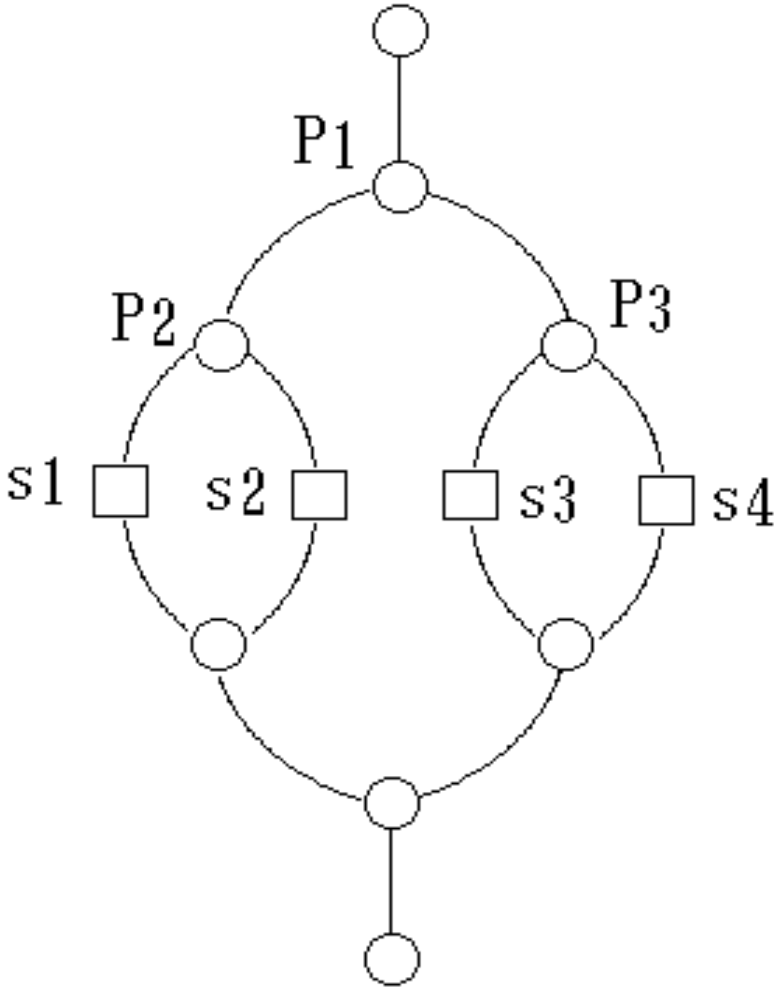
2×2＝4

# 例2：最少测试用例计算

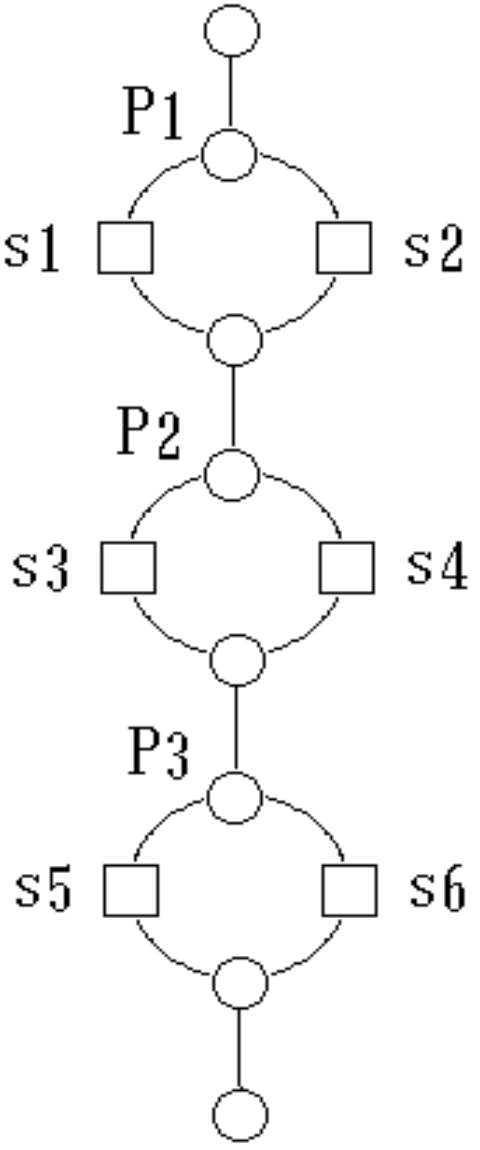


# 条件测试路径选择

- 当程序中判定多于一个时，形成的分支结构可以分为两类：嵌套型分支结构和连锁型分支结构。
- 对于嵌套型分支结构，若有$n$个判定语句，需要$n+1$个测试用例；
- 对于连锁型分支结构， 若有$n$个判定语句，需要有$2^n$个测试用例，覆盖它的$2^n$条路径。当$n$较大时将无法测试。




（a）嵌套型分支结构




（b）连锁型分支结构

# §3　路经分析

- 着眼于路径分析的测试为路径测试；
- 完成路经测试的理想情况时做到路径覆盖；
- 一、路径表达式和路径数
- 可用弧序列或节点序列表示某一条具体路径。给出路径通式时用弧序列表示。
- 1、路径表达式
- 路径表达式有两个运算符：相乘和相加

# §3　路经分析

- （1）弧a和弧b相乘，表示为ab，它表明路径是实现经历弧a接着再经历弧b，弧a和弧b是先后相继的。
- （2）弧a和弧b相加，表示为a+b，它表明弧a和弧b是并立的（或的关系），即两条路径是并行关系。
- 举例：pg53

# §3　路经分析

- （3）运算规律
- 加法交换律
- 加法结合律
- 加法幂等律
- 乘法结合律
- 分配律

- 对路径表达式化简

# §3　路经分析

- 2、路经数计算
- 将路径表达式中的弧都用数字1代入，然后，按照相乘、湘价计算出数值，即为该程序的路径数。

- 注：程序的圈复杂度和路径数有着直接的关系

# §3　路经分析

- 二、基本路经测试
- 如果一个程序的独立路径都被测试过，那么可以认为程序中的每个语句都以检验过了，也就达到了语句覆盖，这种方式就是基本路经测试方法。
- 独立路经测试——基本路经测试

复习：独立路径的概念

# §3　路经分析

- 举例：
- 要求：最多输入100个值（以-999为输入结束标志），计算落在给定范围内的那些值（成为有效输入值）的个数、总和和平均值。
- 下面的程序片断为主程序调用的求平均值的函数average()，变量sum为总和，total为有效值的个数，两者都为全局变量。
- 主程序main()完成数据输入（调用时传给value数组）及平均值、总和、有效值个数的输出。

# §3　路经分析

- 具体步骤
- （1）根据程序给出流图；
- （2）确定圈复杂性度量V(G)；
- （3）确定独立路径集；
- （4）为每个独立路径的执行，设计测试用例

# §3 路径分析

分析：确定每个节点代表的程序

复习while 语句的流图框架

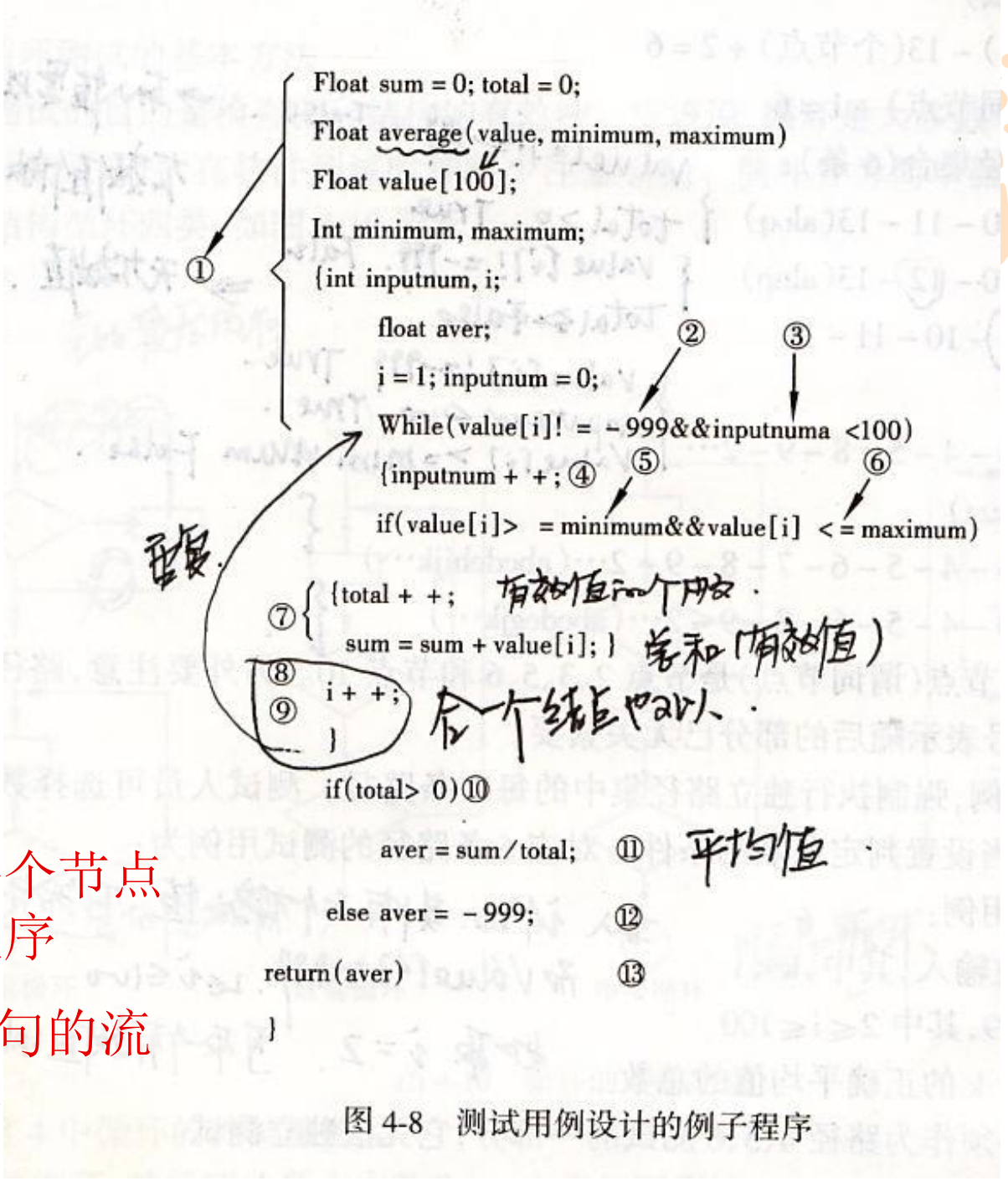



图 4-8 测试用例设计的例子程序

# §3 路径分析

## （1）流图

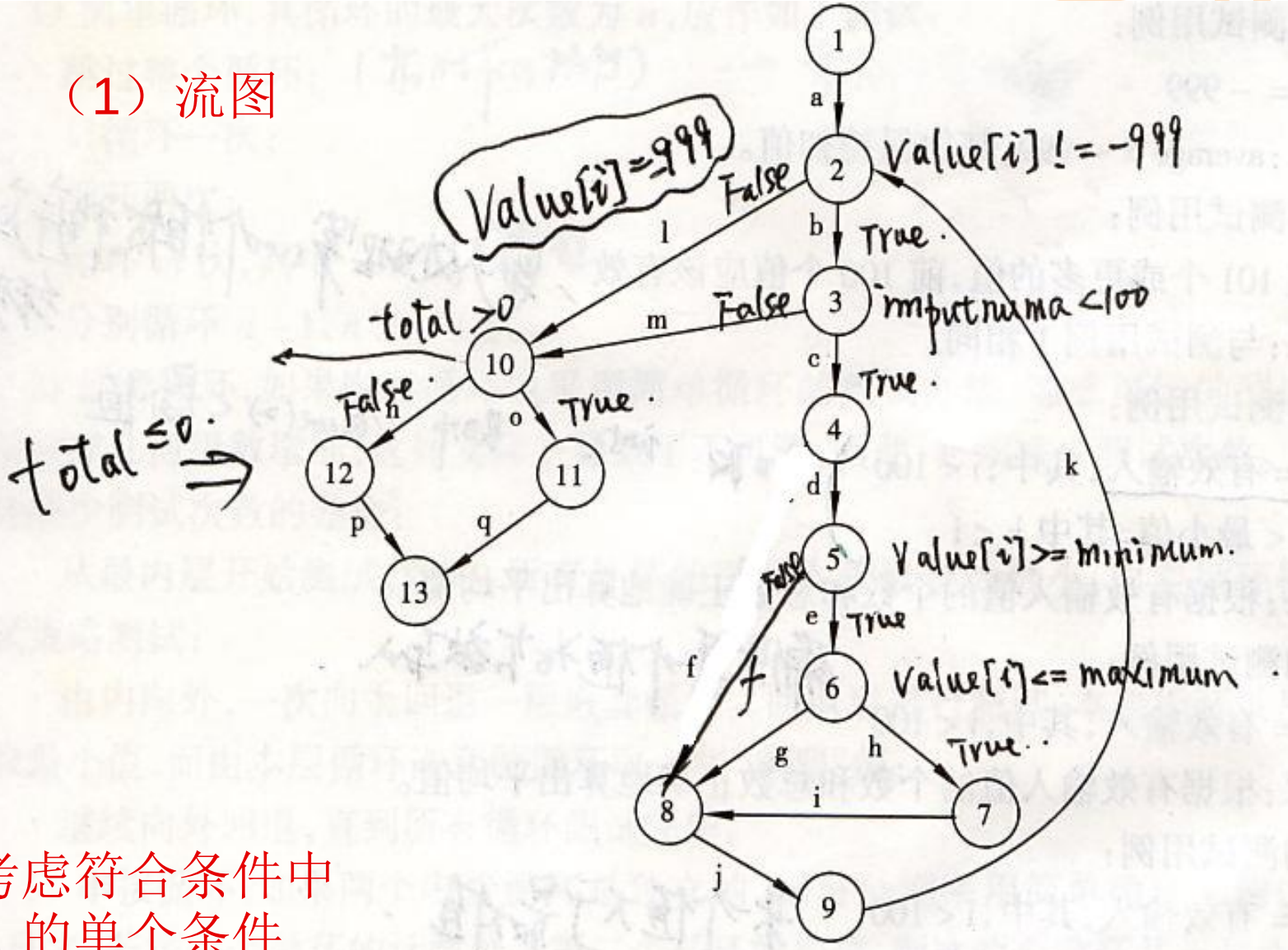



图 4-9　过程求平衡值的流图

考虑符合条件中的单个条件

# §3　路经分析

- （2）圈复杂性V(G)
- V(G)=6　　(围成的区域数)
- V(G)=17（边）-13（节点） +2=6
- V(G)=5（判定节点）+1=6

# §3　路经分析

- （3）六条独立路径
- 路径1　　1-2-10-11-13（aloq）
- 路径2　　1-2-10-12-13
- 路径3　　1-2-3-10-11-13
- 路径4　　1-2-3-4-5-8-9-2……
- 路径5　　1-2-3-4-5-6-7-8-9-2…..
- 路径6　　1-2-3-4-5-6-8-9-2……

# §3　路经分析

- （4）设计测试用例
- 路径1　　1-2-10-11-13（aloq）
- 执行的条件情况：
- 节点2，条件value(I)!=-999为假；
  程序结束以输入-999退出while 循环
- 节点10，条件total>0为真
  要求有有效的输入数据。
  假如最大值为maximum，如150；
  　　最小值为minimum，如22

# §3　路经分析

- 可如下设计一组测试用例数据，如5个数，分别是：12，26，58，160，-999
- 期望结果是：

  平均值：42

  总和：84

  有效值个数：2
- 通用的测试用例表示：
- Value(k)=有效输入，k<I
- Value(i)=-999，2<=I<=100

输入I个值，I大于2，小于100。最后一个值value(I)为-999,I个值中一定有有效值，如，第k个值。

# §3　路经分析

- 路径2　1-2-10-12-13
- 执行的条件情况：
- 节点2，条件value(l)!=-999为假；
  程序结束以输入-999退出while 循环
- 节点10，条件total>0为false，即输入的一组数据中没有一个是有效值。
  假如最大值为maximum，如100；
  　　最小值为minimum，如22

# §3　路经分析

- 可如下设计一组测试用例数据，如5个数，分别是：12，21，158，160，-999
- 期望结果是：

  平均值：-999

  总和：0

  有效值个数：0
- 通用的测试用例表示：
- 任何Value(k)都为无效输入，k<I
- Value(i)=-999，2<=I<=100

输入I个值，I大于2，小于100。最后一个值value(I)为-999,I个值中没有一个有效值。

# §3　路经分析

- 路径3　1-2-3-10-11-13
- 执行的条件情况：
- 节点2，条件value(I)!=-999为真；
  程序以输入-999退出while 循环
- 节点10，条件total>0为真，即输入的一组数据中至有一个是有效值。
- 节点3，inputnum<100为假，程序以输入数据超过100个，退出while 循环。
  假如最大值为maximum，如150；
  　　最小值为minimum，如22。

# §3　路经分析

- 测试用例设计：
- 试图输入101个数或更多，数中没有-999，前100个数中至少有一个是有效数。
- 期望结果：计算100个之后，程序应结束，期望的平均值、有效值总数、平均值根据前100个值中的有效值计算。

# §3　路经分析

- 课堂作业(小测验)：
- 假如输入个数最多为10个，请为本题的独立路径4、5、6涉及具体的测试用例，并给出期望结果。

# §3 路经分析

## 三、循环测试路径选择

- 循环分为4种不同类型：简单循环、连锁循环、嵌套循环和非结构循环。

(1) 简单循环

① 零次循环：从循环入口到出口

② 一次循环：检查循环初始值

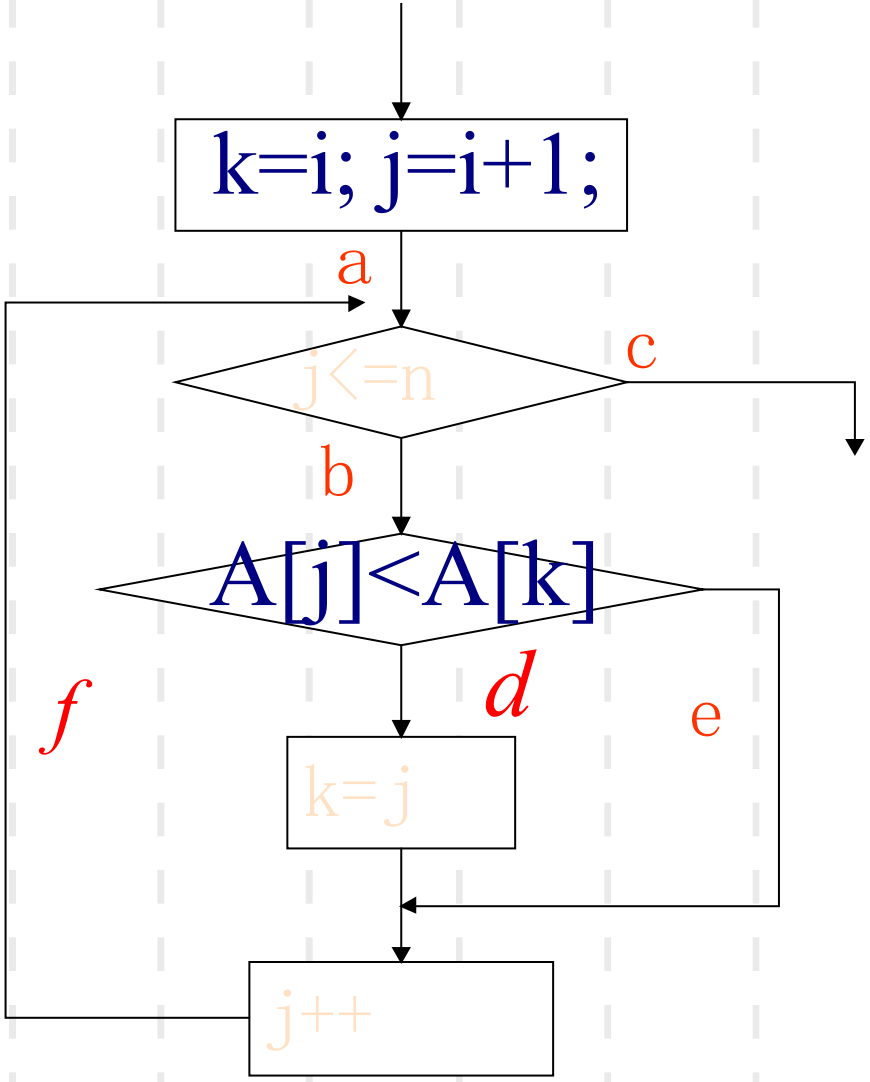
③ 二次循环：检查多次循环

④ *m*次循环：检查在多次循环

⑤ 最大次数循环、比最大次数多一次、少一次的循环。

# 例：求最小值

```
k = i;
    ❶
for ( j = i+1;  j <= n;  j++ )
        ❷          ❸       ❻
   if ( A[j] < A[k] ) then  k = j;
            ❹                ❺
```


k=i; j=i+1;
a
j<=n
c
b
A[j]<A[k]
f
d
e
k= j
j++

# 测试用例选择

| 循环 | i | n | A[i] | A[i+1] | A[i+2] | k | 路 径 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | | | | i | ac |
| 1 | 1 | 2 | 1 | 2 | | i | abefc |
| | | | 2 | 1 | | i+1 | abdfc |
| 2 | 1 | 3 | 1 | 2 | 3 | i | abefefc |
| | | | 2 | 3 | 1 | i+2 | abefdfc |
| | | | 3 | 2 | 1 | i+2 | abdfdfc |
| | | | 3 | 1 | 2 | i+1 | abdfefc |

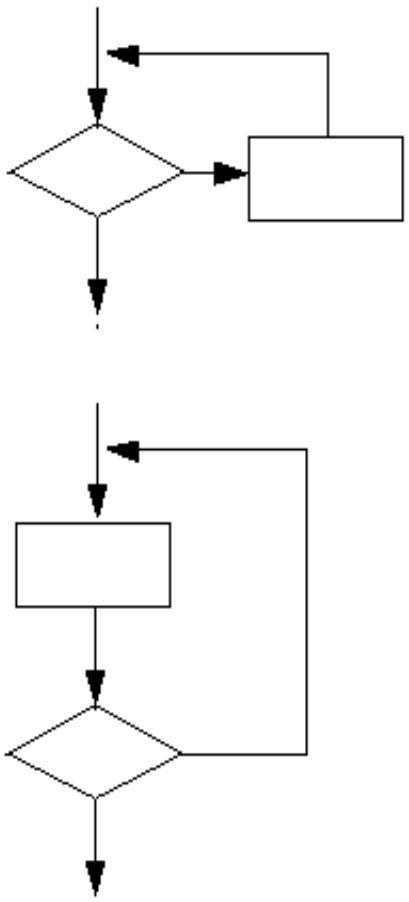
d 改 k 的值，e 不改 k 的值

## (2) 嵌套循环

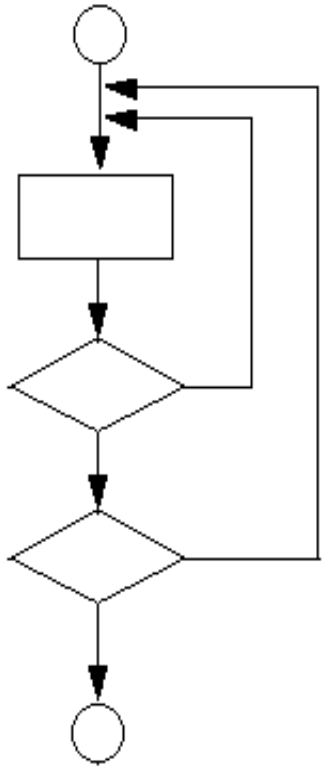
① 对最内层循环做简单循环的全部测试。所有其它层的循环变量置为最小值；

② 逐步外推，对其外面一层循环进行测试。测试时保持所有外层循环的循环变量取最小值，所有其它嵌套内层循环的循环变量取“典型”值。。

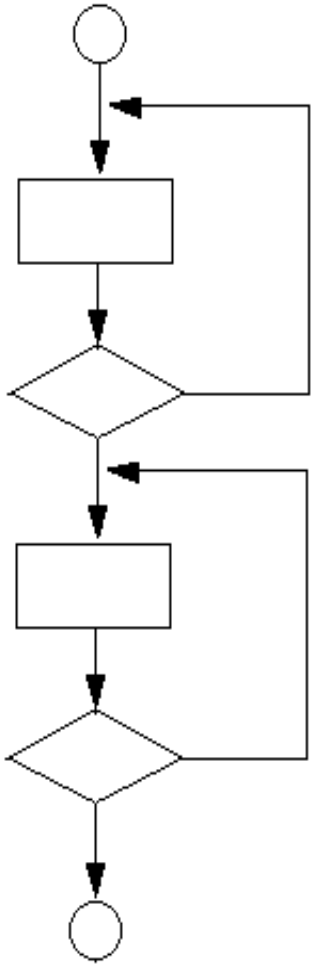
③ 反复进行，直到所有各层循环测试完毕。



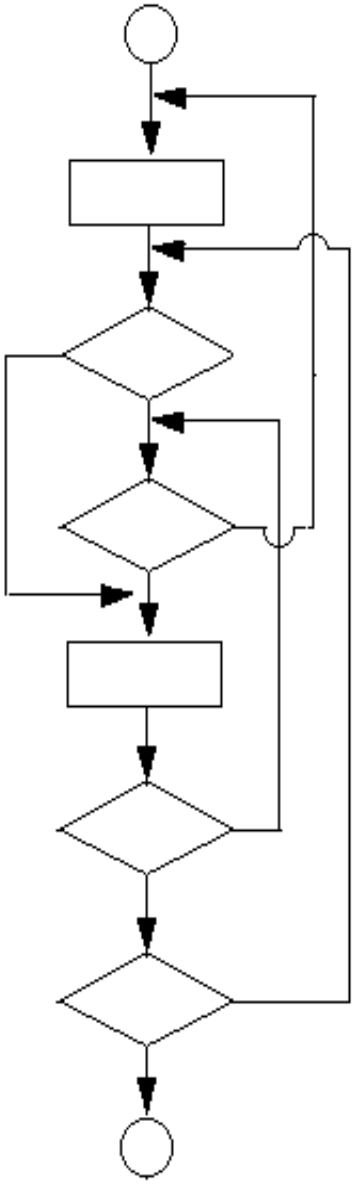
简单循环



嵌套循环



连锁循环



非结构循环

④ 对全部各层循环同时取最小循环次数，或者同时取最大循环次数

(3) 连锁循环

如果各个循环互相独立，则可以用与简单循环相同的方法进行测试。但如果几个循环不是互相独立的，则需要使用测试嵌套循环的办法来处理。

(4) 非结构循环

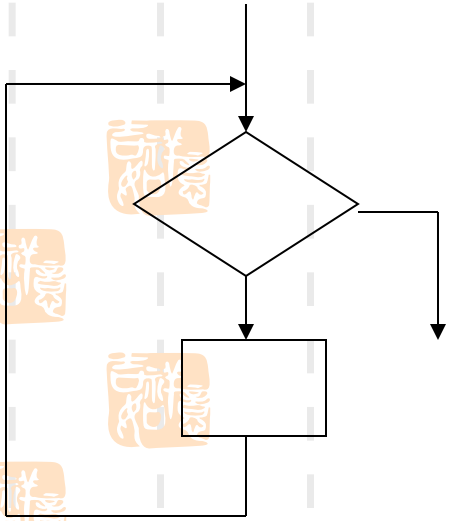
这一类循环应该使用结构化程序设计方法重新设计测试用例。

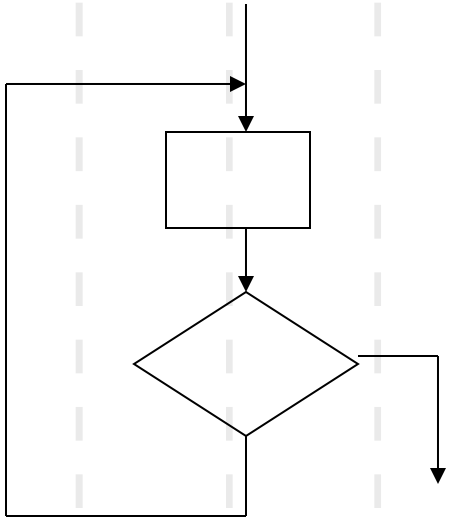
# Z路径覆盖下的循环测试策略

简化循环意义下的路径覆盖称为Z路径覆盖

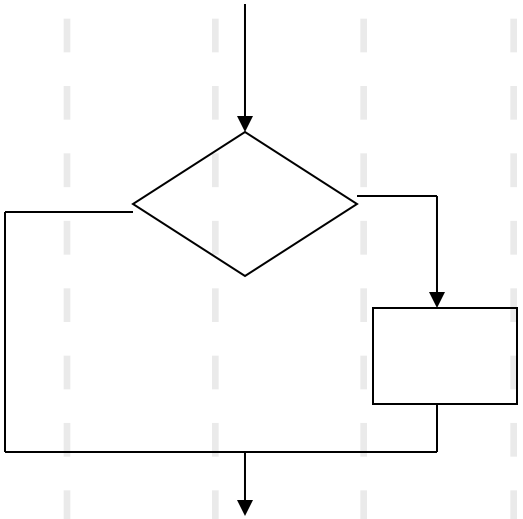
循环简化的含义是限制循环的次数。无论循环的形式和循环体实际执行的次数，简化后的循环测试只考虑执行循环体1次和0次（不执行）两种情况。

一旦循环结构简化为选择结构后，路径的数量将大大减少，通过枚举的办法得到所有的路径是完全有可能的。





两种典型循环结构



简化后的选择结构

例：输入一行字符，分别统计出其中英文字母、空格、数字和其他字符的个数。程序如下，请按路径分析方法设计测试用例。

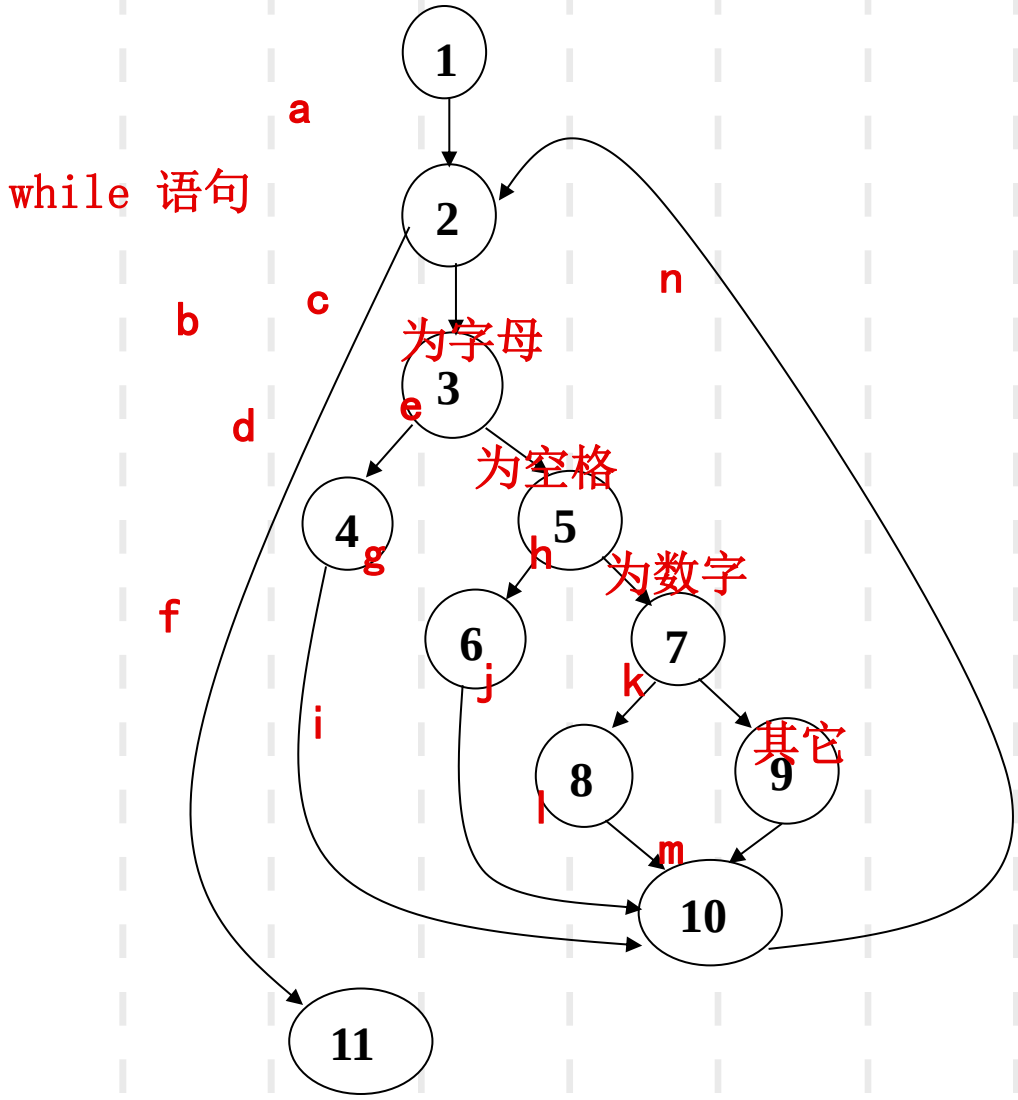
要求：1、画出流图；2、分析复杂性；3、给出独立路径；4、设计测试用例。

```
#include”stdio.h”
main()
{
    char c;
    int letters=0,space=0,digit=0,other=0;
     printf(“请输入一行字符：\n”);
     while((c=getchar())!=’\n’)
     {
        if(c>=’a’&&c<=’z’||c>=’A’&&c<=’Z’)
            letters++;
        else if(c==’ ’)
            space++;
        else if(c>=’0’&&c<=’9’)
            digit++;
        else
       other++
     }
                printf(“其中：字母数=%d空格数=%d 数字数=%d 其它字符数
                    =%d\n”,letters,space,digit,other);
}
```

1、流图；

```
#include”stdio.h”
main()
{
    char c;
    int letters=0,space=0,digit=0,other=0;
    printf(“请输入一行字符：\n”);
    while((c=getchar())!=’\n’)
    {
        if(c>=’a’&&c<=’z’||c>=’A’&&c<=’Z’)
            letters++;
        else if(c==’ ’)
            space++;
        else if(c>=’0’&&c<=’9’)
            digit++;
        else
            other++
    }
    printf(“其中：字母数=%d空格数=%d 数字数=%d 其它字符数
        =%d\n”,letters,space,digit,other);
}
```

1 2 3 4 5 6 7 8 9 10 11


while 语句
1
a
2
b
c
n
为字母
3
d
e
为空格
4
5
g
h
为数字
f
6
7
j
k
i
其它
8
9
l
m
10
11

（2）复杂性为：5

（3）独立路径为：

①ab

②acdfn.....

③acegin.....

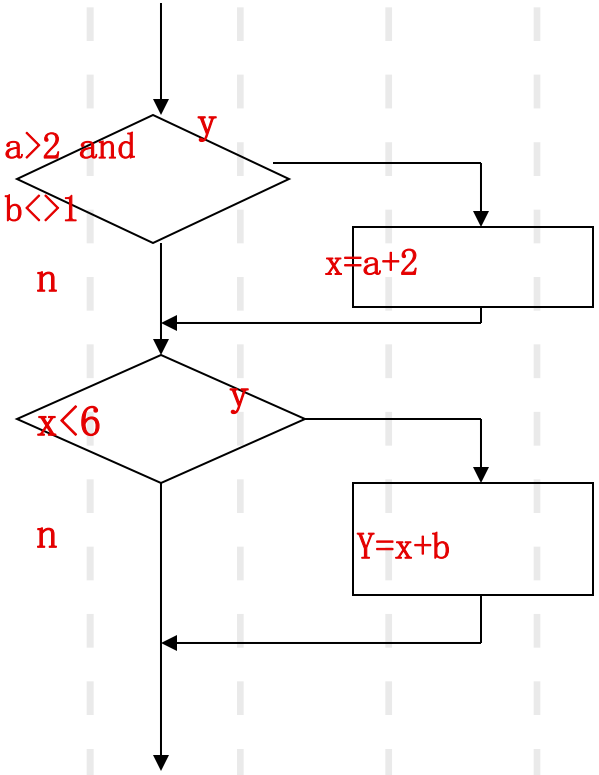
④acehjln.....

⑤acehkmn.....

（4）测试用例为

| 序号 | 测试用例 | 覆盖独立路径 | 预期结果 | 备注 |
|---|---|---|---|---|
| 1 | 输入空串 | ab | 输出结果都为0 | |
| 2 | 输入“ab.....” | acdfn..... | 正确给出字符个数 | |
| 3 | 输入“ a.....” | acegin..... | 正确给出空格个数 | |
| 4 | 输入“8.....” | acehjln..... | 正确给出数字个数 | |
| 5 | 输入“#.....” | acehkmn..... | 正确给出其他字符个数 | |

给出如下一段程序，按照条件覆盖、分支覆盖、条件/分支覆盖、条件组合覆盖给出测试用例。



（1）分支覆盖

| x | a | b | a>2 and b<>1 | x<6 | 执行路径 |
|---|---|---|---|---|---|
| 3 | 3 | 2 | T | T | |
| 6 | 2 | 1 | F | F | |

（2）条件覆盖

| x | a | b | a>2 | b<>1 | x<6 | 执行路径 |
|---|---|---|---|---|---|---|
| 3 | 3 | 2 | T | T | T | |
| 6 | 2 | 1 | F | F | F | |

（3）条件/分支覆盖

| x | a | b | a>2 and b<>1 | x<6 | a>2 | b<>1 | 执行路径 |
|---|---|---|---|---|---|---|---|
| 3 | 3 | 2 | T | T | T | T | |
| 6 | 2 | 1 | F | F | F | F | |

（4）条件组合覆盖

a>2 记t1 b<>1 记t2 x<6 记t3

每个判定可能的条件有：(1)t1 t2,(2)t1 ～t2,(3)～t1 t2,(4)～t1 ～t2 (5)t3 (6)～t3

设计4个测试用例，覆盖以上6种条件组合

| 测试用例 | x | a | b | 覆盖组合 | 覆盖条件 | 执行路径 |
|---|---|---|---|---|---|---|
| 1 | 6 | 3 | 2 | 1, 5 | t1 t2 t3 | |
| 2 | 4 | 3 | 1 | 2, 5 | t1～t2t3 | |
| 3 | 7 | 2 | 2 | 3, 6 | ～t1t2<br>～t3 | |
| 4 | 7 | 2 | 1 | 4, 6 | ～t1～t2<br>～t3 | |

# 四、路径分析
# 基本路径测试归纳

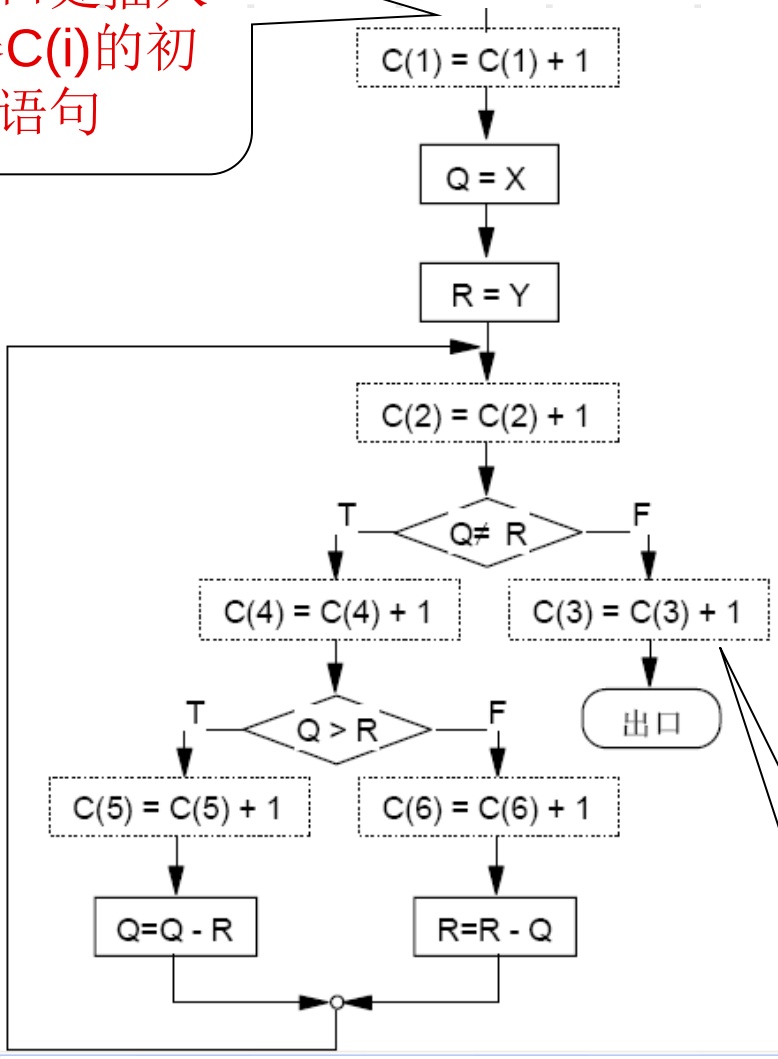
- 基本路径测试方法把覆盖的路径数压缩到一定限度内，程序中的循环体最多只执行一次——独立路径覆盖。
- 它是在程序控制流图的基础上，分析控制构造的环路复杂性，导出基本可执行路径集合，设计测试用例的方法。设计出的测试用例要保证在测试中，程序的每一个可执行语句至少要执行一次。

# §4　程序插桩

- 程序插桩（program instrumentation）是一种基本的测试手段，在软件测试中有着广泛的作用。
- 一、方法简介
- 程序插桩方法简单的说是借助于往被测程序中插入操作来实现测试目的的方法。
- 以计算整数X和Y的最大公约数程序为例，说明插桩方法的要点。

# § 4 程序插桩

在程序入口处插入对计数器C(i)的初始化语句




在程序出口处加入打印语句

图中的虚线框并不是原来的内容，而是为了纪录语句执行次数而插入的。这些虚线框要完成的工作都是计数语句。

C(i)=C(i)+1

即构成完整的插桩程序

# §4 程序插桩

1、2需结合具体的问题解决

- 二、设计程序插桩时序考虑的问题
- （1）探测哪些信息？
- （2）在程序的那些地方设置探测点；
- （3）需要设置那些探测点。

可根据控制结构考虑

在没有分支结构的程序段只需一个插桩语句

# §4 程序插桩

这里我们以FORTRAN程序为例，列举至少应在哪些部位设置计数语句：

① 程序块的第一个可执行语句之前；

② ENTRY语句的前后；

③ 有标号的可执行语句处；

④ DO、DO WHILE、DO UNTIL及DO终端语句之后；

⑤ BLOCK-IF、ELSE IF、ELSE及ENDIF语句之后；

⑥ LOGICAL IF语句处；

⑦ 输入/输出语句之后；

⑧ CALL语句之后；

⑨ 计算GO TO语句之后。

# § 4　程序插桩

## 三、断言（assertions）语句

在程序中的特定部位插入某些用以判断变量特性的语句，使得程序执行中这些语句得以证实，从而使程序的运行特性得到证实。我们把插入的这些语句称为断言

这里仅对断言做一个简单的说明，以后我们还要进一步讨论

FLOYD的归纳断言法。这里仅以求两个非负数NUM和DEN之商的Wensley迭代算法为例，对断言语句的作用作一简要说明。

假定两个非负数中，NUM小于M（即所得之商小于1），算法中只用到加、减及除2的运算。该迭代算法的程序如图13所示。

# § 4 程序插桩

```
procedure DIVIDE (NUM, DEN, E, Q)
*  E is the accuracy required. E≥ 0. Q is both        *
*  the result at exit and at any interim stage.       *
*  A.  B and W are the other elements of the pro-*
*  gram vector.
      Q: = 0
      A: = 0
      B: = DEN/2
      W:=1
      until W < E loop
         if (NUM-A-B) ≥ 0
         then
             Q: = Q + W/2
              A: = A + B
         endif
         B: =B/2
         W:=W/2
      endloop
      end
```

图13 计算两非负数之商的迭代程序

从程序中可以看出，在每次迭代中由分母得到的变量B以及权增量W都要缩小一半，而且变量A随着迭代次数的增加将接近分子。

# § 4 程序插桩

粗略的观测和分析可以用一下

4个断言语句表达，在每次迭代开始时4个断言必定为真：

① $W = 2^{-K}$（$K$是迭代次数，并且$K \bullet 0$；

② A = DEN * Q;

③ B = DEN * W/2;

④ $NUM/DEN - W < Q \bullet NUM/DEN$。

此外，我们还看出，在循环外W<E，而且结果Q总是从下面逼近真正的商。这就得到了输出断言：

$$NUM/DEN - E < Q \bullet NUM/DEN$$

它和上面的第④断言很相似。

假定我们所用的编译系统能够处理表达式形式的断言语句，插入断言以后的程序如：

# § 4 程序插桩

```
  procedure DIVIDE (NUM, DEN, E, Q)
*  E is the required accuracy. E≥ 0. Q is both        *
*  the result at exit and at any interim stage.       *
*  A.  B and W are the other elements of the pro-*
*  gram vector.
        Q: = 0
        A: = 0
        B: = DEN/2
        W:=1
@       K: =0
        until K < E loop
@       assert W = 1/2**K
@       assert A = DEN*Q
@       assert B = DEN*W/2
@       assert NUM/DEN - W<Q and Q≤ NUM/DEN
        if (NUM-A-B) ≥ 0
        then
                Q: = Q + W/2
                A: = A + B
        endif
        B: =B/2
        W:=W/2
@       K: = K+1
        endloop
@       assert NUM/DEN - E<Q and Q≤ NUM/DEN
        end
```

图14 插入断言后的迭代程序

图14所示。其中带有标记@的语句是断言语句。新增加的变量K只是在计算第①断言时用到。

# § 4　程序插桩

首先来检验在初始化以后循环后的断言。

① 由于 K = 0，所以$W = 2^{-K} = 1$是初值。

② 由于Q = 0，A的初始 A = DEN * Q = 0。

③ 将W的值代入DEN * W/2，则得B的初值 B = DEN/2。

④ 我们曾假定 0 ≤ NUM〈 DEN，NUM/DEN - 1（W的值）必定小于零，因而也就小于Q（它是零）。而且，NUM/DEN必定大于Q，因为NUM和DEN均为正，Q为零。

以上说明了这些断言在初始状态下为真。如果继续迭代，要证明断言为真，就必须证明无论ir-then结构中执行什么路径这些断言都是真。让我们先来考虑，在初始测试中NUM - A - B≥0为假，即检验失败。然后给出程序向量的新值（A’，B’，W’，Q’和K’），我们有：

# § 4 程序插桩

$$A^{\bullet} = A$$

$$B^{\bullet} = B/2$$

$$W^{\bullet} = W/2$$

$$Q^{\bullet} = Q$$

$$K^{\bullet} = K+1$$

再来检验4个断言：

① $W^{\bullet} = W/2 = 1/2 \bullet \bullet K^{\bullet}$

② $A^{\bullet} = A = DEN \bullet Q = DEN \bullet Q^{\bullet}$

③ $B^{\bullet} = B/2 = DEN \bullet W/4 = DEN \bullet W^{\bullet}/2$

④ 把$A$和$B$代入（$NUM - A - B < 0$），得到$NUM - DEN \bullet Q - DEN \bullet W/2 < 0$，对此关系式两端除以DEN，并加Q，得到$NUM/DEN - W/2 < Q$，由于Q’ =Q，W’ =W/2，我们有$NUM/DEN - W^{\bullet} < Q^{\bullet}$，且$Q^{\bullet} \bullet NUM/DEN$。

# § 4　程序插桩

如果if-then检验成立，再来看4个断言。使用A″，B″，W″，Q″和K″作为新的程序向量，我们有：

① $W' = W/2 = 1/2 \cdot \cdot K''$

② $A'' = DEN \cdot Q + DEN \cdot W/2 = DEN \cdot (Q + W/2) = DEN \cdot Q''$

③ $B'' = B/2 = DEN \cdot W/4 = DEN \cdot W''/2$

④ 代入（NUM-A-B≥0），并作同前的变换，得到NUM/DEN - W″/2＜Q″ + W″/2或：

NUM/DEN - W″＜Q″，并且Q″≤NUM/DEN。

总之，无论执行哪一路径，在每一迭代的开始，4个断言均为真。尽管并未考虑输出断言，但是我们知道，第④断言成立，由于W<E，NUM/DEN - E< Q和Q ≤NUM/DEN必定为真，也就必定满足输出断言。

# §5 界面测试考虑

- 目前流行的界面风格有三种方式：多窗体、单窗体以及资源管理器风格，无论那种风格，以下规则是应该被重视的。

1：易用性 2： 规范性 3：帮助设施

4：合理性 5：美观与协调性 6：菜单位置

7:独特性 8：快捷方式的组合

9:安全性考虑 10:多窗口的应用与系统资源

# §5 界面测试考虑

- 1：易用性
- 按钮名称应该易懂，用词准确，要与同一界面上的其他按钮易于区分，能望文知意最好。理想的情况是用户不用查阅帮助就能知道该界面的功能并进行相关的正确操作。
- 易用性细则：
- 1):完成相同或相近功能的按钮用Frame框起来，常用按钮要支持快捷方式。
- 2):完成同一功能或任务的元素放在集中位置，减少鼠标移动的距离。
- 3):按功能将界面划分区域块，用Frame框括起来,并要有功能说明或标题。

# §5 界面测试考虑

- 4):界面要支持键盘自动浏览按钮功能，即按Tab键、回車鍵的自动切换功能。
- 5):界面上首先要输入的和重要信息的控件在Tab顺序中应当靠前,位置也应放在窗口上较醒目的位置。
- 6):同一界面上的控件数最好不要超过10个，多于10个时可以考虑使用分页界面显示。
- 7):分页界面要支持在页面间的快捷切换，常用组合快捷键Ctrl+Tab
- 8):默认按钮要支持Enter及选操作，即按Enter后自动执行默认按钮对应操作。

# §5 界面测试考虑

- 9):可寫控制項檢測到非法輸入後應給出說明並能自動獲得焦點。
- 10):Tab键的顺序与控件排列顺序要一致，目前流行总体从上到下，同时行间从左到右的方式。
- 11):核取方塊和選項框按選擇幾率的高底而先後排列。
- 12):核取方塊和選項框要有默認選項，並支援Tab選擇。
- 13):選項數相同時多用選項框而不用下拉清單框。

# §5 界面测试考虑

- 14):界面空间较小时使用下拉框而不用选项框。
- 15):选项数较少时使用选项框，相反使用下拉列表框。
- 16):专业性强的软件要使用相关的专业术语，通用性界面则提倡使用通用性词语。

# §5 界面测试考虑

- 2： 规范性
- 通常界面设计都按Windows界面的规范来设计，可以说：界面遵循规范化的程度越高，则易用性相应的就越好。小型软件一般不提供工具厢。
- 规范性细则：
- 1):常用菜单要有命令快捷方式。
- 2):完成相同或相近功能的菜单用横线隔开放在同一位置。
- 3):菜单前的图标能直观的代表要完成的操作。
- 4):菜单深度一般要求最多控制在三层以内。

# §5 界面测试考虑

- 5):工具栏要求可以根据用户的要求自己选择定制。
- 6):相同或相近功能的工具栏放在一起。
- 7):工具栏中的每一个按钮要有及时提示信息。
- 8):一条工具栏的长度最长不能超出屏幕宽度。
- 9): 工具栏的图标能直观的代表要完成的操作。
- 10):系统常用的工具栏设置默认放置位置。
- 11):工具栏太多时可以考虑使用工具箱。

# §5 界面测试考虑

- 12):工具箱要具有可增减性，由用户自己根据需求定制。
- 13):工具箱的默认总宽度不要超过屏幕宽度的1/5。
- 14): 状态条要能显示用户切实需要的信息，常用的有：
- 目前的操作、系统状态、用户位置、用户信息、提示信息、错误信息等，如果某一操作需要的时间较长，还应该显示进度条和进程提示。
- 15): 滚动条的长度要根据显示信息的长度或宽度能及时变换，以利于用户了解显示信息的位置和百分比。

# §5 界面测试考虑

- 16)：状态条的高度以放置五号字为宜，滚动条的宽度比状态条的略窄。
- 17)：菜单和工具条要有清楚的界限;菜单要求凸出显示，这样在移走工具条时仍有立体感。
- 18)：菜单和状态条中通常使用5号字体。工具条一般比菜单要宽，但不要宽的太多，否则看起来很不协调。
- 19)：右键快捷菜单采用与菜单相同的准则。

# §5 界面测试考虑

- 3：帮助设施：
- 系统应该提供详尽而可靠的帮助文档，在用户使用产生迷惑时可以自己寻求解决方法。
- 帮助设施细则：
- 1)：帮助文档中的性能介绍与说明要与系统性能配套一致。(我们的系统帮助文档都是系统的祖先时期的说明，让人困惑)。
- 2)：打包新系统时，对作了修改的地方在帮助文档中要做相应的修改。
- 3)：操作时要提供及时调用系统帮助的功能。常用F1。

# §5 界面测试考虑

- 4)：在界面上调用帮助时应该能够及时定位到与该操作相对的帮助位置。也就是说帮助要有即时针对性。
- 5)：最好提供目前流行的联机帮助格式或HTML帮助格式。
- 6)：用户可以用关键词在帮助索引中搜索所要的帮助，当然也应该提供帮助主题词。
- 7)：如果没有提供书面的帮助文档的话，最好有打印帮助的功能。
- 8)：在帮助中应该提供我们的技术支持方式，一旦用户难以自己解决可以方便的寻求新的帮助方式。

# §5 界面测试考虑

- 4：合理性：
- 屏幕对角线相交的位置是用户直视的地方，正上方四分之一处为易吸引用户注意力的位置，在放置窗体时要注意利用这两个位置。
- 合理性细则：
- 1)：父窗体或主窗体的中心位置应该在对角线焦点附近。
- 2)：子窗体位置应该在主窗体的左上角或正中。
- 3)：多个子窗体弹出时应该依次向右下方偏移，以显示窗体出标题为宜。

# §5 界面测试考虑

4)：重要的命令按钮与使用较频繁的按钮要放在界面上注目的位置。

5)：错误使用容易引起界面退出或关闭的按钮不应该放在易点击的位置。横排开头或最后与竖排最后为易点位置。

6)：与正在进行的操作无关的按钮应该加以屏蔽(Windows中用灰色显示，没法使用该按钮)。

7)：对可能造成数据无法恢复的操作必须提供确认信息,给用户放弃选择的机会。

# §5 界面测试考虑

8)：非法的输入或操作应有足够的提示说明。

9)：对运行过程中出现问题而引起错误的地方要有提示，让用户明白错误出处，避免形成无限期的等待。

10)：提示、警告、或错误说明应该清楚、明了、恰当。

# §5 界面测试考虑

- 5：美观与协调性：
- 界面应该大小适合美学观点，感觉协调舒适，能在有效的范围内吸引用户的注意力。
- 美观与协调性细则：
- 1)：长宽接近黄金点比例，切忌长宽比例失调、或宽度超过长度。
- 2)：布局要合理,不宜过于密集，也不能过于空旷，合理的利用空间。
- 3)：按钮大小基本相近，忌用太长的名称，免得占用过多的界面位置。

# §5 界面测试考虑

- 4)：按钮的大小要与界面的大小和空间要协调。
- 5)：避免空旷的界面上放置很大的按钮。
- 6)：放置完控件后界面不应有很大的空缺位置。
- 7)：字体的大小要与界面的大小比例协调，通常使用的字体中宋体9-12较为美观，很少使用超过12号的字体。
- 8)：前景与背景色搭配合理协调，反差不宜太大，最好少用深色，如大红、大绿等。常用色考虑使用Windows界面色调。

# §5 界面测试考虑

- 9)：如果使用其他颜色，主色调要柔和，具有亲和力与磁力，坚决杜绝刺目的颜色。
- 10)：大型系统常用的主色有"#E1E1E1"、"#EFEFEF"、"#C0C0C0"等。
- 11)：界面风格要保持一致，字的大小、颜色、字体要相同，除非是需要艺术处理或有特殊要求的地方。
- 12)：如果窗体支持最小化和最大化或放大时，窗体上的控件也要随着窗体而缩放；切忌只放大窗体而忽略控件的缩放。

# §5　界面测试考虑

- 13)：对于含有按钮的界面一般不应该支持缩放，即右上角只有关闭功能。
- 14)：通常父窗体支持缩放时，子窗体没有必要缩放。
- 15)：如果能给用户提供自定义界面风格则更好，由用户自己选择颜色、字体等。

# §5 界面测试考虑

- 6：菜单位置：
- 菜单是界面上最重要的元素，菜单位置按照按功能来组织。
- 菜单测试细则：
- 1)：菜单通常采用“常用--主要--次要--工具--帮助”的位置排列，符合流行的Windows风格。
- 2)：常用的有“文件”、“編輯”，“查看”等，幾乎每個系統都有這些選項，當然要根據不同的系統有所取捨。
- 3)：下拉菜单要根据菜单选项的含义进行分组，並且按照一定的规则进行排列，用横线隔开。

# §5 界面测试考虑

- 4)：一组菜单的使用有先后要求或有向导作用时，应该按先后次序排列。
- 5)：没有顺序要求的菜单项按使用频率和重要性排列，常用的放在开头， 不常用的靠后放置；重要的放在开头，次要的放在后边。
- 6)：如果菜单选项较多，应该采用加长菜单的长度而减少深度的原则排列。
- 7)：菜单深度一般要求最多控制在三层以内。
- 8)：对常用的菜单要有快捷命令方式，组合原则见8。
- 9)：对与进行的操作无关的菜单要用屏蔽的方式加以处理，如果采用动态加载方式——即只有需要的菜单才显示——最好。

# §5 界面测试考虑

- 10)：菜单前的图标不宜太大，与字高保持一直最好。
- 11)：主菜单的宽度要接近，字数不应多于四个，每个菜单的字数能相同最好。
- 12)：主菜单数目不应太多，最好为单排布置。
- 13):菜单条是否显示在合适的语境中？
- 14):应用程序的菜单条是否显示系统相关的特性（如时钟显示）？
- 15):下拉式操作能正确工作吗？
- 16):菜单、调色板和工具条是否工作正确？

# §5 界面测试考虑

- 17):是否适当地列出了所有的菜单功能和下拉式子功能?
- 18):是否可能通过鼠标访问所有的菜单功能?
- 19):相同功能按钮的图标和文字是否一致?
- 20):是否能够用其他的文本命令激活每个菜单功能?
- 21):菜单功能是否随当前的窗口操作加亮或变灰?
- 22):菜单功能是否正确执行?
- 23):菜单功能的名字是否具有自解释性?

# §5 界面测试考虑

- 24):菜单项是否有帮助，是否语境相关?
- 25):在整个交互式语境中，是否可以识别鼠标操作?
- 26):如果要求多次点击鼠标，是否能够在语境正确识别?
- 27):如果鼠标有多个按钮，是否能够在语境中正确识别?
- 28):光标、处理指示器和识别指针是否随操作恰当地改变?

# §5 界面测试考虑

- 7:独特性：
- 如果一味的遵循业界的界面标准，则会丧失自己的个性.在框架符合以上规范的情况下，设计具有自己独特风格的界面尤为重要。尤其在商业软件流通中有着很好的迁移默化的广告效用。
- 测试细则：
- 1):安装界面上应有单位介绍或产品介绍，并有自己的图标。
- 2):主界面，最好是大多数界面上要有公司图标。

# §5　界面测试考虑

- 3)：　登录界面上要有本产品的标志，同时包含公司图标。
- 4)：　帮助菜单的“关于”中应有版权和产品信息。
- 5)：　公司的系列产品要保持一直的界面风格，如背景色、字体、菜单排列方式、图标、安装过程、按钮用语等应该大体一致。

# §5 界面测试考虑

- 8：快捷方式的组合
- 在菜单及按钮中使用快捷键可以让喜欢使用键盘的用户操作得更快一些在西文Windows及其应用软件中快捷键的使用大多是一致的。
- 菜单中：
- 1)：面向事务的组合有：Ctrl-D 删除 ； Ctrl-F 寻找 ； Ctrl –H替换； Ctrl-I 插入 ； Ctrl-N 新记录 ； Ctrl-S 保存 Ctrl-O 打开。
- 2)：列表： Ctrl-R ， Ctrl-G定位； Ctrl-Tab下一分页窗口或反序浏览同一页面控件； 。

# §5 界面测试考虑

- 3)：**编辑**：Ctrl-A 全选； Ctrl-C 拷贝；Ctrl-V 粘贴； Ctrl-X 剪切； Ctrl-Z撤消操作； Ctrl-Y恢复操作。
- 4)**文件操作**:Ctrl-P 打印； Ctrl-W 关闭。
- 5)：**系统菜单**：Alt-A文件； Alt-E编辑；Alt-T工具； Alt—W窗口； Alt—H帮助。

# §5 界面测试考虑

- 6):MS Windows保留键:Ctrl-Esc 任务列表 ；Ctrl-F4 关闭窗口； Alt-F4 结束应用；Alt-Tab 下一应用 ；Enter 缺省按钮/确认操作 ；Esc 取消按钮/取消操作；Shift-F1 上下文相关帮助。
- 按钮中：
- 可以根据系统需要而调节，以下只是常用的组合。
- Alt-Y确定(是)；Alt-C取消；Alt-N 否；Alt-D删除； Alt-Q退出； Alt-A添加； Alt-E编辑；Alt-B浏览；Alt-R读；Alt-W写。

# §5 界面测试考虑

- 9:安全性考虑
- 安全性细则:
- 1)：最重要的是排除可能会使应用非正常中止的错误。
- 2)：应当注意尽可能避免用户无意录入无效的数据。
- 3)：采用相关控件限制用户输入值的种类。
- 4)：当用户作出选择的可能性只有两个时,可以采用单选框。
- 5)：当选择的可能再多一些时,可以采用复选框,每一种选择都是有效的,用户不可能输入任何一种无效的选择。
- 6)：当选项特别多时，可以采用列表框，下拉式列表框。

# §5 界面测试考虑

- 7)：在一个应用系统中,开发者应当避免用户作出未经授权或没有意义的操作。
- 8)：对可能引起致命错误或系统出错的输入字符或动作要加限制或屏蔽。
- 9):对可能发生严重后果的操作要有补救措施。通过补救措施用户可以回到原来的正确状态。
- 10):对一些特殊符号的输入、与系统使用的符号相冲突的字符等进行判断并阻止用户输入该字符。
- 11)：对错误操作最好支持可逆性处理，如取消系列操作。
- 12):在输入有效性字符之前应该阻止用户进行只有输入之后才可进行的操作。

# §5 界面测试考虑

- 13):对可能造成等待时间较长的操作应该提供取消功能。
- 14)：特殊字符常有；;'"><, ` ‘ : “ [ ” { 、 \| } ] +=)-(_*&&^%$#@! ,.。?/还有空格。
- 15)：与系统采用的保留字符冲突的要加以限制。
- 16)：在读入用户所输入的信息时，根据需要选择是否去掉前后空格。
- 17)：有些读入数据库的字段不支持中间有空格，但用户切实需要输入中间空格，这时要在程序中加以处理。

# §5 界面测试考虑

- 10:多窗口的应用与系统资源：
- 设计良好的软件不仅要有完备的功能，而且要尽可能的占用最底限度的资源。
- 1)：在多窗口系统中，有些界面要求必须保持在最顶层，避免用户在打开多个窗口时，不停的切换甚至最小化其他窗口来显示该窗口。
- 2)：在主界面载入完毕后自动卸出内存，让出所占用的WINDOWS系统资源。
- 3)：关闭所有窗体，系统退出后要释放所占的所有系统资源 ，除非是需要后台运行的系统。
- 4)：尽量防止对系统的独占使用。

# §5 界面测试考虑

- 5)：窗口能否基于相关的输入或菜单命令适当地打开？
- 6)：窗口能否改变大小、移动和滚动？
- 7)：窗口中的数据内容能否使用鼠标、功能键、方向箭头和键盘访问？
- 8)：当被覆盖并重调用后，窗口能否正确地再生？
- 9)：需要时能否使用所有窗口相关的功能？
- 10)：所有窗口相关的功能是可操作的吗？
- 11)：是否有相关的下拉式菜单、工具条、滚动条、对话框、按钮、图标和其他控制可为窗口可用，并适当地显示？

# §5 界面测试考虑

- 12)：显示多个窗口时，窗口的名称是否被适当地表示？
- 13)：活动窗口是否被适当地加亮？
- 14)：如果使用多任务，是否所有的窗口被实时更新？
- 15)：多次或不正确按鼠标是否会导致无法预料的副作用？
- 16)：窗口的声音和颜色提示和窗口的操作顺序是否符合需求？
- 17)：窗口是否正确地关闭？

# §6 面向对象的软件测试概述

- 一度实践证明行之有效的软件测试对面向对象技术开发的软件多少显得有些力不从心。尤其是面向对象技术所独有的多态，继承，封装等新特点，产生了传统语言设计所不存在的错误可能性，或者使得传统软件测试中的重点不再显得突出，或者使原来测试经验认为和实践证明的次要方面成为了主要问题。

# §6 面向对象的软件测试概述

- 面向对象程序的结构不再是传统的功能模块结构，作为一个整体，原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已成为不可能。而且，面向对象软件抛弃了传统的开发模式，对每个开发阶段都有不同以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此，传统的测试模型对面向对象软件已经不再适用。针对面向对象软件的开发特点，应该有一种新的测试模型。

# §6 面向对象的软件测试概述

- 一、面向对象测试模型(Object-OrientTestModel)
- 面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析（OOA），面向对象设计（OOD），和面向对象编程（OOP）三个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。

# §6 面向对象的软件测试概述

- 1、对认定的对象的测试
- 2、对认定的结构的测试
- （1）对认定的分类结构的测试
- （2）对认定的组装结构的测试
- 3、对认定的主题的测试
- 4、对定义的属性和实例关联的测试
- 5、对定义的服务和消息关联的测试

# §6 面向对象的软件测试概述

- 二、面向对象设计的测试(OODTest)
- 通常的结构化的设计方法，用的"是面向作业的设计方法，它把系统分解以后，提出一组作业，这些作业是以过程实现系统的基础构造，把问题域的分析转化为求解域的设计，分析的结果是设计阶段的输入"

# §6 面向对象的软件测试概述

- 而面向对象设计（OOD）采用“造型的观点”，以OOA为基础归纳出类，并建立类结构或进一步构造成类库，实现分析结果对问题空间的抽象。OOD归纳的类，可以是对象简单的延续，可以是不同对象的相同或相似的服务。
- OOD不是在OOA上的另一思维方式的大动干戈，而是OOA的进一步细化和更高层的抽象。所以，OOD与OOA的界限通常是难以严格区分的。OOD确定类和类结构不仅是满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，能方便实现功能的重用和扩增，以不断适应用户的要求。

# §6 面向对象的软件测试概述

- 对OOD的测试，建议针对功能的实现和重用以及对OOA结果的拓展，从如下三方面考虑：

☆对认定的类的测试
☆对构造的类层次结构的测试
☆对类库的支持的测试

# §6 面向对象的软件测试概述

- 三、面向对象编程的测试(OOPTest)
- 典型的面向对象程序具有继承、封装和多态的新特性，这使得传统的测试策略必须有所改变。

# §6 面向对象的软件测试概述

- 面向对象程序是把功能的实现分布在类中。能正确实现功能的类，通过消息传递来协同实现设计要求的功能。正是这种面向对象程序风格，将出现的错误能精确的确定在某一具体的类。因此，在面向对象编程（OOP）阶段，忽略类功能实现的细则，将测试的目光集中在类功能的实现和相应的面向对象程序风格，主要体现为以下两个方面（假设编程使用C++语言）。

- ☆数据成员是否满足数据封装的要求
  ☆类是否实现了要求的功能

# §6 面向对象的软件测试概述

- 五、面向对象的单元测试(OOUnitTest)
- 传统的单元测试是针对程序的函数、过程或完成某一定功能的程序块。沿用单元测试的概念，实际测试类成员函数。一些传统的测试方法在面向对象的单元测试中都可以使用。如等价类划分法，因果图法，边值分析法，逻辑覆盖法，路径分析法，程序插装法等等，方法的具体实现参见。

# §6 面向对象的软件测试概述

- 在设计测试用例选择输入数据时，可以基于以下两个假设：
- 1. 如果函数（程序）对某一类输入中的一个数据正确执行，对同类中的其他输入也能正确执行。该假设的思想可参见[6]中介绍的等价类划分。
- 2. 如果函数（程序）对某一复杂度的输入正确执行，对更高复杂度的输入也能正确执行。例如需要选择字符串作为输入时，基于本假设，就无须计较于字符串的长度。除非字符串的长度是要求固定的，如IP地址字符串。

# §6 面向对象的软件测试概述

- 五、面向对象的集成测试(OOIntegrateTest)
- 传统的集成测试，是由底向上通过集成完成的功能模块进行测试，一般可以在部分程序编译完成的情况下进行。而对于面向对象程序，相互调用的功能是散布在程序的不同类中，类通过消息相互作用申请和提供服务。类的行为与它的状态密切相关，状态不仅仅是体现在类数据成员的值，也许还包括其他类中的状态信息。由此可见，类相互依赖极其紧密，根本无法在编译不完全的程序上对类进行测试。所以，面向对象的集成测试通常需要在整个程序编译完成后进行。此外，面向对象程序具有动态特性，程序的控制流往往无法确定，因此也只能对整个编译后的程序做基于黑盒子的集成测试。

# §6 面向对象的软件测试概述

- 面向对象的集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。基于单元测试对成员函数行为正确性的保证，集成测试只关注于系统的结构和内部的相互作用。面向对象的集成测试可以分成两步进行：先进行静态测试，再进行动态测试。

# §6 面向对象的软件测试概述

- 静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求。现在流行的一些测试软件都能提供一种称为"可逆性工程"的功能，即通过原程序得到类关系图和函数功能调用关系图，例如InternationalSoftwareAutomation公司的Panorama-2forWindows95、Rational公司的RoseC++Analyzer等，将"可逆性工程"得到的结果与OOD的结果相比较，检测程序结构和实现上是否有缺陷。换句话说，通过这种方法检测OOP是否达到了设计要求。

# §6 面向对象的软件测试概述

- 动态测试设计测试用例时，通常需要上述的功能调用结构图、类关系图或者实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，减少测试工作量，使得进行的测试能够达到一定覆盖标准。测试所要达到的覆盖标准可以是：达到类所有的服务要求或服务提供的一定覆盖率；依据类间传递的消息，达到对所有执行线程的一定覆盖率；达到类的所有状态的一定覆盖率等。同时也可以考虑使用现有的一些测试工具来得到程序代码执行的覆盖率。

# §6 面向对象的软件测试概述

- 具体设计测试用例，可参考下列步骤：

1. 先选定检测的类，参考OOD分析结果，仔细出类的状态和相应的行为，类或成员函数间传递的消息，输入或输出的界定等。

2. 确定覆盖标准。

3. 利用结构关系图确定待测类的所有关联。

4. 根据程序中类的对象构造测试用例，确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

# §6 面向对象的软件测试概述

- 六、面向对象的系统测试(OOSystemTest)
- 系统测试时，应该参考OOA分析的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全“再现”问题空间。系统测试不仅是检测软件的整体行为表现，从另一个侧面看，也是对软件开发设计的再确认。
- 系统测试应该尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对临时没有的系统设备部件，也应有相应的模拟手段。