

程序设计基础

教学团队：徐明星，兴军亮，任炬

renju@tsinghua.edu.cn

2024秋，每周一第2节，三教2301

【调试技巧】有错误的示例代码（任务2.3）

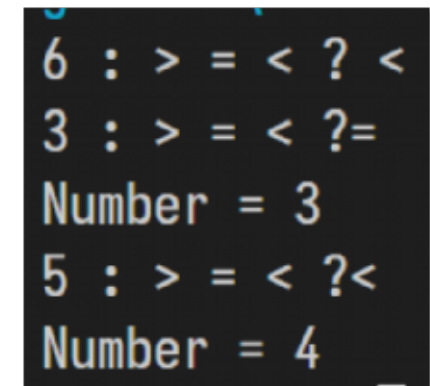
```
#include <iostream>
using namespace std;

int main() {
    char ans;
    cout << "6 : > = < ? " ;
    cin >> ans;
    if (ans == '=')
        cout << "Number = 6" << endl;
    if (ans == '<') // 1 2 3 4 5
    {
        cout << "3 : > = < ?";
        cin >> ans;
        if (ans == '=') cout << "Number = 3" <<
endl;
        if (ans == '<') { // 1, 2
            cout << "2 : > = < ? ";
            cin >> ans;
            if (ans == '=')
                cout << "Number = 2" << endl;
            else
                cout << "Number = 1" << endl;
        }
        else { // 4, 5
            cout << "5 : > = < ?";
            cin >> ans;
            if (ans == '=')
                cout << "Number = 5" << endl;
            else
                cout << "Number = 4" << endl;
        }
    }
}
```

```
else { // 7 8 9 10
    cout << "9 : > = < ?";
    cin >> ans;
    if (ans == '=')
        cout << "Number = 9" << endl;
    if (ans == '<') // 7, 8
    {
        cout << "8 : > = < ?";
        cin >> ans;
        if (ans == '=')
            cout << "Number = 8" << endl;
        else
            cout << "Number = 7" << endl;
    }
    else // 10
        cout << "Number = 10" << endl;
}

return 0;
} // L02-07.cpp
```

当设想小物品价格为3时
程序运行不正常/错误



```
6 : > = < ? <
3 : > = < ?=
Number = 3
5 : > = < ?<
Number = 4
```

程序的运行截图

当有确切测试数据让程序出错时 ……

```
10  if (ans == '<')  /// 1 2 3 4 5
11  {
12      cout << "3 : > = < ?";
13      cin >> ans;
14  if (ans == '=') cout << "Number = 3" << endl;
15  if (ans == '<') { // 1, 2
16      cout << " 2 : > = < ? ";
17      cin >> ans;
18      if (ans == '=')
19          cout << "Number = 2" << endl;
20      else
21          cout << "Number = 1" << endl;
22  }
23  else { // 4, 5
24      cout << "5 : > = < ?";
25      cin >> ans;
26      if (ans == '=')
27          cout << "Number = 5" << endl;
28      else
29          cout << "Number = 4" << endl;
30  }
31  }
```

用测试数据3对程序代码
进行“静态调试”——
心算（基于推理进行调试
）

发现程序在输出答案后继
续转至第24执行！

显然，这是不对的！

应该转到第31行，结束第
10行开始的if语句！

当有确切测试数据让程序出错时 ……

```
10  ✓   if (ans == '<')   /// 1 2 3 4 5
11      {
12          cout << "3 : > = < ?";
13          cin >> ans;
14      if (ans == '=') cout << "Number = 3" << endl;
15  ✓   else if (ans == '<') { // 1, 2
16          cout << " 2 : > = < ? ";
17          cin >> ans;
18          if (ans == '=')
19              cout << "Number = 2" << endl;
20          else
21              cout << "Number = 1" << endl;
22      }
23  ✓   else { // 4, 5
24      cout << "5 : > = < ?";
25          cin >> ans;
26          if (ans == '=')
27              cout << "Number = 5" << endl;
28          else
29              cout << "Number = 4" << endl;
30      }
31  }
```

在逻辑上，第15行应该是“不等于3”的情形入口，所以，应该将14-15行代码改为if-else语句来处理“等于/不等于3”

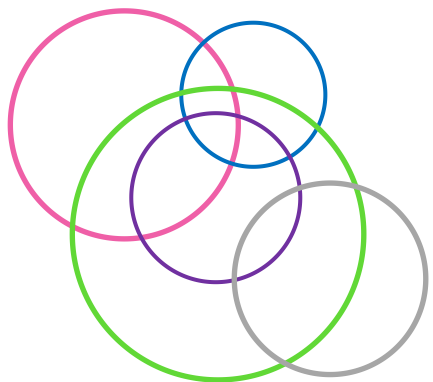
```
6 : > = < ? <
3 : > = < ? =
Number = 3
```

修改后的运行截图

第3讲 逻辑思维与计算机解题

计算机强大的逻辑分析功能是由人通过程序赋给它的。

逻辑问题必须转换成计算机能够理解的
数学表达式和相应的程序指令。



【任务3.1】谁做的好事？

清华附中有四位同学A、B、C、D，其中一位做了好事，不留名，表扬信来了之后，校长问这四位是谁做的好事。

- A说：不是我。
- B说：是C。
- C说：是D。
- D说：他胡说。

已知：三个人说的是真话，一个人说的是假话。现在请你根据这些信息，编写程序找出做了好事的人。

分析事件真相的可能性

根据任务3.1的题意，A、B、C、D四个人中只有一位是做好事者。若是做好事，则记为T（True），若不是好事的，则记为F（False），则存在如下 4 种可能性（情况）：

可能性	A	B	C	D
第1种	T	F	F	F
第2种	F	T	F	F
第3种	F	F	T	F
第4种	F	F	F	T

T: True; F: False

关键点1：如何实现推理？

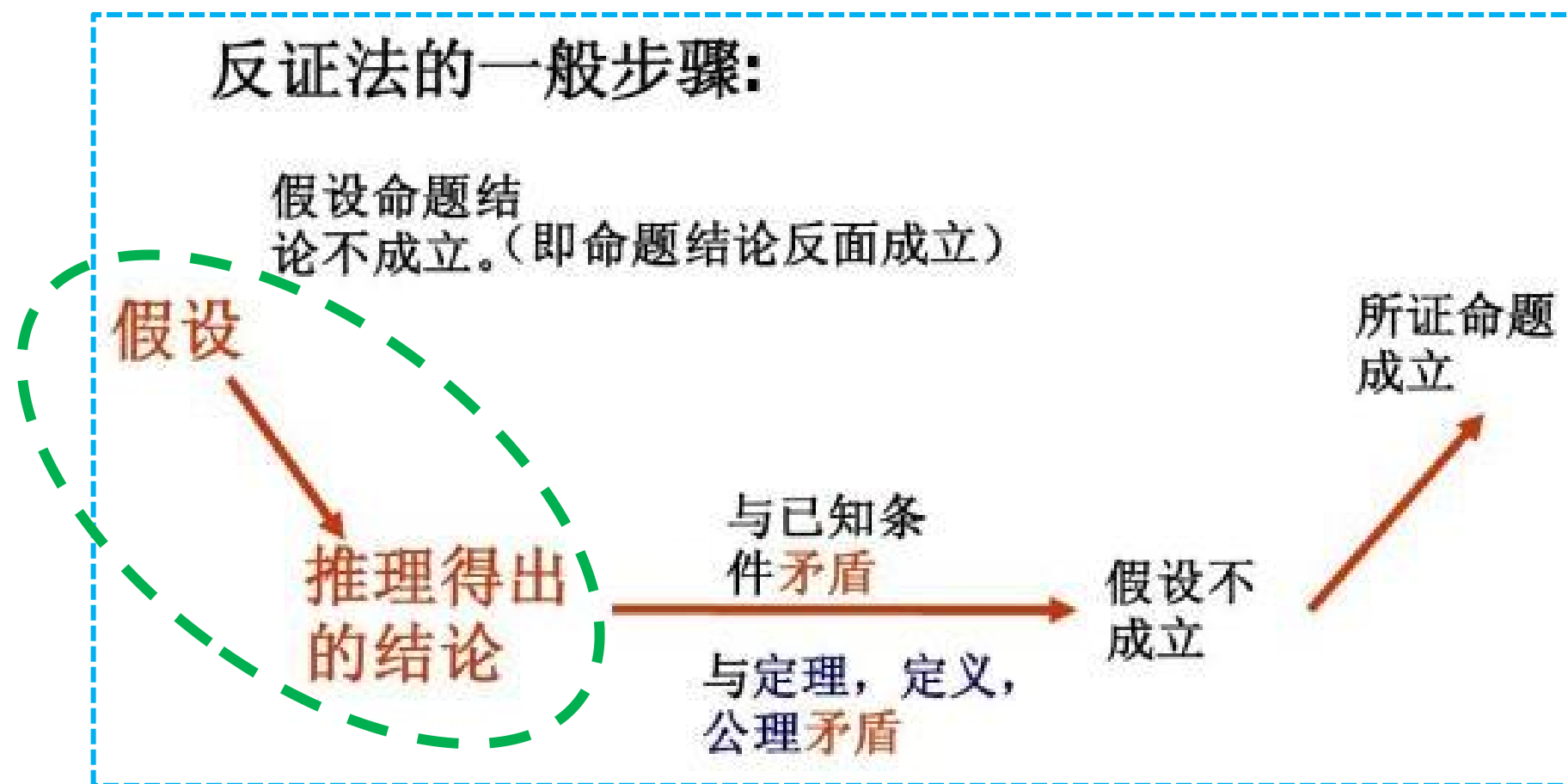
用枚举法验证各可能性

按照题目包含的四种可能性，**逐一**对每一种可能性进行**检验**，即：依据该可能性，对四个人的话进行检验，看看有几句是真话。如果不满足三句为真的条件，即不符合题目要求，就抛弃这种可能性，换下一个可能性再试。

我们称上述解决问题的思路为“枚举法”。

用枚举法验证各可能性

按照四种可能性，逐一对每种可能性去测试四个人的话，



“按照可能性进行测试”的过程，与大家熟悉的数学上的“反证法”很相似，都是“先做出假设，然后基于假设进行推理”。

用枚举法验证各可能性

按照四种可能性，逐一对每种可能性去测试四个人的话，

例1. 已知：一个整数的平方能被2整除，
求证：这个数是偶数。

证明：设整数 a 的平方能被2整除。

假设 a 不是偶数，

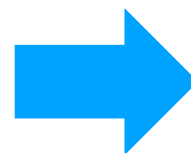
则 a 是奇数，不妨设 $a=2m+1$ (m 是整数)

$$\therefore a^2 = (2m+1)^2 = 4m^2 + 4m + 1 = 4m(m+1) + 1$$

$\therefore a^2$ 是奇数，与已知矛盾。

\therefore 假设不成立，所以 a 是偶数。

类似的



假定“做好事者是A”



在计算机（程序）中如何实现上述“假定”呢？

关键点2：如何表示语义？

要将自然语言表达的语义（即四个人说的四句话的意思），用计算机可以处理（运算）的式子来表示，只有这样，才能通过程序语言编程来指挥计算机进行处理。

如何表达“假定做好事者是A”？

假定“做好事者是
A”



假定

“做好事者是
A”

根据前面的分析，“假定 XXX”是对某个可能性（XXX）的临时性承认或接受，是为了开展后续推理、论证的一种条件设定（设置）动作，也就是：有某个东西（条件）被临时性的设置为某个特定的内容了。

根据前一讲学习的知识，上述对“某个东西的内容设定”，与编程语言中的“变量赋值”是对应的，或者说，可以用变量来表示那个条件，用值来表示其内容，用赋值来表示对条件内容的设定。

如何表达“假定做好事者是A”？

“做好事者是
A”

上面的这种“假定”，表达的语义是“X是Y”。X对应做好事者，Y对应A。

根据前面的分析，需要用一個变量来表示“好事者”（即X），把值（即A）赋给变量，在编程语言中，这里的A可以用字符类型的数据来表示，即 'A'。

由于变量要存储的内容是字符类型的数据，所以我们定义一个字符类型的变量来表示“做好事者”，如：

```
char thisman;
```

则，在推理过程中临时性的假定“做好事者是A” 对应一条赋值语句：

```
thisman = 'A';
```

对假设进行验证

例1. 已知：一个整数的平方能被2整除，
求证：这个数是偶数。

证明： 设整数 a 的平方能被2整除。

假设 a 不是偶数，

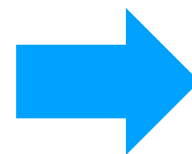
则 a 是奇数，不妨设 $a=2m+1$ (m 是整数)

$$\therefore a^2 = (2m+1)^2 = 4m^2 + 4m + 1 = 4m(m+1) + 1$$

$\therefore a^2$ 是奇数，与已知矛盾。

\therefore 假设不成立，所以 a 是偶数。

类似的



根据某个假设，对ABCD四个人的话进行计算（推理），来验证假设是不是成立。

判断性的话（命题）如何表示？

陈述命题的句式 —— “X是Y”，表达的是一种判断，对是否具有相等关系的判断。命题的“值”只有“真”与“假”两种。在用编程语言来表示命题“X是Y”时，要使用关系运算。

说话人	说的话	写成关系表达式
A	“[这个人]不是我”	<code>thisman != 'A'</code>
B	“[这个人]是C”	<code>thisman == 'C'</code>
C	“[这个人]是D”	<code>thisman == 'D'</code>
D	“他(C)胡说” “C胡说”的意思，是说C说的不对，意思是C所说的 <code>thisman = 'D'</code> 是错的	<code>thisman != 'D'</code>

，即：D的观点是 `thisman != 'D'`

逐一测试4种可能性：代入四句话中

(1) 假定 thisman 是 'A'，代入四句话中

STEP 1. 做出假设，即 thisman = 'A'

说话人	说的话
A	thisman != 'A';
B	thisman == 'C';
C	thisman == 'D';
D	thisman != 'D';

基于假设 thisman = 'A'

STEP 2. 进行推理



关系表达式	
'A' != 'A'	false
'A' == 'C'	false
'A' == 'D'	false
'A' != 'D'	true



只有一句是真话，不满足题目要求，所以，假设不成立，做好事的不是A。

逐一测试4种可能性：代入四句话中

(2) 假定 thisman 是 'B'，代入四句话中

说话人	说的话	关系表达式	真话吗
A	thisman != 'A' ;	'B' != 'A'	1
B	thisman == 'C' ;	'B' == 'C'	0
C	thisman == 'D' ;	'B' == 'D'	0
D	thisman != 'D' ;	'B' != 'D'	1

逐一测试4种可能性：代入四句话中

(3) 假定 thisman 是 'C'，代入四句话中

说话人	说的话	关系表达式	真话吗
A	thisman != 'A' ;	'C' != 'A'	1
B	thisman == 'C' ;	'C' == 'C'	1
C	thisman == 'D' ;	'C' == 'D'	0
D	thisman != 'D' ;	'C' != 'D'	1

逐一测试4种可能性：代入四句话中

(4) 假定 thisman 是 'D'，代入四句话中

说话人	说的话	关系表达式	真话吗
A	thisman != 'A' ;	'D' != 'A'	1
B	thisman == 'C' ;	'D' == 'C'	0
C	thisman == 'D' ;	'D' == 'D'	1
D	thisman != 'D' ;	'D' != 'D'	0

如何对假设进行检验呢？

通过给变量赋值，做出假设，例如： `thisman = 'A'`

例1. 已知：一个整数的平方能被2整除，
求证：这个数是偶数。

证明： 设整数a的平方能被2整除。

假设a不是偶数，

则a是奇数，不妨设 $a=2m+1$ (m是整数)

$$\therefore a^2 = (2m+1)^2 = 4m^2 + 4m + 1 = 4m(m+1) + 1$$

$\therefore a^2$ 是奇数，与已知矛盾。

\therefore 假设不成立，所以a是偶数。



关系表达式	真话吗
'A' != 'A'	0
'A' == 'C'	0
'A' == 'D'	0
'A' != 'D'	1

从左表计算可得真话数量是1，与事实（3名学生说真话）不符。

所以，假设（“做好事者是A”）不成立。

如何数真话的数量（方法1）

变量定义

```
char thisman;  
int sum;
```

设定初值

```
thisman = 'A'; // 假设做好事者为A  
sum = 0;      // 计数器初始值为0
```

统计
真话数量

```
if (thisman != 'A') sum++; // 若A说的话为真，则sum加1  
if (thisman == 'C') sum++; // 若B说的话为真，则sum加1  
if (thisman == 'D') sum++; // 若C说的话为真，则sum加1  
if (thisman != 'D') sum++; // 若D说的话为真，则sum加1
```

判断假设
是否成立

```
if (sum == 3)  
    cout << "this man is A\n"; // 若真话数量为3，则做好事者为A  
else  
    cout << "this man is NOT A\n";
```

如何数真话的数量（方法2）

变量定义

```
char thisman;  
int sum;
```

设定初值

```
thisman = 'A'; // 假设做好事者为A
```

统计
真话数量

```
sum = (thisman != 'A') + (thisman == 'C') +  
      (thisman == 'D') + (thisman != 'D');
```

判断假设
是否成立

```
if (sum == 3)  
    cout << "this man is A\n"; // 若真话数量为3，则做好事者为A  
else  
    cout << "this man is NOT A\n";
```

如何枚举4种可能性呢？

```
thisman = 'A'; // 假设做好事者为A
```

```
sum =
```

```
thisman = 'B'; // 假设做好事者为B
```

```
sum =
```

```
thisman = 'C'; // 假设做好事者为C
```

```
thisman = 'D'; // 假设做好事者为D
```

```
if (su
```

```
co
```

```
else
```

```
co
```

```
if (su
```

```
co
```

```
else
```

```
co
```

```
if (su
```

```
co
```

```
else
```

```
co
```

```
sum = (thisman != 'A') + (thisman == 'C') +  
      (thisman == 'D') + (thisman != 'D');
```

```
if (sum == 3)
```

```
    cout << "this man is D\n";
```

```
else
```

```
    cout << "this man is NOT D\n";
```

基于“C-V大法”对4种可能性进行枚举

```
#include <iostream>
using namespace std ;
int main()
{
```

```
    char thisman = 'A' - 1;
    int sum;
```

这种写法有什么特别的考虑吗？
(或：有什么好处呢？)

```
    /// A
```

```
    thisman++;
    sum = ( thisman != 'A' ) + ( thisman == 'C' )
        + ( thisman == 'D' ) + ( thisman != 'D' );
```

```
    if (sum == 3)
    {    // 如果3句话为真，则输出当前可能性所假定的人为做好事者
        cout << "做好事者为" << thisman << endl;
        return 0;
    }
```

```
    /// B
```

```
    thisman++;
    sum = ( thisman != 'A' ) + ( thisman == 'C' )
        + ( thisman == 'D' ) + ( thisman != 'D' );
```

```
    if (sum == 3)
    {    // 如果3句话为真，则输出当前可能性所假定的人为做好事者
        cout << "做好事者为" << thisman << endl;
        return 0;
    }
```

改进版本

/// C

```

thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{
    // 如果3句话为真，则输出当前可能性所假定的人为做好事者
    cout << "做好事者为" << thisman << endl;
    return 0;
}

```

/// D

```

thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{
    // 如果3句话为真，则输出当前可能性所假定的人为做好事者
    cout << "做好事者为" << thisman << endl;
    return 0;
}

```

```

    cout << "找不出做好事者" << endl; // 输出无解信息
    return 0 ;
} // L03-01.cpp

```

应尽量避免程序中出现重复代码

```
/// A
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{
    // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
    cout << "做好事者为" << thisman << endl;
    return 0;
}
```

```
/// B
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{
    // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
    cout << "做好事者为" << thisman << endl;
    return 0;
}
```

```
/// C
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{
    // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
    cout << "做好事者为" << thisman << endl;
    return 0;
}
```

```
/// D
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{
    // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
    cout << "做好事者为" << thisman << endl;
    return 0;
}
```

同一代码段重
复出现了4次!

出现“重复代码”时的风险和隐患

- 1、字面上的重复，并**不能精确表达**“实质性重复”的含义
- 2、如果使用的算法确实是一样的，那么，当情况发生变化而要修改时，就需要在多个地方进行修改，而这些地方有可能并不集中，这样就**容易遗漏**
- 3、代码长了，增加阅读负担，对代码逻辑的**理解更困难**
- 4、维护代码的人，在进行优化时仍需仔细对比，并不会因为看起来似乎一样，就会认为是一样，这就导致**维护工作量**也加大了
- 5、如果不小心修改了某一处重复的代码，但没有发觉，以为还是跟以前一样是重复的，那就会导致BUG，产生了**调试工作量**

需要用专门的语句表达重复



```
/// A
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```

```
/// B
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

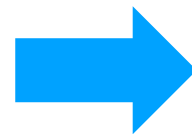
if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```

```
/// C
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```

```
/// D
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```



请重复执行以下操作4次

{

```
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```

}

使用“循环语句”表达重复

```
/// A
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```

```
/// B
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

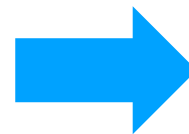
if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```

```
/// C
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```

```
/// D
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
  cout << "做好事者为" << thisman << endl;
  return 0;
}
```



```
for (int i=0; i<4; i++)
```

```
{
```

```
    thisman++;
    sum = ( thisman != 'A' ) + ( thisman == 'C' )
          + ( thisman == 'D' ) + ( thisman != 'D' );

    if (sum == 3)
    { // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
      cout << "做好事者为" << thisman << endl;
      return 0;
    }
}
```

```
}
```

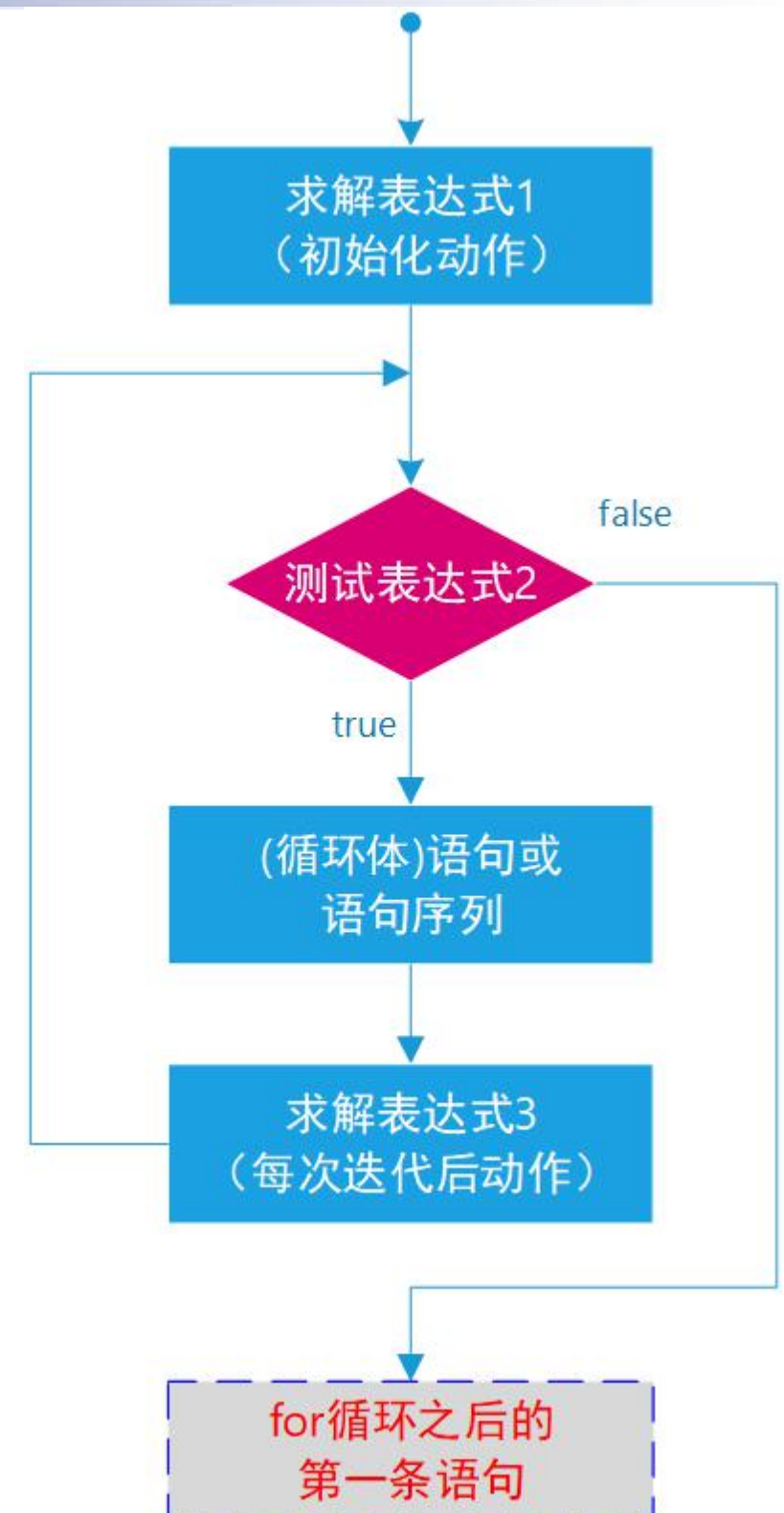

for 循环语句

for是计数型循环的标识符，语句格式为：

```
for ( 表达式1; 表达式2; 表达式3 )  
{  
    循环体（语句组）；  
}
```

圆括号括起的是三个表达式。其下的大括号括起的部分是循环体。

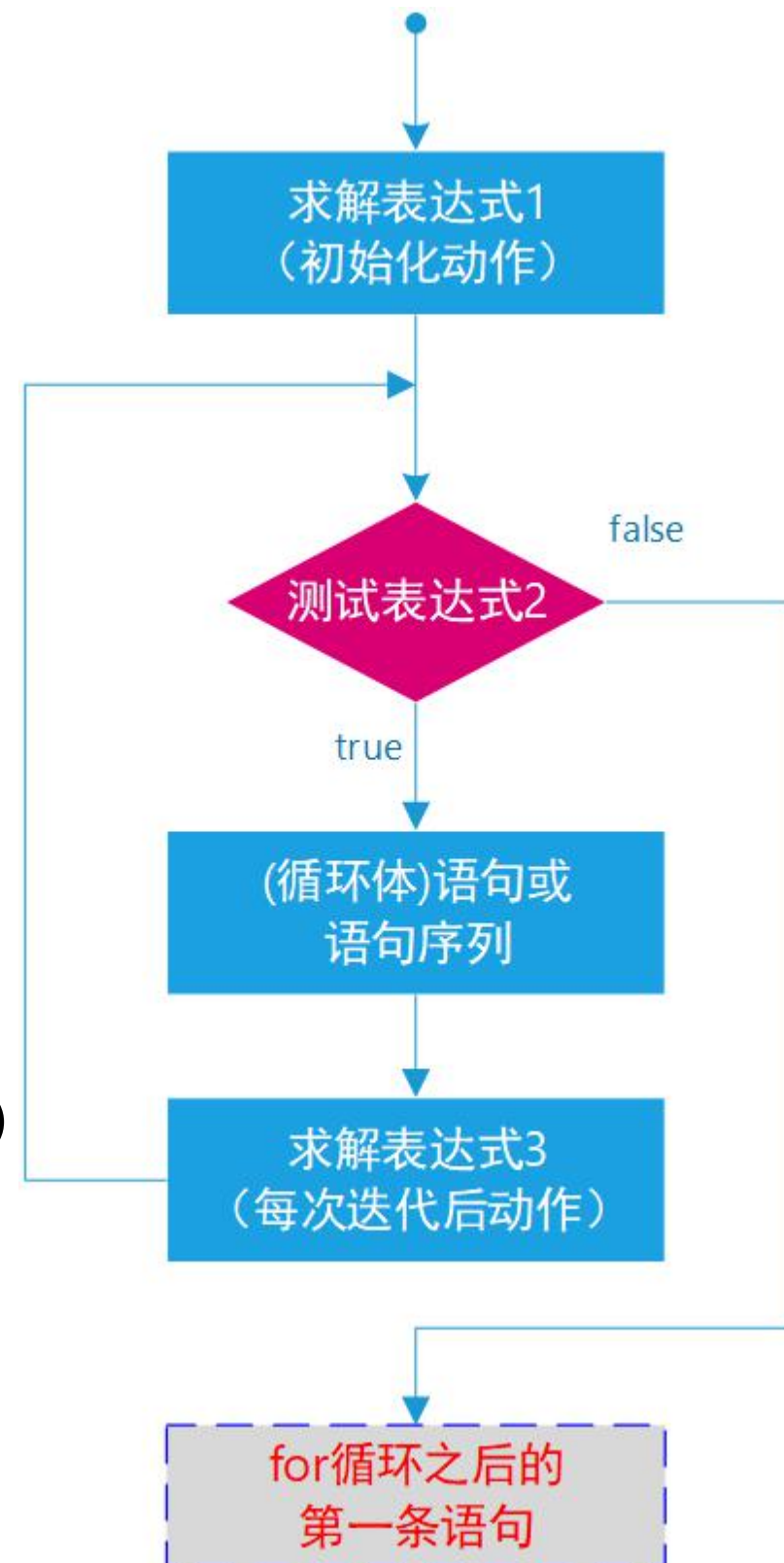
右图描述了 for 循环的执行流程图。



如何理解循环语句的运行？

让自己像计算机一样“思考”（执行）

- (1) 执行表达式1，设置循环变量的初值
- (2) 测试表达式2，测试是否未到循环变量的终值
 - (2.1) 如表达式2的值为真，则转到【步骤(3)】
 - (2.2) 如表达式2的值为假，则转到【步骤(5)】
- (3) 执行循环体语句组
- (4) 执行表达式3，改变循环变量的值，转到【步骤(2)】
- (5) 执行for语句的下一条语句



【小练习】for 循环语句

练习1、输出ASCII表中从'!'到'~'的所有字符

```
#include <iostream>
using namespace std;

int main()
{
    for (char c='!'; c<='~'; c++)
        cout << c << ' ';
    cout << endl;

    return 0;
} // E03-03.cpp
```

字符类型的变量（如左边源程序中的变量c）在进行关系运算、加减运算时，其执行过程是：使用该字符的ASCII编码值（整数）进行运算。

因此，左边源程序中的 `c<='~'`，是将字符c的ASCII值与'~'的ASCII值进行比较；`c++`；是将字符c的ASCII值加1对应的字符保存到变量c中。

【小练习】for 循环语句

练习2、计算并输出 $1+2+\dots+100$ 的和

```
#include <iostream>
using namespace std;

int main()
{
    int sum = 0;
    for (int n=1; n<=100; n++)
        sum += n;
    cout << "1+2+...+100 = " << sum << endl;
    return 0;
} // E03-01.cpp
```

【小练习】for 循环语句

练习3、计算并输出 $1*2*...*10$ 的积

```
#include <iostream>
using namespace std;

int main()
{
    int mul = 1;
    for (int n=1; n<=10; n++) // or n=2 .. 10
        mul *= n;
    cout << "1*2*...*10 = " << mul << endl;
    return 0;
} // E03-02.cpp
```

作业“回头再看”——奇数且3倍数

V1

```
int main()
{
    int sum = 0, cnt = 0;
    int tmp;

    for (int i=0; i<6; i++)
    {
        cin >> tmp;
        if (tmp % 2 == 1)
        {
            if (tmp % 3 == 0)
            {
                sum += tmp;
                cnt ++;
            }
        }
    }

    if (cnt == 0)
        cout << "0.0000" << endl;
    else
        cout << fixed << setprecision(4) << 1.0 * sum / cnt << endl;

    return 0;
} // 0302-02B.cpp
```

“一边接受用户输入，一边完成检测判断，一边完成累加计数”

作业“回头再看”——奇数且3倍数

V2

```
int main()
{
    double sum = 0;
    int cnt = 0, tmp;

    for (int i=0; i<6; i++)
    {
        cin >> tmp;
        if (tmp % 2 == 1)
        {
            if (tmp % 3 == 0)
            {
                sum += tmp;
                cnt ++;
            }
        }
    }

    if (sum > 0.0)
        sum /= cnt;
    cout << fixed << setprecision(4) << sum << endl;

    return 0;
} // 0J02-02C.cpp
```

【任务3.1】方法1：循环仅用来控制重复的次数

```
/// for ( 表达式1; 表达式2; 表达式3 )  
for (int i=0; i<4; i++) /// 重复4次循环体中的语句!  
{
```

```
    thisman++;  
    sum = ( thisman != 'A' ) + ( thisman == 'C' )  
          + ( thisman == 'D' ) + ( thisman != 'D' );  
  
    if (sum == 3)  
    { // 如果3句话为真, 则输出当前可能性所假定的人为做好事者  
        cout << "做好事者为" << thisman << endl;  
        return 0;  
    }
```

```
}
```

完整的算法实现

```
#include <iostream>
using namespace std ;
int main()
{
    char thisman = 'A' - 1;
    int sum;
```

```
    for (int i=0; i<4; i++)
    {
```

```
        thisman++;
        sum = ( thisman != 'A' ) + ( thisman == 'C' )
              + ( thisman == 'D' ) + ( thisman != 'D' );
```

```
        if (sum == 3)
        { // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
            cout << "做好事者为" << thisman << endl;
            return 0;
        }
    }
```

```
    cout << "找不出做好事者" << endl; // 输出无解信息
    return 1; // 返回1表示题目无解
```

```
} // L03-02.cpp
```

```
thisman++;
sum = ( thisman != 'A' ) + ( thisman == 'C' )
      + ( thisman == 'D' ) + ( thisman != 'D' );

if (sum == 3)
{ // 如果3句话为真, 则输出当前可能性所假定的人为做好事者
    cout << "做好事者为" << thisman << endl;
    return 0;
}
```

【任务3.1】方法2：循环体用到了“循环变量”

```
for (int k=0; k<4; k++)           // 变量k称为循环变量
{ // 循环体开始
    thisman = 'A' + k;             // 产生被试者，依次为'A','B','C','D'
                                    // 赋值给thisman

    sum = ( thisman != 'A' )       // 测试A的话是否为真
          + ( thisman == 'C' )     // 测试B的话是否为真
          + ( thisman == 'D' )     // 测试C的话是否为真
          + ( thisman != 'D' );    // 测试D的话是否为真

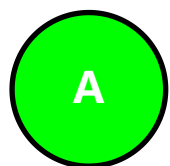
    // TO DO ...

} // 循环体结束
```

“循环变量”是循环体（过程）的“变化因素”，是从循环体中归纳提炼出来的可变部分

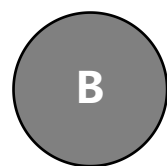
前一页源程序片段中的循环语句，是否可以改写为如下形式？

```
for (char thisman='A'; thisman<='D'; thisman++)  
{  
    sum = ( thisman != 'A')  
        + ( thisman == 'C')  
        + ( thisman == 'D')  
        + ( thisman != 'D');  
  
    // TO DO ...  
}
```



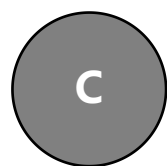
A

可以



B

不可以



C

不知道

提交

算法优化 —— 如何知道问题无解？

用标志变量记录是否有解

```
#include <iostream>
using namespace std ;
int main()
{
    int g = 0 ;    // 初始化为0，表示“无解”
    for (int k=0; k<4; k++) // k是循环控制变量
    { // for 循环体开始
        char thisman = 'A' + k;
        int sum = ( thisman!='A' ) + ( thisman=='C' )
                + ( thisman=='D' ) + ( thisman!='D' );

        if (sum == 3)
        { // 如果3句话为真，则输出当前可能性所假定的人为做好事者
            cout << "做好事者为" << thisman << endl;
            g = 1; // 有解标志置1，表示找到解了
        }
    } // for 循环体结束

    if (g != 1)
    {
        cout << "找不出做好事者" << endl; // 输出无解信息
    }

    return 0 ;
} // L03-03.cpp
```

循环开始前！

循环进行中！

循环结束后！

根据变量g的算法作用，

以下哪些说明是正确的？

```
int g = 0 ;    // 初始化为0，表示“无解”。在枚举开始前根据算法需要设置初值。
for (...; ...; ...) {
    ...
    if (...) {
        ...
        g = 1; // 有解标志置1，表示找到解了
    }
} // for 循环体结束
```

A

变量g必须在循环开始前设置初值。

B

变量g的初值可以设置成任意值。

C

设置有解标志时，变量g的值可以随便赋一个与初始值不同的值。

D

初始值和必须和有解标志值不同。

提交

算法优化 —— 当需要**提前结束**循环时

如果循环没执行完就发现当前可能性成立，即四句话中有3句为真，是否可以**立即**退出循环不再继续呢？

```
for (int k=0; k<4; k++)  
{  
    char thisman = 'A' + k;  
    int sum = ( thisman!='A' ) + ( thisman=='C' )  
              + ( thisman=='D' ) + ( thisman!='D' );  ///// 变量含义为“真话数量”  
    if (sum == 3) // 判断真话数量是否为3（即是否有3句是真）?  
    {  
        // 如果是，则输出当前可能性所假定的人为做好事者  
        cout << "做好事者为" << thisman << endl;  
        break;  
    }  
}
```

使用“break语句”，立即中止当前循环，结束for语句

这里是for语句的结束点

【任务3.1】方法3：彻底数字化

数字 0	表示 A
数字 1	表示 B
数字 2	表示 C
数字 3	表示 D

让 `int k` 表示要找的人，`k`从0变化到3，分别表示从A找到D。
这时：

A说：不是我；	可写成表达式	<code>k != 0;</code>
B说：是C；	可写成表达式	<code>k == 2;</code>
C说：是D；	可写成表达式	<code>k == 3;</code>
D说：他胡说；	可写成表达式	<code>k != 3;</code>

任务3.1的参考源代码简化版 V2

```
#include <iostream>
using namespace std;
int main()
{
    int g = 0;                // 声明变量为整数类型，初始化为0
    for(int k=0; k<4; k++)    // 循环从k为0到3，
    {
        A说      B说      C说      D说
        if ((k != 0) + (k == 2) + (k == 3) + (k != 3)) == 3)
        {
            cout << "做好事者为" << char('A'+k) << endl; // 输出做好事者
            g = 1;                // 让有解标志置1
        }
    }
    if (g != 1)                // 则输出无解信息
    {
        cout << "找不出做好事者" << endl;
    }
    return 0;
} // L03-04.cpp
```

【学而时习之】强制类型转换

```
cout << "做好事者为" << char('A'+k) << endl;
```

表达式 'A' + k 的结果是一个整数。为了输出做好事者的“名字”（字母），需要把这个整数值（字符的ASCII值）转换成为对应的字母。不是随便转换，而是要按照ASCII表的编码规则进行转换。这个转换可以通过“强制类型转换”来实现，即“类型名（表达式）”来实现。

```
#include <iostream>
using namespace std;
```

```
int main()
{
    cout << 'A' + 3 << ' ' << char('A'+3) << endl;
    return 0;
} // E03-04.cpp
```


`char('A'+2)` 的结果: [填空1]

`int(3.14159)` 的结果: [填空2]

`int('D')` 的结果 (请查附录) : [填空3]

`int(3.68)` 的结果: [填空4]

【任务3.2】百家姓游戏卡片

上周思考题解答

如果游戏中的最大数（比如100）允许用户自由来设定，又该如何编程来生成所需的卡片呢？

用姓氏序号1~100来
代表各姓氏



【任务3.2】编程输出猜姓氏游戏中卡片上的数字。

程序设计策略：自顶向下，逐步求精

```
#include <iostream>
using namespace std;
```

先写框架，逐步求精

```
int main()
```

```
{
```

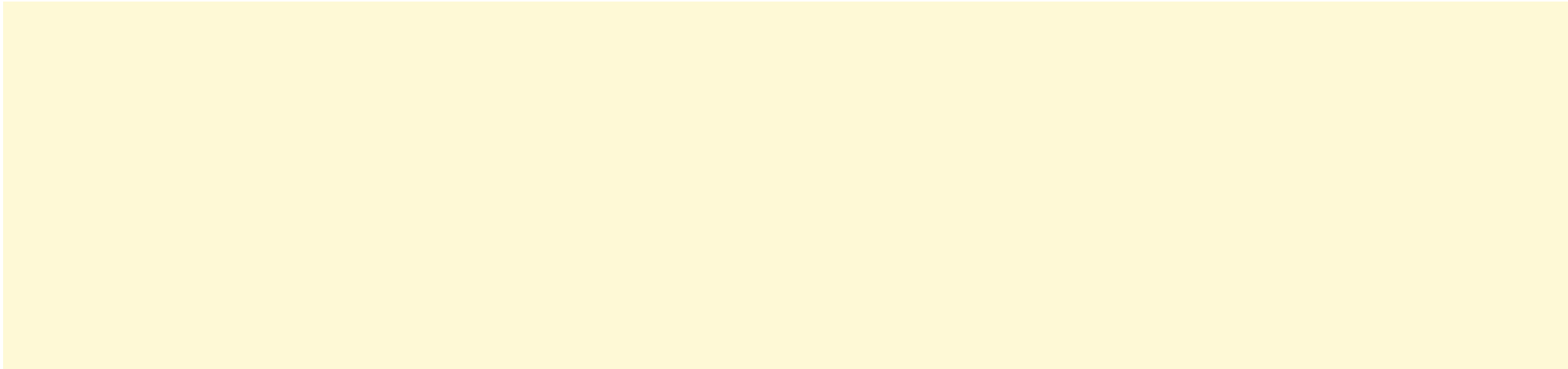
```
    return 0;
```

```
}
```

程序设计策略：自顶向下，逐步求精

```
#include <iostream>
using namespace std;
```

依次输出各张卡片的数字

```
int main()
{
    for (int m=0; m<7; m++) // m 是卡片编号，百家姓需要7张卡片
    {
        
    }
    return 0;
}
```

程序设计策略：自顶向下，逐步求精

```
#include <iostream>
using namespace std;
```

卡片内容的开头和结尾

```
int main()
{
    for (int m=0; m<7; m++) // m 是卡片编号
    {
        cout << "CARD " << m << endl;
        // 哪些数在这张卡片上?
        cout << endl << endl; /// 每张卡片输出之间空一行
    }
    return 0;
}
```

程序设计策略：自顶向下，逐步求精

```
#include <iostream>
using namespace std;
```

枚举1~100所有数字

```
int main()
{
    for (int m=0; m<7; m++) // m 是卡片编号
    {
        cout << "CARD " << m << endl;
        for (int i = 1; i <= 100; i++) /// 循环中的循环
            // 逐一枚举1-100中的数，看是否“符合条件”
            ;
        cout << endl << endl; /// 每张卡片输出之间空一行
    }
    return 0;
}
```

程序设计策略：自顶向下，逐步求精

```
#include <iostream>
using namespace std;
```

判断数字是否符合要求

```
int main()
{
    for (int m=0; m<7; m++)
    {
        cout << "CARD " << m << endl;
        for (int i = 1; i <= 100; i++)
        {
            if (i & (1 << m)) cout << i << ' ';
            cout << endl << endl;
        }
        return 0;
    }
}
```



获得指定位置bit值

cout << endl << endl; /// 每张卡片输出之间空一行

二进制位运算

运算符	运算
&	与运算
	或运算
^	异或运算
~	反码
<<	左移
>>	右移

示例

$6 \& 3 = 2$

$6 | 3 = 7$

$6 \wedge 3 = 5$

$\sim 6 = -7$

$3 \ll 2 = 12$ $3 * 2 * 2 = 12$

$3 \gg 1 = 1$ $3 / 2 = 1$

```
#include <iostream>
using namespace std;

int main()
{
    cout << (6 & 3) << endl;
    cout << (6 | 3) << endl;
    cout << (6 ^ 3) << endl;
    cout << (~6) << endl;
    cout << (3 << 2) << endl;
    cout << (3 >> 1) << endl;

    return 0;
}
```

C++取反运算: https://blog.csdn.net/fyq_sdut/article/details/105628117

计算机的反码存储与计算规则: https://blog.csdn.net/weixin_43167418/article/details/108332557

二进制位运算：求指定位置的bit值

整数 **i**

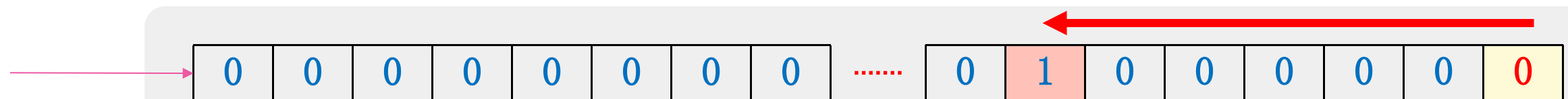


第 **m** 位的bit值是多少？（从低位向高位数，第 **m** 位）

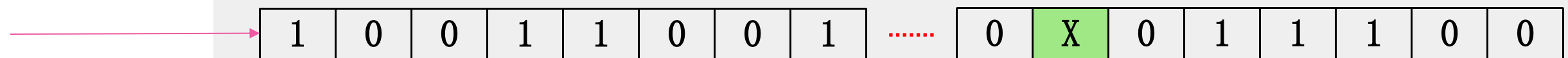
整数 **1**



$1 \ll m$



i



$i \& (1 \ll m)$



X: 0 or 1

程序设计策略：自顶向下，逐步求精

获得指定位置bit值

```
#include <iostream>
using namespace std;

int main()
{
    for (int m=0; m<7; m++)
    {
        cout << "CARD " << m << endl;
        for (int i = 1; i <= 100; i++) /// 获得指定位置bit值
            if (i & (1 << m)) cout << i << ' ';
        cout << endl << endl; /// 每张卡片输出之间空一行
    }
    return 0;
} // L03-05.cpp
```

程序运行的输出结果

CARD 0

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61
63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99

CARD 1

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31 34 35 38 39 42 43 46 47 50 51 54 55 58 59 62
63 66 67 70 71 74 75 78 79 82 83 86 87 90 91 94 95 98 99

CARD 2

4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31 36 37 38 39 44 45 46 47 52 53 54 55 60 61 62
63 68 69 70 71 76 77 78 79 84 85 86 87 92 93 94 95 100

CARD 3

8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31 40 41 42 43 44 45 46 47 56 57 58 59 60 61
62 63 72 73 74 75 76 77 78 79 88 89 90 91 92 93 94 95

CARD 4

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 48 49 50 51 52 53 54 55 56 57 58 59 60 61
62 63 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95

CARD 5

32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
62 63 96 97 98 99 100

CARD 6

64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100

【任务3.3】谁做的坏事？

某地刑侦大队对涉及六个嫌疑人的一桩疑案进行分析，以下内容叙述的是刑侦队搜集到的线索和情报：

- A、B 至少有一人作案；
- A、E、F 三人中至少有两人参与作案；
- A、D 不可能是同案犯；
- B、C 或同时作案，或与本案无关；
- C、D 中有且仅有一人作案；
- 如果 D 没有参与作案，则 E 也不可能参与作案。

请你编写程序，将作案人找出来。

如何解决此类问题？

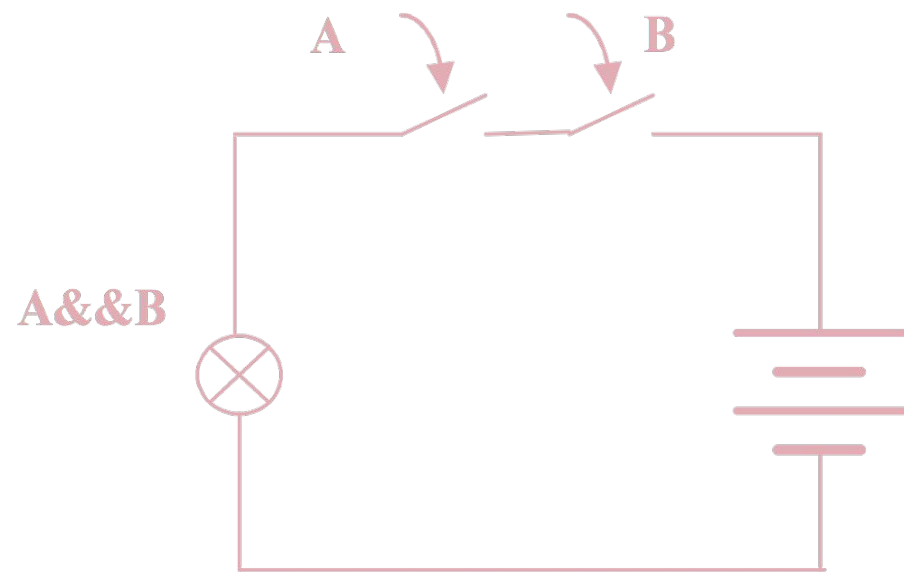
这是另一类逻辑推理问题。难点和关键还是如何用计算机编程语言来表达自然语言的语义和关系。

为了用计算机编程来求解此类题目，需要学习掌握如下一些程序设计的基本技巧：

- (1) 逻辑运算符
- (2) 逻辑表达式

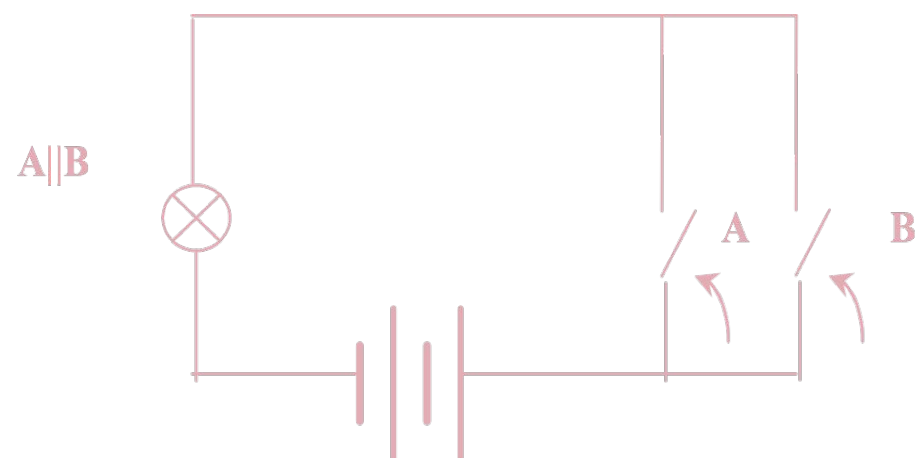
逻辑与运算&&、逻辑或运算||

布尔变量**A**表示开关**A**的开合状态；布尔变量**B**表示开关**B**的开合状态。当**A**取值为0（假）时，表示开关断开；取值为1（真）时，表示开关合上。



逻辑运算式 $A \& B$ 表示 灯是否点亮；

结果为0：灯是熄的；结果为1：灯是亮的。



逻辑运算式 $A || B$ 表示 灯是否点亮；

结果为0：灯是灭的；结果为1：灯是亮的。

作业“回头再看”——奇数且3倍数

```
int main()
{
    double sum = 0;
    int cnt = 0, tmp;

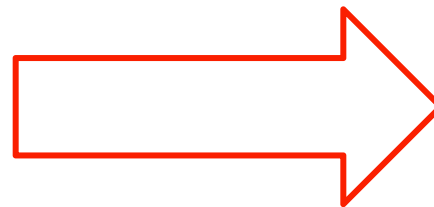
    for (int i=0; i<6; i++)
    {
        cin >> tmp;
        if (tmp % 2 == 1 && tmp % 3 == 0)
        {
            sum += tmp;
            cnt ++;
        }
    }

    if (sum > 0.0)
        sum /= cnt;
    cout << fixed << setprecision(4) << sum << endl;

    return 0;
} // OJ02-02D.cpp
```

作业“回头再看”——判断闰年

```
int main()
{
    int y;
    cin >> y;
    if (y % 4 > 0)
    {
        cout << "NO";
        return 0;
    }
    if (y % 100 > 0)
    {
        cout << "YES";
        return 0;
    }
    if (y % 400 > 0)
    {
        cout << "NO";
        return 0;
    }
    cout << "YES";
    return 0;
}
```



请在课后自行补充

逻辑非运算！

下面以两支队比赛篮球来说明逻辑运算（与、或、非）的语义。

若以变量 A 表示 “A 队到场” 这个命题（即：若A为真，则 ‘A’ 队到场是事实；若A为假，则 ‘A’ 队未到场是事实），变量 B 表示 “B 队到场” 这个命题，则：表达式 !A 表示 ‘A’ 队不到场命题，表达式 !B 表示 ‘B’ 队不到场命题。

若要球赛能赛成，则显然两队都得到场。

若变量C表示 “两队能赛成” 这个命题，则 $C = A \ \&\& \ B$ 。

若变量D表示 “两队赛不成” 这个命题，则 $D = !C$ ；

或者 $D = !A \ || \ !B$ ，即 ‘A’ 队不到场或 ‘B’ 队不到场。

以下哪些说法是正确的?

A

如果变量A表示'A'在现场, 则!A表示'A'不在现场

B

变量A表示'A'是否在现场, 若其值为true, 则表示'A'在现场

C

若变量A表示'A'不在现场, 则!A表示'A'在现场

D

若true表示在现场, false表示不在现场, 则根据变量A的值, 可以知道'A'是否在现场

提交

任务3.3解题思路：案情分析

将叙述案情的每一条线索视为一个“命题”，用一个逻辑变量表示，将线索的内容以逻辑表达式来表示，即：

- 第一条线索用 CC1 表示
- 第二条线索用 CC2 表示
-
- 第六条线索用 CC6 表示

接下来将讲解如何将“线索内容”转为“逻辑表示式”！

CC1: A和B至少有一人作案

线索CC1是否成立，与A和B的作案状态是密切相关的。“A和B至少....”的意思是：“要么一人，要么两人”。若令逻辑变量A表示A是否作案的状态，逻辑变量B表示B是否作案的状态，则上面的线索内容写成逻辑表示式来表示就是： $CC1 = A \vee B$

证明如下

A \vee B	A	B	含义	CC1
0	0	0	没有作案	0
1	1	0	A作案	1
1	0	1	B作案	1
1	1	1	A和B都作案	1

CC2: A和D不可能是同案犯

可做如下分析:

- 如果 A 和 D 是同案犯, 则应写成 $A \ \&\& \ D$
- 所以, 对于“A 和 D 不可能是同案犯”, 应写成 $!(A \ \&\& \ D)$

因此有 $CC2 = !(A \ \&\& \ D)$

以上几条结论, 都可以使用前一页的方法, 用下面的表格来证明。

$A \ \&\& \ D$	A	D	含义	CC2
0	0	0	均不作案	1
0	1	0	A作案	1
0	0	1	D作案	1

CC3: A、E、F中至少有两人涉嫌作案

经过分析，这条线索有三种可能（即在如下三种情形下线索为真）

- 第1种：A 和 E 作案，表示为 (A && E)
- 第2种：A 和 F 作案，表示为 (A && F)
- 第3种：E 和 F 作案，表示为 (E && F)

从线索内容看，以上三种可能性之间是 **或** 的关系，因此有

$$CC3 = (A \ \&\& \ E) \ || \ (A \ \&\& \ F) \ || \ (E \ \&\& \ F)$$


CC4: B和C或同时作案，或都与本案无关

经过分析，这条线索有两种可能

- 第1种：同时作案，表示为 $(B \ \&\& \ C)$
- 第2种：都与本案无关，表示为 $(!B \ \&\& \ !C)$

两种可能的情况是 或 的关系，因此有

$$CC4 = (B \ \&\& \ C) \ || \ (!B \ \&\& \ !C)$$

CC5: C、D中有且仅有一人作案

经过分析，这条线索有两种可能

- 第1种：C作案且D不作案，表示为 $(C \ \&\& \ !D)$
- 第2种：D作案且C不作案，表示为 $(D \ \&\& \ !C)$

两种可能的情况是 或 的关系，因此有

$$CC5 = (C \ \&\& \ !D) \ || \ (D \ \&\& \ !C)$$

CC6: 如果D没有参与作案, 则E也不可能参与作案

经过分析, 这条线索有两种可能性: (1) D作案; (2) D未作案

- 对于可能性 (1) D作案: 依题意, E可能作案, 也可能不作案, 说明在此种情况下, 线索CC6与E是否作案的状态无关, 因此, 直接用D来描述。
- 对于可能性 (2) D未作案: 根据线索陈述的要求, E应该是不作案的。因此, 线索CC6可表示为: $!D \ \&\& \ !E$

综上所述, $CC6 = D \ || \ (\ !D \ \&\& \ !E \)$

【注意】在上式中, 运算符“||”的右边隐含着!D的意思, 所以右半部分中的 !D 可以去掉, 即: $CC6 = D \ || \ !E$

CC6: 如果D没有参与作案, 则E也不可能参与作案

经过分析, 这条线索有两种可能性: (1) D作案; (2) D未作案

- 对于可能性 (1) D作案: 依题意, E可能作案, 也可能不作案, 说明在此种情况下, 线索CC6与E是否作案的状态无关, 因此, 直接用D来描述。
- 对于可能性 (2) D未作案: 根据线索陈述的要求, E应该是不作案的。因此, 线索CC6可表示为: $!D \ \&\& \ !E$

综上所述, $CC6 = D \ || \ (\ !D \ \&\& \ !E \)$

【注意】在上式中, 运算符“||”的右边隐含着!D的意思, 所以右半部分中的 !D 可以去掉, 即: $CC6 = D \ || \ !E$

任务3.3解题思路：枚举所有可能性

总体思路：6个嫌疑人，每个人都有作案或不作案两种可能，因此有 2^6 种组合，从这些组合中筛选出符合条件的嫌疑人（作案者）。

算法实现：定义6个整型变量，分别表示6个人A, B, C, D, E, F是否作案的状态，用0表示不作案，用1表示作案。

关键技术：如何枚举 2^6 种可能性的组合？即如何依次（有顺序）得到这些组合？

解决方案：使用“多重循环”技术！即：每一重循环对应一个人，用来对他的是否作案的两种可能性分别进行测试！

在任务3.3中使用多重循环语句

算法实现：定义 6 个整型变量A, B, C, D, E, F, 分别表示 6 个人 是否作案的状态, 0表示不作案, 1表示作案。

```
for (int A=0; A<=1; A++)
{
    for (int B=0; B<=1; B++)
    {
        for (int C=0; C<=1; C++)
        {
            ... /// 继续对DEF三人是否作案进行循环枚举
        } /// END_for_3
    } /// END_for_2
} /// END_for_1
```


在多重循环语句的循环体中进行判断

总体思路： 6个嫌疑人，每个人都有作案或不作案两种可能，因此有 2^6 种组合，从这些组合中筛选出符合条件的嫌疑人（作案者）。

```
... /// 从A到E的外层循环
for (int F=0; F<=1; F++) /// 最内层的循环（最后一个人）
{
    cc1 = A || B;
    cc2 = !(A && D);
    cc3 = (A && E) || (A && F) || (E && F);
    cc4 = (B && C) || (!B && !C);
    cc5 = (C && !D) || (D && !C);
    cc6 = D || !E;
    if (cc1 + cc2 + cc3 + cc4 + cc5 + cc6 == 6)
    {
        /// ...
    }
} /// END_for_6
...
```

```
#include <iostream>
using namespace std;

int main()
{
    int cc1, cc2, cc3, cc4, cc5, cc6;
    for (int A=0; A<=1; A++)
        for (int B=0; B<=1; B++)
            for (int C=0; C<=1; C++)
                for (int D=0; D<=1; D++)
                    for (int E=0; E<=1; E++)
                        for (int F=0; F<=1; F++)
                        { /// 最内层循环，最后一个人
                            cc1 = A || B;
                            cc2 = !(A && D);
                            cc3 = (A && E) || (A && F) || (E && F);
                            cc4 = (B && C) || (!B && !C);
                            cc5 = (C && !D) || (D && !C);
                            cc6 = D || !E;
                            if (cc1 + cc2 + cc3 + cc4 + cc5 + cc6 == 6)
                            {
                                此处代码见下一页
                            }
                        }
    }
```

```
cout << "A: ";  
if (A)  
    cout << "是罪犯";  
else  
    cout << "不是罪犯";  
cout << endl;  
  
cout << "B: ";  
if (B)  
    cout << "是罪犯";  
else  
    cout << "不是罪犯";  
cout << endl;  
  
cout << "C: ";  
if (C)  
    cout << "是罪犯";  
else  
    cout << "不是罪犯";  
cout << endl;
```

```
cout << "D: ";  
if (D)  
    cout << "是罪犯";  
else  
    cout << "不是罪犯";  
cout << endl;  
  
cout << "E: ";  
if (E)  
    cout << "是罪犯";  
else  
    cout << "不是罪犯";  
cout << endl;  
  
cout << "F: ";  
if (F)  
    cout << "是罪犯";  
else  
    cout << "不是罪犯";  
cout << endl;
```

条件运算符：？

在输出每个人是否为罪犯时，可以使用下面的语句：

```
cout << "A: " << (A==1 ? "是罪犯" : "不是罪犯")  
    << endl;
```

其中，`(A==1 ? "是罪犯" : "不是罪犯")` 条件运算符 `？：`，该运算符的表达式由三部分组成，即

条件判断式 `？` 结果1 `:` 结果2

如果 条件判断式 为真，则表达式返回 结果1；
如果 条件判断式 为假，则表达式返回 结果2。

使用条件运算符，简化代码

常规写法

```
cout << "A: ";
if (A)
    cout << "是罪犯";
else
    cout << "不是罪犯";
cout << endl;

cout << "B: ";
if (B)
    cout << "是罪犯";
else
    cout << "不是罪犯";
cout << endl;

cout << "C: ";
if (C)
    cout << "是罪犯";
else
    cout << "不是罪犯";
cout << endl;

cout << "D: ";
if (D)
    cout << "是罪犯";
else
    cout << "不是罪犯";
cout << endl;

cout << "E: ";
if (E)
    cout << "是罪犯";
else
    cout << "不是罪犯";
cout << endl;

cout << "F: ";
if (F)
    cout << "是罪犯";
else
    cout << "不是罪犯";
cout << endl;
```

注意：

在cout语句中，要用括号（）
把条件运算式括起来！

简洁写法



```
cout << "A: " << (A == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "B: " << (B == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "C: " << (C == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "D: " << (D == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "E: " << (E == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "F: " << (F == 1 ? "是罪犯" : "不是罪犯") << endl;
```

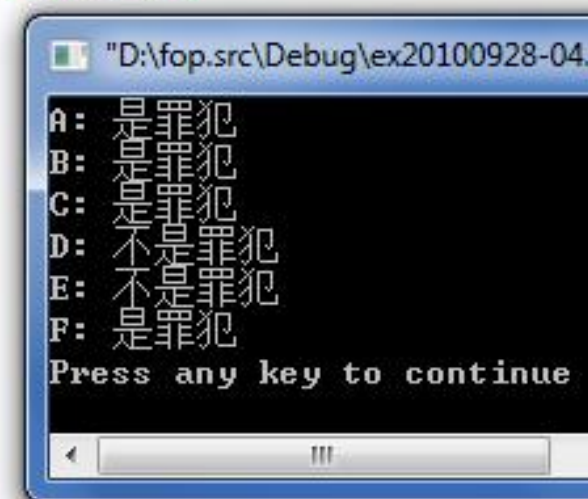

任务3.3的参考源代码和输出结果

```
#include <iostream>
using namespace std;

int main()
{
    int cc1, cc2, cc3, cc4, cc5, cc6; //定义6个变量,分别表示6句话
    //A和B至少有一人作案;          cc1 = A || B
    //A和D不可能是同案犯;          cc2 = !(A && D)
    //A, E, F中至少有两人涉嫌作案 cc3 = (A && E) || (A && F) || (E && F)
    //B和C或同时作案或都与本案无关 cc4 = (B && C) || (!B && !C)
    //C, D中有且仅有一人作案;      cc5 = (C && !D) || (D && !C)
    //如果D没有参与作案,则E也不可能参与作案; cc6 = D || (!E)

    for (int A=0; A<=1; A=A+1) // 枚举A
        for (int B=0; B<=1; B=B+1) // 枚举B
            for (int C=0; C<=1; C=C+1) // 枚举C
                for (int D=0; D<=1; D=D+1) // 枚举D
                    for (int E=0; E<=1; E=E+1) // 枚举E
                        for (int F=0; F<=1; F=F+1) // 枚举F
                        {
                            cc1 = A || B;
                            cc2 = !(A && D);
                            cc3 = (A && E) || (A && F) || (E && F);
                            cc4 = (B && C) || (!B && !C);
                            cc5 = (C && !D) || (D && !C);
                            cc6 = D || (!E);
                            if (cc1 + cc2 + cc3 + cc4 + cc5 + cc6 == 6)
                            { // 测试6句话都为真时,才输出谁是罪犯
                                cout << "A: " << (A == 1 ? "是罪犯" : "不是罪犯") << endl;
                                cout << "B: " << (B == 1 ? "是罪犯" : "不是罪犯") << endl;
                                cout << "C: " << (C == 1 ? "是罪犯" : "不是罪犯") << endl;
                                cout << "D: " << (D == 1 ? "是罪犯" : "不是罪犯") << endl;
                                cout << "E: " << (E == 1 ? "是罪犯" : "不是罪犯") << endl;
                                cout << "F: " << (F == 1 ? "是罪犯" : "不是罪犯") << endl;
                            } // _IF_结束
                        } // _FOR_结束

    return 0;
}
```



课后思考题1

不使用6重循环，如何对 2^6 种组合进行枚举？

某地刑侦大队对涉及六个嫌疑人的一桩疑案进行分析，以下内容叙述的是刑侦队搜集到的线索和情报：

A、B 至少有一人作案；

- A、E、F 三人中至少有两参与作案；
- A、D 不可能是同案犯；
- B、C 或同时作案，或与本案无关；
- C、D 中有且仅有一人作案；
- 如果 D 没有参与作案，则 E 也不可能参与作案。

请你编写程序，将作案人找出来。

课后思考题2

能否进一步简化下面的代码？

```
cout << "A: " << (A == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "B: " << (B == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "C: " << (C == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "D: " << (D == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "E: " << (E == 1 ? "是罪犯" : "不是罪犯") << endl;
cout << "F: " << (F == 1 ? "是罪犯" : "不是罪犯") << endl;
```

至少有3种不同的办法可以简化上述代码


```
#include <iostream>
using namespace std;

int main()
{
    cout << "End." " See you later!" << endl;
    return 0;
}
```