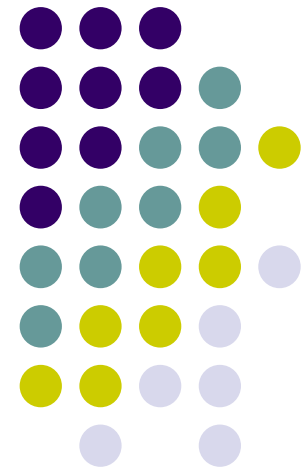


計算組合數 $C(n,k)$

丁培毅



計算 C_k^n and P_k^n



$$\frac{n!}{k!(n-k)!}$$

- 版本一 **long int**

```
01 #include <stdio.h>    17 int find_nCk(int n, int k) {
02 #include <stdlib.h>   18     return factorial(n) / (factorial(k)*factorial(n-k));
03 int factorial(int);    19 }
04 int find_nCk(int, int);
05 int find_nPk(int, int);
06 int main(void) {
07     int n, k, nCk, nPk;
08     printf("Enter the value of n and k\n");
09     scanf("%d%d",&n,&k);
10     nCk = find_nCk(n, k);
11     nPk = find_nPk(n, k);
12     printf("%dC%d = %d\n", n, k, nCk);
13     printf("%dP%d = %d\n", n, k, nPk);
14     system("pause");
15     return 0;
16 }
```

```
20 int find_nPk(int n, int k) {
21     return factorial(n)/factorial(n-k);
22 }
```

```
23 int factorial(int n) {
24     int i, result=1;
25     for (i=1; i<=n; i++)
26         result = result*i;
27     return result;
28 }
```

$$\frac{n!}{(n-k)!}$$

$$P_3^{15} = 15 \cdot 14 \cdot 13 = 2730$$

$$15! = 1307674368000$$

$$2^{31}-1 = 2147483647$$

這樣子寫理論上是對的，
實際上卻有很嚴重的問題!!

計算 C_k^n and P_k^n



- 版本二 **long long int**

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 long long int factorial(int);
04 int find_nCk(int, int);
05 int find_nPk(int, int);
06 int main(void) {
07     int n, k, nCk, nPk;
08     printf("Enter the value of n and k\n");
09     scanf("%d%d",&n,&k);
10     nCk = find_nCk(n, k);
11     nPk = find_nPk(n, k);
12     printf("%dC%d = %d\n", n, k, nCk);
13     printf("%dP%d = %d\n", n, k, nPk);
14     system("pause");
15     return 0;
16 }
```

```
17 int find_nCk(int n, int k) {
18     return factorial(n)/(factorial(k)*factorial(n-k));
19 }
20 int find_nPk(int n, int k) {
21     return factorial(n)/factorial(n-k);
22 }
23 long long int factorial(int n) {
24     int i; long long int result=1;
25     for (i=1; i<=n; i++)
26         result = result*i;
27     return result;
28 }
```

好一些，但是仍然
有很大的限制

$$P_3^{21} = 21 \cdot 20 \cdot 19 = 7980$$

$$21! = 51090942171709440000$$

$$2^{63}-1 = 9223372036854775807$$

計算 C_k^n and P_k^n



- 版本三 **skip unnecessary factorial $n!$, $(n-k)!$**

$$P_k^n = n (n-1) \dots (n-k+1)$$

```
int find_nPk(int n, int k) {  
    int i, result=1;  
    for (i=0; i<k; i++)  
        result = result * (n-i);  
    return result;  
}
```

```
Ex. 30P6 = 427518000  
    30P7 = 1670497408  
    30P8 = -233265280  
    100P5 = 444567808  
    100P6 = -715731200
```

```
long long int find_nPk(int n, int k) {  
    int i;  
    long long int result=1;  
    for (i=0; i<k; i++)  
        result = result * (n-i);  
    return result;  
}
```

```
Ex. 30P13 = 745747076954880000  
    30P14 = -5769043765476591616  
    100P10 = 7475418734400817152  
    100P11 = 8704899442529685504  
    100P12 = -27200710659158016
```

```
printf("result = %I64d\n", result);  
printf("result = %lld\n", result);
```

計算 C_k^n and P_k^n



- 版本三 (cont'd)

```
long long find_nCk(int n, int k) {
    int i;
    long long int numerator=1, denominator=1;
    for (i=0; i<k; i++) {
        numerator *= n-i;
        denominator *= k-i;
    }
    return numerator/denominator;
}
30C14 = -66175233
100C10 = 2060025003968
```

```
long long int find_nCk(int n, int k) {
    int i;
    double result=1.0;
    for (i=0; i<k; i++)
        result *= ((double)(n-i))/(k-i);
    return (long long int) (result+0.5);
}
100C18 = -9223372036854775808
```

$$C_k^n = \frac{n (n-1) \dots (n-k+1)}{k (k-1) \dots 1}$$

```
long long find_nCk(int n, int k) {
    int i;
    long long numerator=1;
    long long denominator=1;
    for (i=0; i<k; i++) {
        numerator *= n-i;
        if (numerator % (k-i) == 0)
            numerator /= k-i;
        else
            denominator *= k-i;
    }
    return numerator/denominator;
}
100C15 = 3327654233778599
```

程式該怎樣寫才能計算出 C_{33}^{67}



- $C(66,33) = 7,219,428,434,016,265,740$
 $2^{63}-1 = 9,223,372,036,854,775,807$
 $C(67,33) = 14,226,520,737,620,288,370$
 $2^{64}-1 = 18,446,744,073,709,551,616$
- 由前面版本三的程式你可以看到直接讓分子乘分子、分母乘分母的話很快就爆表了，第二個程式只是把正下方可以整除的分母除掉就可以看到改善了（沒有試看看其它的分母，也沒有沒有嘗試把公因數除掉）
- 因為 $C(n,k)$ 一定是整數，所有的分母拆解成質因數乘積一定可以和分子的質因數乘積消掉，所以基本的方法就是記住每一個分子和分母，每一個分母都和所有的分子找公因數，除掉公因數

計算 C_{33}^{67}

- 需要計算最大公因數的程式

```
01 int gcd(int p, int q) { // assume p >= q
02     if (p % q == 0)
03         return q;
04     else
05         return gcd(q, p % q);
06 }
```

```
01 int find_gcd(int n1, int n2) {
02     int p, q, r;
03     p = abs(n1);
04     q = abs(n2);
05     r = p % q;
06     while (r != 0) {
07         p = q;
08         q = r;
09         r = p % q;
10     }
11     return q;
12 }
```



- 動手計算看看

C_5^{10}

$$\begin{array}{cccccc}
 2 & 9 & 2 & 7 & 1 & = 252 \\
 \hline
 10 & 9 & 8 & 7 & 6 \\
 \hline
 5 & 4 & 3 & 2 & 1
 \end{array}$$

- 要像上面這樣子算的話，程式需要記住所有部份消去的分子，分母一定都會被消完

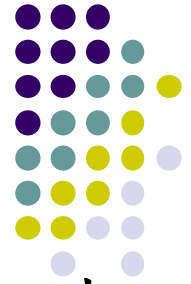
記憶體是很好用的東西



- 可以設計一個陣列紀錄所有的分子 10, 9, 8, 7, 6, 然後每一個分母denum和所有的分子num[j]一個一個找最大公因數d來消去, 直到每一個分母都完全消去成為1 (這個條件不需要檢查, 只需要把所有分子試完)

```
int n=10, k=5, i, j, d, denum, nCk;
int num[67] = {10, 9, 8, 7, 6}; // 先直接放測試範例進去
for (i=k; i>1; i--)
    for (denum=i, j=0; denum>1&& j<k; j++)
        if ((num[j]>1) && ((d=gcd(denum, num[j]))>1))
            denum/=d, num[j]/=d; // 消去 d
for (nCk=1, i=0; i<k; i++) nCk *= num[i];
printf("nCk = %d\n", nCk);
```


如果不允許使用額外的記憶體

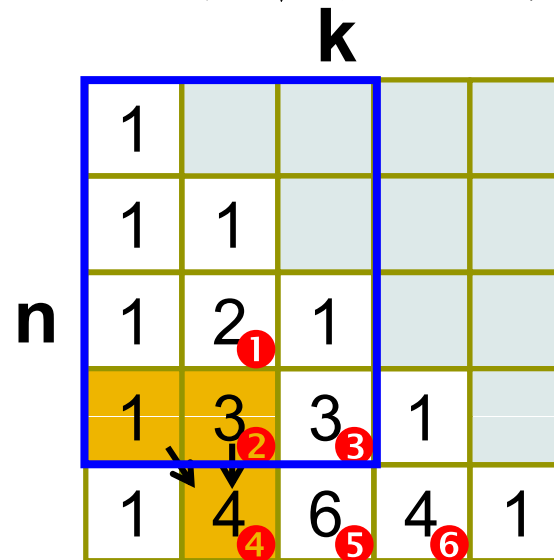
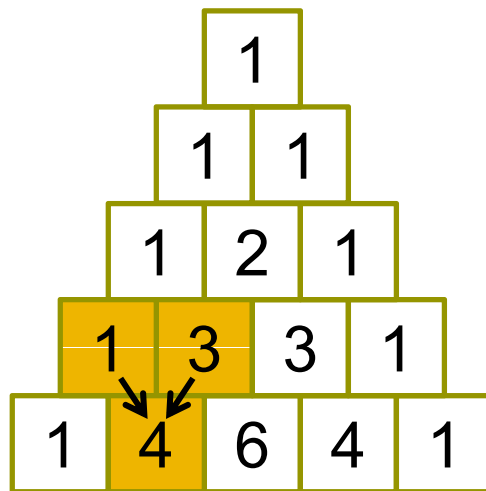


- 版本三的第二個程式就是沒有用陣列來紀錄分子，也沒有考慮分母是可以拆開成為質因數乘積，在不同的分子裡找到可以消去的質因數
- 如果不使用額外記憶體，要稍微改進一下的話，第一就是numerator不需要整除k-i，只需要把最大公因數 $d = \gcd(\text{numerator} * (n-i), \text{denominator} * (k-i))$ 消去就可以，numerator和denominator兩個變數裡累積沒有完全消去的因數，不過這樣子計算時，還是有可能在計算中間數值 $\text{numerator} * (n-i)$ 或是 $\text{denominator} * (k-i)$ 時比計算 $C(n, k)$ 提早產生溢位

如果允許使用更多的記憶體



- 我們可以運用遞推公式來計算 $C(67,33)$
$$C(n,k) = C(n-1,k-1) + C(n-1,k)$$
- 這種計算需要清楚地掌握資料陣列的配置方法以及資料計算的順序，下圖左是一般的帕斯卡三角形
- 下圖右我們用二維的陣列來放 $C(n,k)$, n 是橫列, k 是直行, $k=0$ 和 $n=k$ 時固定都是1, 其它數字計算的順序標示如下



如果只需要計算 $C(n^*,k^*)$
只須計算小於
等於 n^*, k^* 的就
足夠了

帕斯卡三角形



- 定義 `long long comb[67][67];` 的二維陣列

1. 先把固定的數值填好: `comb[*][0], comb[i][i]`

```
for (i=0; i<=n; i++) comb[i][0] = comb[i][i] = 1;
```

2. 由第三列開始一列一列往下填

```
for (i=2; i<=n; i++)
```

```
for (j=1; j<i; j++)
```

```
comb[i][j] = comb[i-1][j-1] + comb[i-1][j];
```

- 如果使用的記憶體數量是很重要的考量, 這個程式也可以只使用兩列的記憶體, `long long comb[2][67];` 由前一列的組合數算本列的組合數, 兩列的記憶體輪流切換; 或是更進一步只用一列的記憶體 `long long comb[67];` 來實作
- 請注意程式多使用記憶體以後演算法變得非常簡潔

帕斯卡三角形 (續)



- 運用 `long long comb[2][67]={1}`; 的二維陣列

```
int i, j, b;  
for (b=1,i=2; i<=n; i++,b=1-b)  
    for (j=1; j<i; j++)  
        comb[b][j] = comb[1-b][j-1] + comb[1-b][j];
```

- 運用 `long long comb[67]={1}`; 的一維陣列

```
int i, j; long long p, t;  
for (i=1; i<n; i++)  
    for (p=j=0; j<i+1; j++,p=t)  
        t=comb[j], comb[j]=p+t;
```

| | | j | | | | | | | | | | |
|---|---|---|---|---|----|----|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | | | | | |
| i | 0 | 0 | p | t | p | t | 0 | 0 | 0 | 0 | | |
| | 1 | 0 | p | t | p | t | 1 | p | t | 0 | 0 | 0 |
| | 2 | 0 | | 1 | | 2 | | 1 | | 0 | 0 | 0 |
| | 3 | 0 | 1 | 3 | 3 | 1 | 0 | 0 | | | | |
| | 4 | 0 | 1 | 4 | 6 | 4 | 1 | 0 | | | | |
| | 5 | | 1 | 5 | 10 | 10 | 5 | 1 | | | | |

更大的組合數 $\text{Comb}(500, 250)$



- 需要使用大整數類別來運算
例如 `Crypto++`, `GMP`, `Java`

```
Integer result;  
result = 1;  
for (i=0; i<n; i++) {  
    d = gcd(m-i, i+1);  
    num = (m-i) / d;  
    denum = (i+1) / d;  
    result /= denum;  
    result *= num;  
}
```

此程式輸入 $m \geq n \geq 0$, 輸出組合數 $\text{comb}(m, n)$:

請輸入 m: 500

請輸入 n: 250

$\text{comb}(500, 250)=$

11674431578827768292093473476217661965923008118031144612410028495

7811112673608473715666417775521605376810865902709989580160037468226

393900042796872256.

偵測「溢位」- 純軟體



```
#define INT64_MIN_VALUE (0x8000000000000000LL)
#define INT64_MAX_VALUE (0x7FFFFFFFFFFFFFFFLL)
int isMultOverflow(long long a, long long b) {
    long long ap, bp, cp, max=INT64_MAX_VALUE;
    if (a==INT64_MIN_VALUE) return b!=1;
    if (b==INT64_MIN_VALUE) return a!=1;
    ap = a >= 0 ? a : -a;
    bp = b >= 0 ? b : -b;
    cp = max / bp;
    return (ap > cp);
}
```

計算每一個乘法之前, 先判斷一下是否會發生溢位, 不會的話才乘起來

等於是花兩倍的時間
像是計算 $\text{Comb}(500, n)$ 真的不知道 64 位元
可以算到多大的 n 值, 只好邊算邊測試

偵測「溢位」- 硬體狀態旗標



- 主要運用CPU的指令 **JNO** (jump no overflow)

```
gcc/g++  
  
int isOverflow()  
{  
    __asm__ ("movl $0, %eax;");  
    __asm__ ("jno end;");  
    __asm__ ("movl $1, %eax;");  
    __asm__ ("end:");  
}
```

如果偵測位置離開
運算敘述太遠就會
沒有效果

```
Visual C/C++  
  
int isOverflow()  
{  
    _asm mov    eax, 0  
    _asm jno   end  
    _asm mov    eax, 1  
    _asm end:  
}
```

vc2010 把64位元乘法
用一個函式來完成, 這
種偵測就完全無效

C++ Exception Handling (1/3)



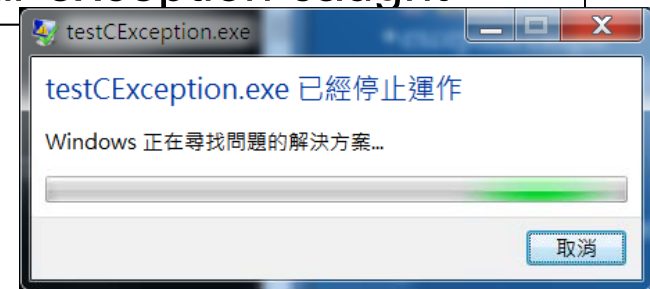
- There are three exception handling mechanisms you can use with C++ on MS Windows:
 - **C++ exceptions**, implemented by the compiler (**try/catch**), cl /EHsc
 - **Structured Exception Handling (SEH)**, provided by Windows (**__try / __except**), cl /EHa
 - **MFC exception macros** (TRY, CATCH - built on top of SEH / C++ exceptions - see also TheUndeadFish's comment)
- C++ exceptions guarantee **automatic cleanup** during stack unwinding (i.e. dtors of local objects), other mechanisms don't.
- C++ exceptions only occur when they are **explicitly thrown**. Structured Exceptions may occur for any operations (especially **divide by zero**, H/W, and invalid memory access, S/W).
- MFC introduces the macros to support exception handling even if compilers didn't implement them.

C++ Exception Handling (2/3)



```
for (y=3; y>=0; y--)  
  try {  
    int x=1, *z=0;  
    if (y==3) throw exception("explicit");  
    else if (y==2) throw 'a';  
    else if (y==1) *z = 10;  
    else  
      x = x/y;  
    cout << "end\n";  
  }  
  catch (exception) {  
    cout << "exception caught\n";  
  }  
  catch (char e) {  
    cout << "char exception caught\n";  
  }  
  catch (...) {  
    cout << "... caught\n";  
  }  
}
```

```
cl /EHsc  
testCException.cpp  
exception caught  
char exception caught
```



```
cl /EHa testCException.cpp  
exception caught  
char exception caught  
... caught  
... caught
```

C++ Exception Handling (3/3)



```
for (y=3; y>=0; y--)
```

```
    __try {  
        int x=1, *z=0;  
        if (y==3) RaiseException(15,0,0,NULL);  
        else if (y==2) throw 'a';  
        else if (y==1) *z = 10;  
        else  
            x = x/y;  
        cout << "end\n";  
    }
```

```
    __except (FilterFunction(GetExceptionCode())) {  
        cout << GetExceptionCode() << " exception caught\n";  
    }
```

```
DWORD FilterFunction(unsigned int code) {  
    if (code == 15)  
        return EXCEPTION_CONTINUE_SEARCH;  
    else  
        return EXCEPTION_EXECUTE_HANDLER;  
}
```

Something like rethrow

Both cl /EHsc or cl /EHa work.