Chapter 4

邏輯運算與迴圈

基本邏輯運算

- 假：數字 0 或者是空字元 '\0'
  真：非假的數值一律定為真

```
真： -1 , 1.0 , '0' , "" , "Orz" , true

假： 0 , 0.0 , '\0' , false
```

❖ 空字串也為真

邏輯運算子 (一)

- ! ： 否定運算子
  && ： 而且運算子
  || ： 或者運算子

!a

a	真	假
!a	假	真

a && b

a \ b	真	假
真	真	假
假	假	假

a || b

a \ b	真	假
真	真	真
假	真	假

logical operator

邏輯運算子 (二)

- 否定運算子的執行順序較 && 與 || 高

```
bool a = true ;
bool b = !a ;          // b = false
bool c = !a && b ;     // c = false
```

以上先執行 否定運算子 !a

再執行 而且運算子 !a && b

比較運算子（一）

- < ：小於
- > ：大於
- == ：等於
- <= ：小於或等於
- >= ：大於或等於
- != ：不等於

❖ == ：數值比較

= ：數值指定

```
int a = 3;
cout << (a==2) << '\n';   // 印出假值即為 0
cout << (a=2)  << '\n';   // 讓 a 為 2 後印出 2
```

comparison operator

比較運算子 (二)


- ```
  bool a = (3 < 5) ;  // a true
  bool b = (1 != 3);  // b true
  ```

- ```
  bool c = (0.1+0.1+0.1==0.3);
  ```

  這裡的 **c** 為 **false**，原因為截去誤差

- ```
  bool d = ( 1 <= x && x <= 5 );
  ```

  ```
  bool e = ( 1 <= x <= 5 ); // 錯誤
  ```
  ```

短路運算子

- && 和 || 有時也稱為 短路運算子

  若左邊運算式的結果已足以確認整個邏輯式子的結果時，則運算式右邊的式子即予以跳過不執行

邏輯運算子	左邊運算元	右邊運算元	輸出值
&&	假	不執行	假
&&	真	執行	與右邊運算元同真假
||	真	不執行	真
||	假	執行	與右邊運算元同真假

short-circuit operator

if 條件敘述式 (一)

- 若 A 為真假值判定式而 B 為一般敘述

```
(1) if ( A ) B ;
```

如果 A 為真，則執行 B

```
(2) if ( A )
        B1;
    else
        B2;
```

如果 A 為真，則
　執行 B1
否則
　執行 B2

if 條件敘述式（二）


```
(3) if ( A1 )
       B1 ;
    else if ( A2 )
       B2 ;
    else if ( A3 )
       B3 ;
```

如果 **A1** 為真，則
　　執行 **B1**
否則如果 **A2** 為真，則
　　執行 **B2**
否則如果 **A3** 為真，則
　　執行 **B3**

```
(4) if ( A1 )
       B1 ;
    else if ( A2 )
       B2 ;
    else
       B3 ;
```

如果 **A1** 為真，則
　　執行 **B1**
否則如果 **A2** 為真，則
　　執行 **B2**
否則就
　　執行 **B3**

if 條件敘述式（三）

- 利用簡單的 **if** 將大寫字母轉為小寫字母：

```
char c;
cin >> c;

if( c >= 'A' && c <= 'Z' )
   c = static_cast<char>('a'+c-'A');

cout << c;
```

if 條件敘述式 (四)

- 如果 if 敘述內又有 if 敘述，則儘量用大括號包住以避免混淆

```
// (1)
if (A1) {
    if (A2) B1;
} else
    B2;
```

```
// (2)
if (A1)
    if (A2) B1;
else
    B2;
```

```
// (3)
if (A1) {
    if (A2)
        B1;
    else
        B2;
}
```

❖ 以上(2)式相當於(3)式

一元二次方程式

- $ax^2 + bx + c = 0$

  則

  $b^2 - 4ac \geqq 0$ ⇨ 二個實數根

  $b^2 - 4ac < 0$ ⇨ 二個複數根

程式 輸出

if 敘述常犯的錯誤


```
//錯誤的程式
#include <iostream>
using namespace std;
int main(){
      int foo;
      cin >> foo;
      if ( foo < 0 )
            cout << "數字小於零\n";
      else if ( foo = 0 )
            cout << "數字等於零\n";
      else
            cout << "數字大於零\n";
      return 0;
}
```

指定 foo 為 0 後回傳 0

A ? B : C 條件敘述 (一)


```
if( A )
   B ;
else
   C ;
```

```
A ? B : C ;
```

```
if( A )
   foo = B ;
else
   foo = C ;
```

```
foo = (A ? B : C);
```

❖ B , C 的資料型別須相同

英文字元替換

舊：a b c d e f g h i j k l m n o p q r s t u v w x y z

新：n o p q r s t u v w x y z a b c d e f g h i j k l m

```
char c;
cin >> c;

if ( c >= 'A' && c <= 'Z' )                    // c 為大寫字母
   c = (c < 'N') ? c + 13 : c - 13 ;
else if ( c >= 'a' && c <= 'z' )               // c 為小寫字母
   c = (c < 'n') ? c + 13 : c – 13 ;

cout << c << endl ;
```

A ? B : C 條件敘述 (二)

- 與輸出 cout 合用：

```
int no;
cin >> no;
cout << ( no > 10 ? 10 : no ) << endl;
```

- 切記冒號前後的型別必須一致：

```
// 錯誤
cout << ( no == 1 ? "one" : no );
```

switch 敘述(一)


```
if ( A == 1 )
   B ;
else if ( A == 2 )
   C ;
else
   D ;
```

```
switch( A ){
   case 1:
      B ; break ;
   case 2:
      C ; break ;
   default :
      D ;
}
```

```
if( A == 'a' || A == 'b' )
   B ;
else
   C ;
```

```
switch( A ){
   case 'a': case 'b':
      B ; break ;
   default :
      C ;
}
```

switch 敘述（二）

- **switch** 內所能比較的變數型別只能是整數類型 (integral type)
  也就是資料是以整數方式儲存的

```
double  A ;
switch( A ){      // 錯誤 ， A 須為整數類型
    ....
}
```

- 列舉型別是以整數方式儲存資料

```
enum  Season { spring , summer , fall, winter };
Season foo ;
switch ( foo ) {
    case 'spring' : ....        // 正確
}
```

❖ **switch** 內各 **case** 的比較是以 " **==**" 來運算的

for 迴圈 (一)


```
for( A ; B ; C )  D ;
```

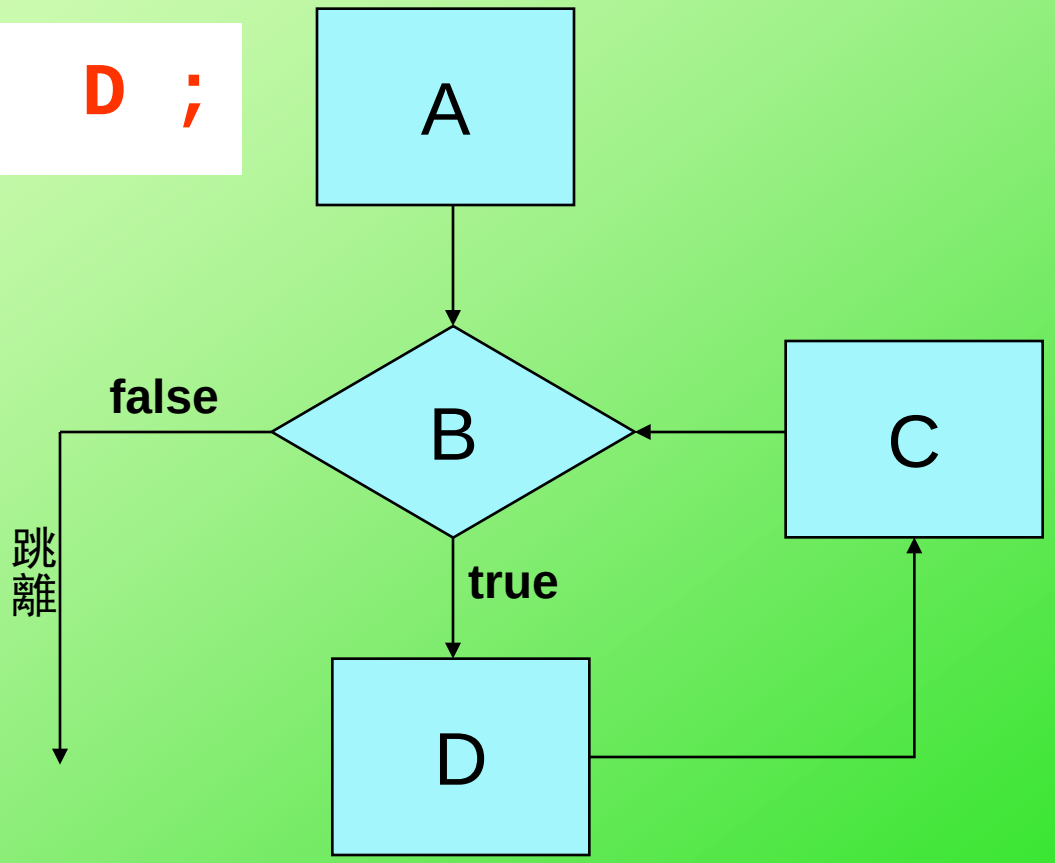
- 執行順序：

  ABDCBDCBDC...

- **A,B,C,D**

  皆可省略

  但分號須保留



for 迴圈 (二)

- 列印小寫英文字母：

```
for( int i = 0 ; i < 26 ; ++i )
   cout << static_cast<char>('a' + i)
        << endl;
```

- 計算 1 到 100 之間的奇數和：

```
int i , sum = 0;
for( i = 1; i <= 100 ; i += 2 ) sum += i ;
cout << sum << '\n';
```

for 迴圈 (三)

■ 迴圈內也可以有迴圈：

```
#include <iomanip>
...
int i , j , n = 4;

for( i = 0 ; i < n ; ++i ){
   for( j = 0 ; j < 2*i+1 ; ++j ){
      if (j==0) cout << setw(n-i) << " " ;
   }
   cout << '\n';
}
```

數字金字塔

```
4
333
22222
1111111
```

❖ **setw(x)** 是以 **x** 格的空間來列印其後的資料
使用時需要加入 **iomanip** 標頭檔

橫式九九乘法表

橫式九九乘法表：

```
1x1= 1 2x1= 2 3x1= 3 4x1= 4 5x1= 5 6x1= 6 7x1= 7 8x1= 8 9x1= 9
1x2= 2 2x2= 4 3x2= 6 4x2= 8 5x2=10 6x2=12 7x2=14 8x2=16 9x2=18
1x3= 3 2x3= 6 3x3= 9 4x3=12 5x3=15 6x3=18 7x3=21 8x3=24 9x3=27
1x4= 4 2x4= 8 3x4=12 4x4=16 5x4=20 6x4=24 7x4=28 8x4=32 9x4=36
1x5= 5 2x5=10 3x5=15 4x5=20 5x5=25 6x5=30 7x5=35 8x5=40 9x5=45
1x6= 6 2x6=12 3x6=18 4x6=24 5x6=30 6x6=36 7x6=42 8x6=48 9x6=54
1x7= 7 2x7=14 3x7=21 4x7=28 5x7=35 6x7=42 7x7=49 8x7=56 9x7=63
1x8= 8 2x8=16 3x8=24 4x8=32 5x8=40 6x8=48 7x8=56 8x8=64 9x8=72
1x9= 9 2x9=18 3x9=27 4x9=36 5x9=45 6x9=54 7x9=63 8x9=72 9x9=81
```

程式 輸出

直式九九乘法表

直式九九乘法表：

```
  1     2     3     4     5     6     7     8     9
x 1   x 1   x 1   x 1   x 1   x 1   x 1   x 1   x 1
---   ---   ---   ---   ---   ---   ---   ---   ---
  1     2     3     4     5     6     7     8     9

  1     2     3     4     5     6     7     8     9
x 2   x 2   x 2   x 2   x 2   x 2   x 2   x 2   x 2
---   ---   ---   ---   ---   ---   ---   ---   ---
  2     4     6     8    10    12    14    16    18

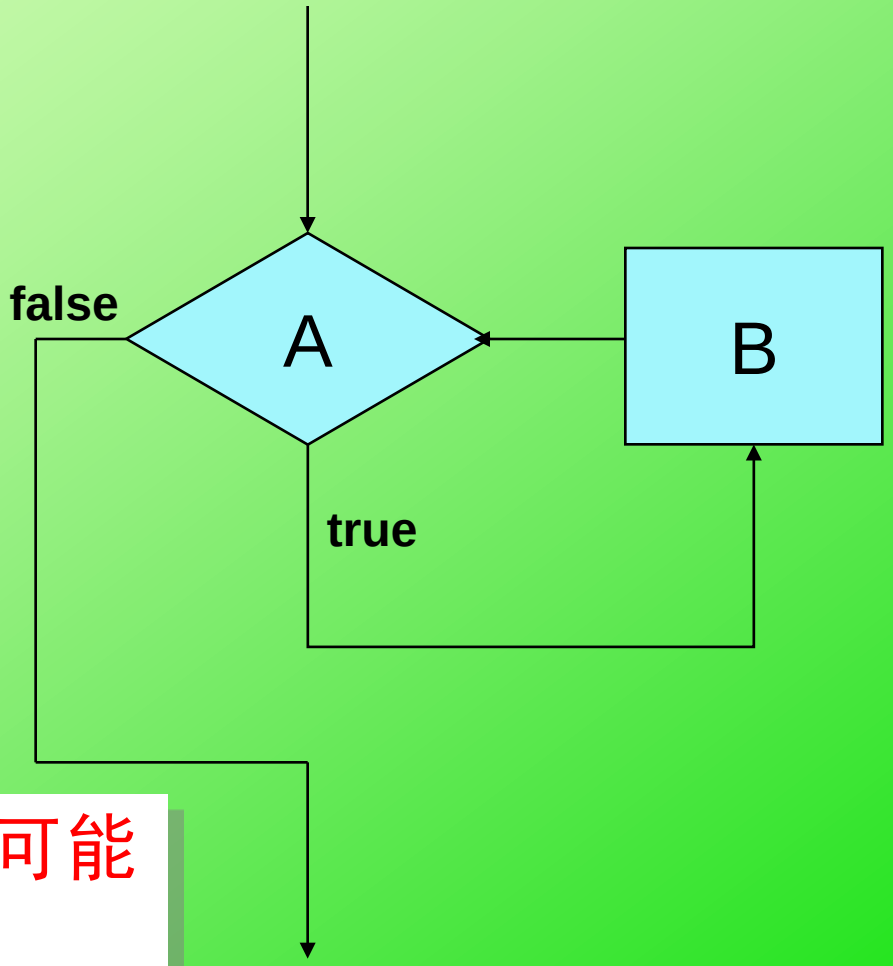
 ...
  1     2     3     4     5     6     7     8     9
x 9   x 9   x 9   x 9   x 9   x 9   x 9   x 9   x 9
---   ---   ---   ---   ---   ---   ---   ---   ---
  9    18    27    36    45    54    63    72    81
```

程式 輸出

while 迴圈 (一)


```
while ( A ) B ;
```

- 當 A 為真時重複執行 B
- 若 A 一開始即不滿足
  則 B 就不會被執行



❖ 迴圈內的敘述式 B 也有可能不會被執行到

while 迴圈 (二)

- while 迴圈也可以換成 for 迴圈

while 迴圈型式：

```
int i , n;
cin >> n ;

i = 1 ;
while ( i <= n ){
    cout << i*i << endl ;
    ++i;
}
```

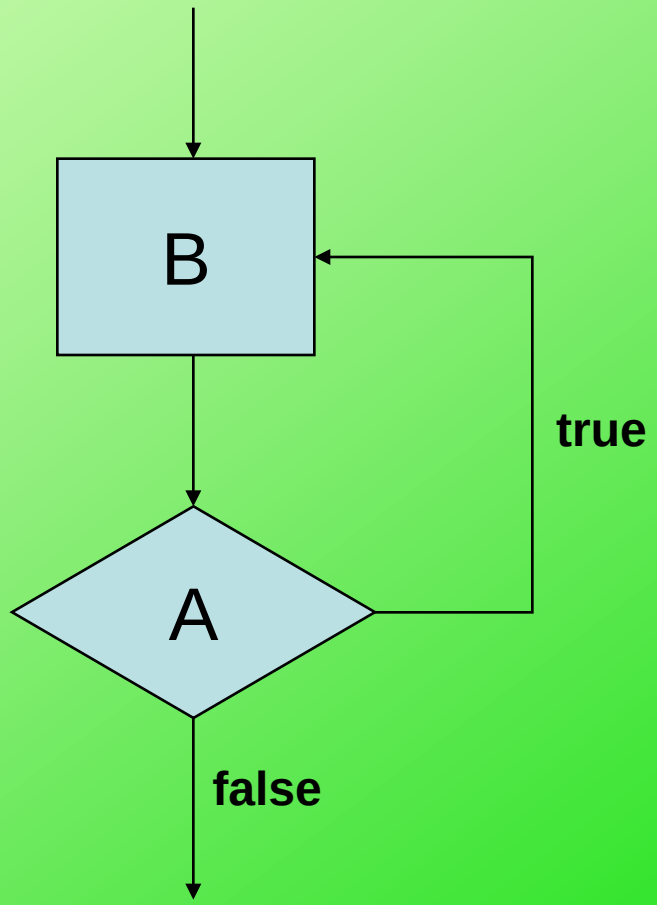
for 迴圈型式：

```
int i , n;
cin >> n ;

for( i = 1 ; i <= n ; ++i ){
    cout << i*i << endl ;
}
```

do-while 迴圈 (一)


```
do{
    B
} while( A );
```



- 重複執行 **B** 直到 **A** 為假
- **B** 至少會被執行一次

do-while 迴圈 (二)

■猜數字遊戲

```
...
int main{
    int no , guess , i ;
    no = rand()%5 + 1 ;        // 須加入 cstdlib 標頭檔
    i  = 0;
    do {
        cout << "> 請猜一位 1 .. 5 的數字:" ;
        cin >> guess ;
        ++i ;
        if( no != guess ) cout << "> 沒猜對\n" ;
    } while( no != guess );
    cout << "> 猜對了, 總共猜了 " << i << "次\n" ;
    return 0;
}
```

break (一)

- 在迴圈內可使用 break 提早跳出迴圈：

```
int i ;
while ( 1 ) {
    cin >> i ;
    // 若 i 為負數則直接跳出迴圈
    if (i <= 0) break ;
    cout << i*i << endl ;
}
```

break (二)

- 若程式碼中有多重迴圈則 break 僅跳出一層迴圈

```
int i , j ;
for ( i = 0 ; i < 3 ; ++i ){
   for ( j = 0 ; j < 3 ; ++j ) {
      // break 僅跳出一層迴圈
      if( i+j >= 3 ) break ;
      cout << "*" ;
   }
   cout << endl;
}
```

```
***
**
*
```

continue

- 若迴圈內使用 continue 則可讓程式提早進入下一輪的迴圈迭代執行

```
int j = 3 , i ;
for( i = 1 ; i <= 10; ++i ){
    // 若 i 為 3 的倍數則提早進入下一輪迭代
    if( i%j == 0 ) continue ;
    cout << i << ' ';
}
```

```
1 2 4 5 7 8 10
```

goto

- 在多層迴圈內，可使用 goto 直接跳到最外層執行

```
int i , j ,k ;
cin >> no;
for( i = 1 ; i < 1000 ; ++i ){
   for( j = 1 ; j < 1000 ; ++j ){
      for( k = 1 ; k < 1000 ; ++k ){
         if( (i*j*k) == no ) goto OUTSIDE ;
      }
   }
}
OUTSIDE:
cout << "i = " << i << " , j = " << j <<" , k = "<< k << endl;
```

- 以上的 **OUTSIDE** 為使用者自定的跳出標幟，其後須有冒號
  goto 會破壞程式執行的連貫性須避免使用

迴圈中的變數存在領域（一）

- 變數存在領域：變數存在的區間

```
int i = 9 ;
for( int j = 1 ; j <= 10 ; ++j ){
      int k = i*j;
      cout << k << '\n';
}
```

以上：

- i 的存在領域 跨越整個迴圈
- j 的存在領域 僅在迴圈內
- k 的存在領域 僅在迴圈內做一次迭代

scope

迴圈中的變數存在領域（二）

- 無效率的程式設計：

```
for( int i = 0 ; i < 100000 ; ++i ) {
   for( int j = 0 ; j < 100000 ; ++j ) {
       int k ;
          ...
   }
}
```

- 改良後的程式碼：

```
int i , j , k ;
for( i = 0 ; i < 100000 ; ++i ){
   for( j = 0 ; j < 100000 ; ++j ){
          ...
   }
}
```

靜態變數

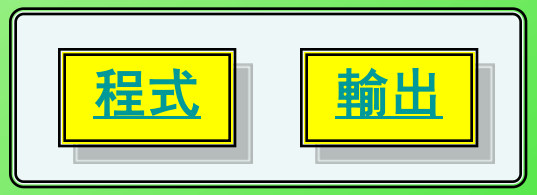
- 讓暫時變數轉成永久型變數的機制，使得變數在離開其存在領域後繼續存在
- 使用時在變數的型別之前加入 **static**

```
for ( int i = 0 ; i < 10 ; ++i ){
    int a = 0;                          // a 為暫時變數
    static int b = 0;                   // b 為靜態變數，初值為 0
    ++a;     ++b;                       // 各自累加一
    cout << "> a : " << a               // a : 1 1 1 1 ... 1
         << "> b : " << b               // b : 1 2 3 4 ... 10
         << endl;
}
cout << "> b " << endl ;                // 錯誤，b 僅存在迴圈內
```

static

質數問題

- 檢查某數 n 是否為質數：
  - n == 2，則 n 是質數
  - n 不是個大於 2 的偶數
  - 由 3 開始，每次增加 2，檢查 n 可否整除此數
    直到此數的平方大於 n 為止
    若是則 n 不是質數，否則為質數
- 質數檢查：

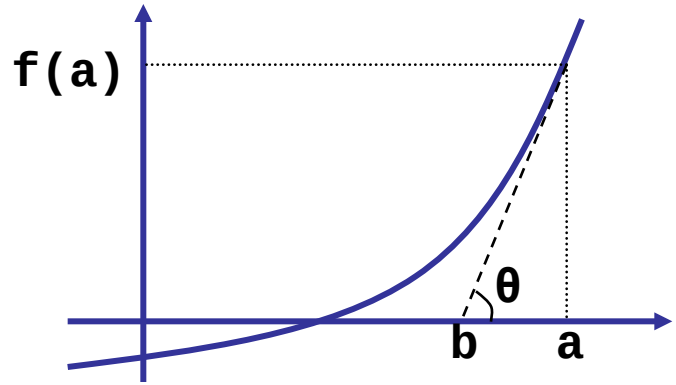


數值求根法：牛頓迭代法 (一)

- 求 f(x) = 0 的根：

  重複迭代 $b = a - \frac{f(a)}{f'(a)}$

  直到 b 和 a 的值非常靠近為止



求 √n 的值，則可以讓

$$f(x) = x^2 - n$$
$$f'(x) = 2x$$

$$b = a - \frac{f(a)}{f'(a)} = a - \frac{a^2 - n}{2a}$$

$$= 0.5\left(a + \frac{n}{a}\right)$$

Newton's iteration

數值求根法:牛頓迭代法(二)

- 以牛頓迭代法公式而言

  其程式代碼 (pseudo code) 可以寫成以下方式：

```
1. 輸入 n
2. 讓 a = 10 及設定最大容許誤差 SMALL
3. 計算 b = 0.5(a + n/a)
4. 計算 b 與 a 的距離
5. 如果 4 式的值大於最大容許誤差 SMALL
   則讓 a = b 且回到 3 式再執行，否則印出 b 後結束
```

- 牛頓迭代法：

程式　輸出

有趣的數學問題

- 找尋字母所代表的數字

```
      FORTY
        TEN
+       TEN
-------------
      SIXTY
```

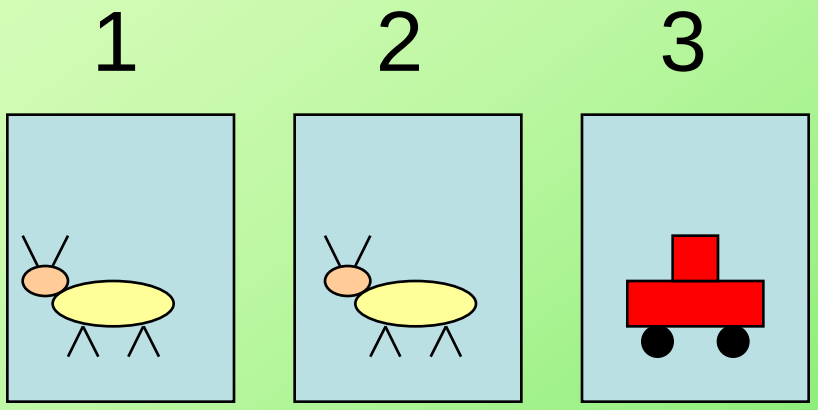
- 蠻力法：

使用計算機高速運算的特性來檢查問題的所有可能性

程式 輸出

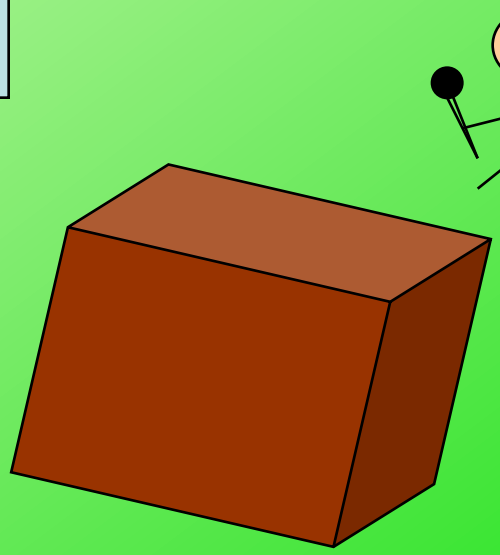
brute force method

羊與汽車（一）



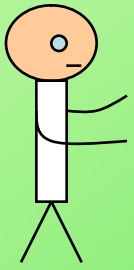
這3扇門裡只有一扇門有車，你猜是選一扇門呢?

主持人

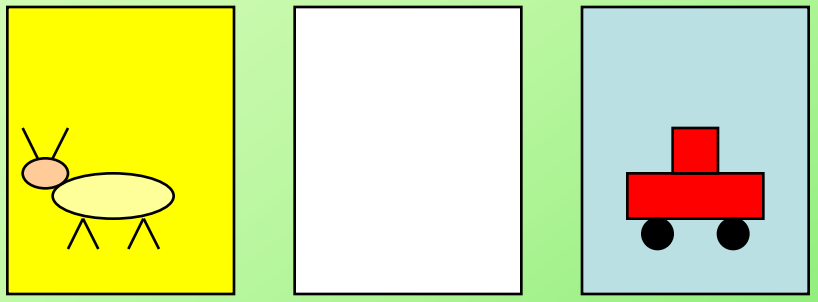


我猜第一扇門裡是車

來賓

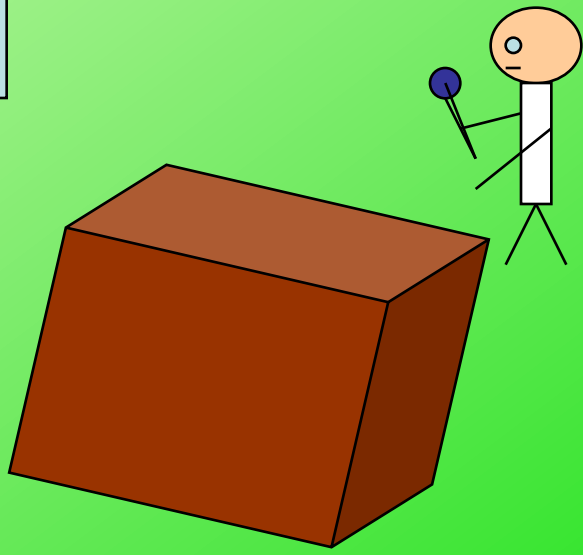


羊與汽車 (二)



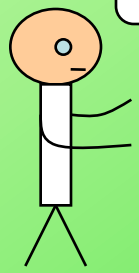
好，那我打開第 2 扇門給你看 是羊喔!! 那請問你要不要和第3扇門交換阿...??

主持人



讓我考慮一下...

來賓



羊與汽車 (三)

- 機率：
  - 不換門但中獎的機率：$\frac{1}{3}$
  - 換門　但中獎的機率：

$$1/3 \times 0 + 2/3 \times 1 = \frac{2}{3}$$

程式　輸出

換門後中獎機率為 1

未猜對門的機率為 2/3

換門後中獎機率為 0

未猜對門的機率為 1/3

動態螢幕顯示（一）

■ 美國標準控制序列指令：顯示器與週邊設備

跳離序列	功用
esc [n A	游標往上移 n 列(row)
esc [n B	游標往下移 n 列
esc [n C	游標往右移 n 列(column)
esc [n D	游標往左移 n 列
esc [2 J	清除螢幕且游標回歸左上角
esc [K	清除在游標之後的同列文字
esc [s	儲存游標位置
esc [u	回復游標位置
esc [att m	設定字元輸出的顯示模式
esc [r ; c H	將游標移到 r列 c行的位置 (螢幕左上角為第一列第一行)

➢ 跳離字元(esc)： \x1b

動態螢幕顯示（二）

- 清除畫面後將 **HELLO** 字串於第二列第一行處顯示

```
cout << "\x1b[2J"
     << "\x1b[2;1H" << "HELLO" << flush ;
```

// 這裡 **flush** 會將資料立即輸出到螢幕上顯示

- 將數字 **55** 顯示於第 **X** 列,第 **Y** 行

```
int x , y;
...
cout << "\x1b[" << x << ";" << y << "H"
     << 55 << flush ;
```

動態螢幕顯示（三）

■ 字元顯示模式：

基本顯示模式	功用	基本顯示模式	功用
0	回復正常顯示	1	增亮顯示
2	減亮顯示	3	斜體字顯示
4	加入字元底線	5	字元閃爍顯示
6	快速字元閃爍	7	前景背景顏色
8	隱形顯示		

■ 字元顯示顏色：

- 前景：3x　　- 後景：4x

0	黑 (black)	1	紅 (red)	2	綠 (green)	3	黃 (yellow)
4	藍 (blue)	5	洋紅 (magenta)	6	青綠 (cyan)	7	白 (white)

動態螢幕顯示（四）

- 輸出的資料 ：

  字元顏色為藍色　字元背景為洋紅色

```
cout << "\x1b[34;45m" << "hello"
     << "\x1b[0m"     << flush ;
```

- 簡單的螢幕保護程式：

程式