第 2 章
神奇的質數

質數在自然數中佔有非常重要的地位，質數是一類既簡單又神秘的數字。說其簡單，是因為小學生也知道什麼是質數；說其神秘，是因為從古至今，多少數學家都想弄明白質數的規則，卻一直沒有找到其分佈規律。

數學家都沒研究出來的規律，程式設計師當然也不可能會找到。但是，任何事物都有正反兩面，正是由於質數的無規律特點，在密碼學中就可以大量採用。另外，在一些齒輪嚙合設計中，也通常將齒輪的齒數設計成質數，以增加兩齒輪中兩個相同的齒相遇嚙合次數的最小公倍數，這樣可增強齒輪的耐用度，減少故障。

2.1 怎麼判斷質數

怎麼判斷質數呢？首先需要對質數進行定義，然後根據其定義判斷指定的數是不是質數。對程式設計師來說，可以按質數的定義編寫相應程式對質數進行判斷。

2.1.1 什麼是質數

數學中的定義是這樣的：質數，又稱為素數，是指在一個大於 1 的自然數中，除了 1 和此整數自身外，無法被其他自然數整除的數。也可說質數是只有 1 和本身兩個因數的數。

比 1 大但不是質數的數稱為合數。1 和 0 既非質數也非合數。

根據質數的定義，可用如下圖所示方式列出10以內各數的因數，從而得出2、3、5、7 為質數，而 4、6、8、9 不是質數。

2的因數：1、2　　✓ 質數
3的因數：1、3　　✓ 質數
4的因數：1、2、4　　× 質數
5的因數：1、5　　✓ 質數
6的因數：1、2、3、6　　× 質數
7的因數：1、7　　✓ 質數
8的因數：1、2、4、8　　× 質數
9的因數：1、3、9　　× 質數

從上面的因數分解可看出，10 以內共有 4 個質數。隨著資料的增大，質數的分佈頻率將變得更稀少，10000 以內的質數共有 1229 個，由於篇幅所限不逐一列出，下面列出 1000 以內的 168 個質數，如下圖所示。

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67
71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151
157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239
241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337
347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433
439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541
547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743
751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857
859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971
977 983 991 997

對上圖所列出的 1000 以內的質數進行總結，可看出在以 100 為間隔的區間中質數的個數是沒有規律的，其中：

- 100 以內的質數有 25 個。

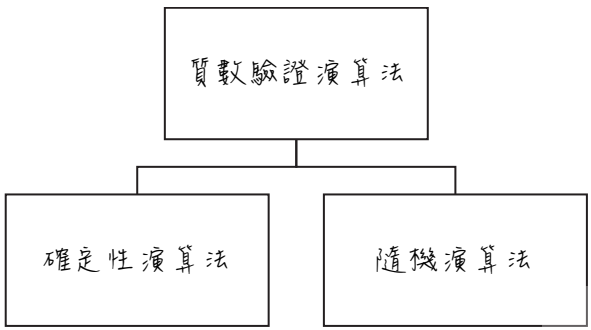
- 100 ~ 200 之間的質數有 21 個。
- 200 ~ 300 之間的質數有 16 個。
- 300 ~ 400 之間的質數有 16 個。
- 400 ~ 500 之間的質數有 17 個。
- 500 ~ 600 之間的質數有 14 個。
- 600 ~ 700 之間的質數有 16 個。
- 700 ~ 800 之間的質數有 14 個。
- 800 ~ 900 之間的質數有 15 個。
- 900 ~ 1000 之間的質數有 14 個。

目前最大的已知質數是 $2^{57885161}-1$（此數字位數長度是 17425170），它是在 2013 年 1 月 25 日由 GIMPS 發現的。該組織還在 2008 年 8 月 23 日發現了目前所知第二大的已知質數 $2^{43112609}-1$（此數字位元長度是 12978189）。

2.1.2　驗證質數

在上頁第二個圖中列出了 1000 以內的質數，究竟這些數字是不是質數呢？需要進行驗證。

驗證一個自然數是否為質數，這個問題在中世紀就引起了人們的注意，當時人們試圖尋找一個公式一勞永逸地解決問題，到了高斯時代，基本上確認了簡單的質數公式是不存在的，因此，高斯認為對素性判定是一個相當困難的問題。從此以後，這個問題吸引了大批數學家。 驗證質數的演算法可分為兩大類，即確定性演算法及隨機演算法，如下圖所示。



確定性演算法可得出確定的結果，但通常演算法較慢，而隨機演算法正好相反。透過電腦進行運算，可解決演算法較慢的問題，其演算法的實現也很簡單。

確定性演算法中最常用的就是試除法。試除法是根據質數的定義得出的一種方法，用需要驗證的數 *N* 逐個除以從 2 開始至 *N*-1 中的所有數，若能被某一個數整除，表示有一個因數，說明數 *N* 不是質數；若直到 *N*-1 都不能被整除，則說明該數是質數。

根據以上思考方式，可編寫以下的試除法函數：

```
int is_prime(int n)
{
   int i;

   if(n <= 1)                                  //1 不是質數，參數 n<=1 時返回 0
   {
      return 0;
   }

   for(i = 2; i < n; i++)
   {
      if(n % i == 0)                           //判斷 n 是否可以被 i 整除
      {
         return 0;
      }
   }

   return 1;
}
```

其實，可以對質數的定義進行進一步的分析。要判斷數 *N* 是否為質數，不需要用 *N* 一直除到 *N*-1 才能確認，而只需要除到 $\sqrt{n}$ 就可以了。例如，*N*=15，則可以整除 15 的除數有 1、3、5，對於除數 5 就不用判斷，因為 *N* 被 3 整除時其商就是 5，也就表示 *N* 能被 5 整除了。

因此，為了減少迴圈判斷的次數，提高程式的執行效率，可將函數 is_prime() 進行修改，以提高程式的效率。

```
int is_prime(int n)
{
   int i;

   if(n <= 1)                              //1 不是質數，參數 n<=1 時返回 0
   {
      return 0;
```

```
    }

    for(i = 2; i * i <= n; i++)
    {
        if(n % i == 0)                    //判斷是否能被 i 整除
        {
            return 0;
        }
    }

    return 1;
}
```

從上面的程式可看到，在 for 迴圈中以 i^2 與 n 值進行比較，就可以顯著地減少迴圈的次數，從而提高驗證的效率。

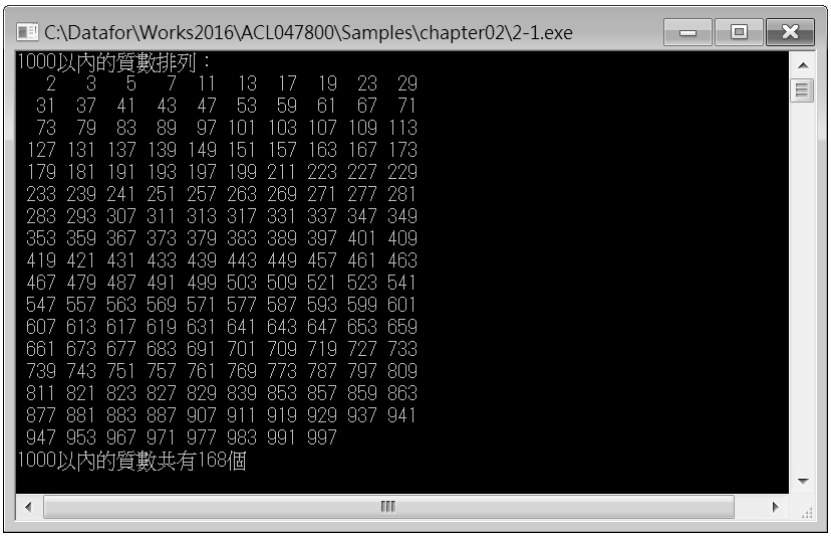
2.1.3　尋找質數的演算法

透過驗證方法可以驗證某個整數是否為質數，而尋找質數的方法，就是尋找在給定限度內的所有質數排列。例如，要求出 1000 以內的所有質數，就是一個尋找質數排列的問題。由於已經有上面定義的 is_prime() 函數，求出 1000 以內所有質數的方法就很簡單了，可以用以下程式來完成。

```
#include <stdio.h>
#include<conio.h>

int is_prime(int n)
{
    int i;

    if(n <= 1)                                    //1 不是質數，參數 n<=1 時返回 0
    {
        return 0;
    }
     for(i = 2; i * i <= n; i++)
    {
        if(n % i == 0)                            //判斷是否能被 i 整除
        {
            return 0;
        }
    }
    return 1;
}

int main()
{
    int i,n=0,t=1;
```

```
    printf("1000 以內的質數排列：\n");

    for(i=2;i<1000;i++)
    {
        if(is_prime(i))
        {
            n++;
            t++;
            printf("%4d",i);                    //輸出質數
            if(t>10)                            //每輸出 10 個質數就換行
            {
                printf("\n");
                t=1;
            }
        }
    }
    printf("\n1000 以內的質數共有%d 個\n",n);

    getch();
    return 0;
}
```

執行以上程式，可得到 1000 以內的所有質數列表，如下圖所示。



在上面的程式碼中，透過 is_prime() 函數來驗證指定區間（2～1000）中的每一個數是否為質數，而 is_prime() 函數中又透過迴圈進行驗證。這種雙迴圈會導致程式執行效率不高。

這時可考慮採用另一種尋找質數的演算法：著名的 Eratosthenes 求質數方法。下面演示如何用這種方法求 100 以內的質數。

Eratosthenes 演算法假設有一個篩子，用來存放 2～100 之間的所有數，如下圖（a）所示。

由於偶數都能被 2 整除，因此偶數都不是質數（2 除外），將這些數篩除，可得到如下圖（b）所示的資料（設定了背景色的數字將被篩去）。

接著再將 3 的倍數篩除，得到如下圖（c）所示結果。

接下來繼續將 5 的倍數篩除，得到下圖（d）所示結果。

最後再將 7 的倍數篩除，得到如下圖（e）所示的結果，即可得到 100 以內的所有質數。

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	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	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

（a）

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	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	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

（b）

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	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	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

（c）

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	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	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

（d）

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	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	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

（e）

從上圖中可看到，在使用 Eratosthenes 演算法進行篩選時，只需要執行 4 次篩選就完成了 100 以內的質數的篩選，效率非常高。如果要篩選的資料範圍更大，由於只需要選擇已經篩選過的質數對後面的數進行篩選，因此可快速篩選出後面的質數。

從上圖中的演算法過程可以看出，Eratosthenes 演算法比試除法的效率要高得多。

根據上圖中所示的過程編寫對應的程式，具體程式碼如下：

```
#include <stdio.h>
#include<conio.h>

#define MAXNUM 1000                          //求 1000 以內的所有質數
int main()
{
    int i,j,c=0;
    int prime[MAXNUM+1];                     //定義保存質數的陣列
    for(i=2;i<=MAXNUM;i++)                   //初始化陣列
        prime[i]=1;                          //旗標為 1 表示對應的數是質數

    for(i=2;i*i<=MAXNUM;i++)                 //迴圈處理前 i 個
    {
        if(prime[i]==1)                      //若為質數，則進行篩選
        {
            for(j=2*i;j<=MAXNUM;j++)         //篩去合數
            {
                if(!prime[j]) continue;      //是合數，則跳過
                if(j%i==0)                   //數 j 能被整除，說明不是質數
                    prime[j]=0;              //清除旗標
            }
        }
    }

    for(i=2;i<MAXNUM;i++)                    //輸出質數
    {
        if(prime[i]==1)                      //是質數，則輸出
```

```
        {
            printf("%4d ",i);                    //輸出質數
            c++;
            if(c%10==0)                          //每行輸出 10 個質數
               printf("\n");
        }
    }

    printf("\n 共有%d 個質數！",c);
    getch();
    return 0;
}
```

2.1.4　已被證明的質數定理

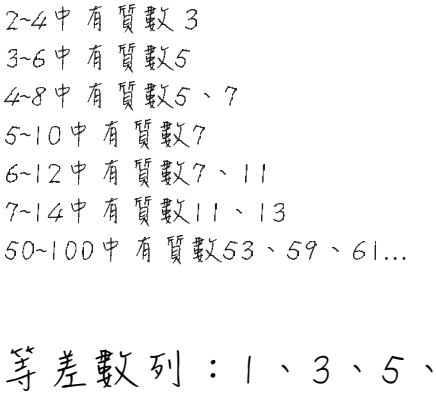
自古以來就有很多數學家研究質數，因此，得出了許多與質數有關的定理。下面簡單介紹一些已被證明的質數定理。

1．在 (a,2a] 之間必有一個質數

在一個大於 1 的數 a 和它的 2 倍之間（即區間 $(a, 2a]$ 中）必存在一個質數。如下圖所示，可看到在 $(a, 2a]$ 區間中都至少包含一個質數。

2．存在任意長度的質數等差數列

什麼是等差數列呢？這是一個古老的數學課題。一個數列從第二項起，從後項減去前項所得的差是一個相同的常數，則這個數列就被稱為等差數列。



用質數構成的等差數列被稱為質數等差數列。例如從 5 開始，以 12 為間隔常數，就可以得到這樣的序列：

5、17、29、41、53、65 …

對這個數列來說，只有前5個數是質數，第6個數65能被5整除，不是質數，因此，在這裡得到的是由5個質數構成的質數等差數列：

5、17、29、41、53

還有更長的質數等差數列嗎？當然有，如下面的 10 個質數就構成間隔為 210 的質數等差數列：

199、409、619、829、1039、1249、1459、1669、1879、2089

2004 年 4 月 18 日，格林和陶哲軒兩人宣佈：他們證明了「存在任意長度的質數等差數列」，也就是說，對於任意值 *K*，存在 *K* 個成等差級數的質數。例如 *K*=3，有質數序列 3、5、7（兩數之間差 2）……*K*=10，有質數序列 199、409、619、829、1039、1249、1459、1669、1879、2089（兩數之間差 210）。他們將長達 50 頁的論文——《質數含有任意長度的等差數列》張貼在當日的預印本網站上，並向《美國數學年鑒》（*Annals of Mathematics*）投稿。

這是一項驚人的成就，他們的發現揭示了質數中存在的某種規律。他們的證明立即在國際學術界引起轟動。

3．其他已證明的質數定理

已證明的質數定理還包括以下幾項：

- 一個偶數可以寫成兩個數字之和，其中每一個數字最多只有 9 個質因數。
- 一個偶數必定可以寫成一個質數加上一個合數，其中的因數個數有上界。

- 一個偶數必定可以寫成一個質數加上一個最多由 5 個因數所組成的合數。後來，有人簡稱這個結果為（1+5）。
- 一個充分大偶數必定可以寫成一個質數加上一個最多由 2 個質因數所組成的合數，簡稱為（1+2）。

2.2　孿生質數

我們知道，質數在自然數中的比例很少，而孿生質數就更少了。那麼，什麼是孿生質數？孿生質數有什麼特點呢？

2.2.1　什麼是孿生質數

所謂孿生質數，是指間隔為 2 的相鄰質數，它們之間的距離已經近得不能再近了，就像孿生兄弟一樣，因此被稱為孿生質數，也稱為雙生質數。

例如，質數 3 和 5，其間距為 2，就是一組孿生質數。100 以內的孿生質數還有 5 與 7，11 與 13，17 與 19，29 與 31，41 與 43，59 與 61，71 與 73。

(3, 5)　(5, 7)　(11, 13)　(17, 19)
(29, 31)　(41, 43)　(59, 61)　(71, 73)

100 以內的孿生質數共有 8 對，不過，隨著數字的增大，孿生質數的分佈變得越來越稀疏，尋找孿生質數也變得越來越困難。那麼會不會在超過某個界限之後就再也不存在孿生質數了呢？

2.2.2　孿生質數的公式

對於自然數 Q 與 Q+2，若都不能被小於$\sqrt{Q+2}$ 的任何質數整除，則 Q 與 Q+2 就構成一對孿生質數。這句話可以用以下公式表達：

$$\begin{aligned} Q &= P_1M_1+B_1 \\ &= P_2M_2+B_2 \\ &= \cdots\cdots \\ &= P_kM_k+B_k \end{aligned}$$

以上公式中，P_1、P_2、P_k 表示從小到大的順序質數 2、3、5、7……，$B_i \neq 0$ 且 $B_i \neq P_i - 2$，若 $Q<(P_{K+1})^2 - 2$，則 Q 與 $Q+2$ 是一對孿生質數。也就是說，將數 Q 分解後，最小剩餘不能為 0 和 P_i-2，例如 Q 不能是 $2M$，$3M+1$，$5M+3$，$7M+5$……，$P_iM_i - 2$，否則 $Q+2$ 就是合數。

看一個例子吧。假設 $Q=29$，可分解為以下算式：

$$\begin{aligned} 29 &= 2\times 14+1 \quad (2M+1) \\ &= 3\times 9+2 \quad (3M+2) \\ &= 5\times 5+4 \quad (5M+4) \end{aligned}$$

由於 $\sqrt{29+2}$ 的結果取整後為 5，因此，在上式中只按公式分解到 5 為止，一共有 3 項（即 $k=3$），且每項的 B 值都不為 0，且 B_i 的每一項不等於 $P_i - 2$。因此，可得到 29 與 29+2 是一對孿生質數。

上式也可使用同餘式來表達：

$$\begin{aligned} Q \div 2 &= 14 \quad \text{餘 } 1 \\ Q \div 3 &= 9 \quad \text{餘 } 2 \\ Q \div 5 &= 5 \quad \text{餘 } 4 \end{aligned}$$

根據中國剩餘定理，對於給定的餘數，在除數為質數的範圍內有唯一的解。例如在上式中，除數為 2、3、5 時餘數也知道了，因此就可以推算出 Q 的值為 29。

2.2.3　中國剩餘定理

什麼是中國剩餘定理呢？先來看一則中國民間的傳說故事——「韓信點兵」。

秦朝末年，楚漢相爭。一次，韓信將 1500 名將士與楚王大將李鋒交戰。苦戰一場，楚軍不敵，敗退回營，漢軍也死傷四五百人，於是韓信整頓兵馬也返回大本營。當行至一山坡，忽有後軍來報，說有楚軍騎兵追來。只見遠方塵土飛揚，殺聲震天。漢軍本來已十分疲憊，這時隊伍大嘩。韓信兵馬到坡頂，見來敵不足五百騎，便急速點兵迎敵。他命令士兵 3 人一排，結果多出 2 名；接著命令士兵 5 人一排，結果多出 3 名；他又命令士兵 7 人一排，結果又多出 2 名。韓信馬上向將士們宣佈：我軍有 1073 名勇士，敵人不足 500，我們居高臨下，以眾擊寡，一定能打敗敵人。漢軍本來就信服自己的統帥，這一來更相信韓信是「神仙下凡」、「神機妙算」。於是士氣大振。一時間旌旗搖動，鼓聲喧天，漢軍步步緊逼，楚軍亂作一團。交戰不久，楚軍大敗而逃。

韓信是怎麼知道軍隊有 1073 名士兵的呢？

對於這個問題，可將其描述為一個數學問題，就是：一個數除以 3 餘 2，除以 5 餘 3，除以 7 餘 2，求這個數是多少？

先列出除以 3 餘 2 的數：

$$2, 5, 8, 11, 14, 17, 20, 23, 26, \ldots$$

再列出除以 5 餘 3 的數：

$$3, 8, 13, 18, 23, 28, \ldots$$

在這兩列數中，首先出現的公共數是 8，3 與 5 的最小公倍數是 15，兩個條件合併成一個就是：

$$8+15\times n$$

將 *n* 分別取 1、2、3、…即可得到數列：

$$8, 23, 38, \ldots$$

再列出除以 7 餘 2 的數：

$$2, 9, 16, 23, 30, \ldots$$

可以看出，符合題目條件的最小數是 23。

也就是說，我們已把題目中三個條件合併成一個：該數除以 105 餘 23。

由於漢軍原有士兵 1500 人，死傷四五百人，即剩餘的士兵應為 1000 餘人，即可得到士兵的總數：

$$105\times10+23=1073$$

2.2.4 孿生質數分佈情況

我們知道，質數本身的分佈是隨著數字的增大而越來越稀疏，不過幸運的是，早在古希臘時代，歐幾里德（Euclid）就證明了質數有無窮多個。長期以來人們猜測孿生質數也應該有無窮多組，可是該怎麼驗證這個猜想呢？

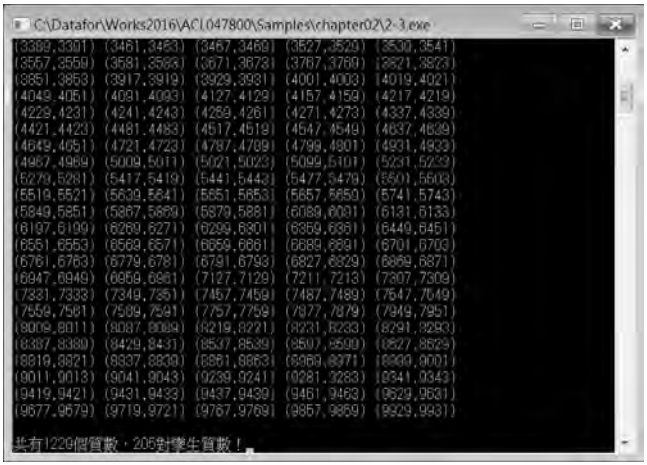
對於程式設計師來說，當然是想編寫程式來搜尋質數和孿生質數。當然，由於電腦位元數的限制，所表示的整數範圍是有限的，如果要找出更多的質數或孿生質數，需要另外編寫大整數處理的相關功能。

由於篇幅限制，本書將不介紹大整數方面的功能，下面只列出一個簡單的求解孿生質數的程式。在程式中，將先篩選出質數，然後在質數中再篩選出孿生質數。

```
#include <stdio.h>

#define MAXNUM 10000                          //求 10000 以內的所有質數
int main()
{
```

```
    long i,j,c=0,twin=2,t=0;
    char prime[MAXNUM+1];                    //定義保存質數的陣列
    prime[0]=0;
    prime[1]=0;
    for(i=2;i<=MAXNUM;i++)                   //初始化陣列
        prime[i]=1;                          //旗標為 1 表示對應的數是質數

    for(i=2;i*i<=MAXNUM;i++)                 //篩選合數
    {
        if(prime[i]==1)                      //若為質數
        {
            for(j=2*i;j<=MAXNUM;j++)         //篩去合數
            {
                if(!prime[j]) continue;
                if(j%i==0)                   //數 j 能被整除，說明不是質數
                    prime[j]=0;              //清除旗標
            }
        }
    }

    //統計質數數量
    for(i=2;i<MAXNUM;i++)
    {
        if(prime[i]==1)
        {
            c++;
            if(i-2==twin)                    //是孿生質數
            {
                printf("(%d,%d) ",twin,i);
                t++;
                if(t%5==0)  printf("\n");
            }
            twin=i;
        }
    }

    printf("\n 共有%d 個質數，%d 對孿生質數！",c,t);
    getch();
    return 0;
}
```

執行以上程式，將得到如下頁圖示的結果。從結果中可以看到，在 10000 以內共有 1229 個質數，而孿生質數只有 205 對。



在上面的程式中，常量 MAXNUM 定義為 10000，即可求出 10000 以內的質數，如果將 MAXNUM 定義為更大的常數值，則可求出更多的質數和孿生質數。當然要注意，由於電腦中變數表示範圍及程式堆疊空間大小的限制，MAXNUM 定義的資料大小是有限的。

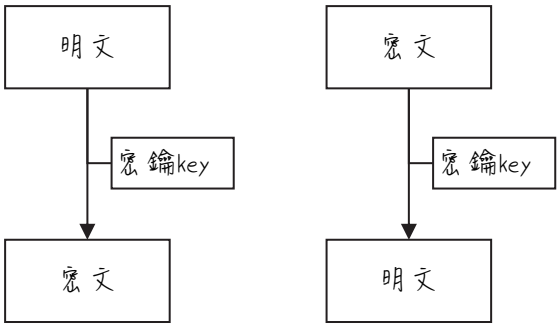
2.3 使用質數的 RSA 演算法

RSA 公開金鑰加密演算法是第一個既能用於資料加密也能用於數位簽章的演算法。它易於理解和操作，也十分流行。RSA 演算法就是質數的典型應用。下面簡單介紹一下 RSA 演算法，要完整地實現 RSA 演算法，需要較長的程式碼，本書就不介紹相對應的程式了，而是重點介紹 RSA 的概念、原理和實現的過程。

2.3.1 什麼是 RSA

在電腦中常用的加解密技術分為兩類，即對稱加密和非對稱加密。

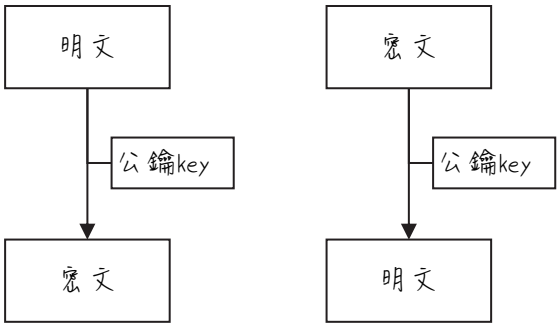
在對稱加密技術中，對資訊的加密和解密都使用相同的金鑰 Key，如下頁圖示，也就是說使用同一個金鑰 Key 對資料進行加密和解密。這種加密方法可簡化加解密的處理過程，資訊交換雙方都不必彼此研究和交換專用的加解密演算法。如果在交換階段，金鑰 Key 沒有洩露，那麼加密資料的機密性和訊息完整性就可以得到保證。



對稱加密技術雖然簡單，但存在一些不足，由於加密、解密都需要使用同一個 Key，這樣，資訊傳送雙方都要接觸到這個 Key，金鑰 Key 更容易洩露。

而非對稱加密（又稱為公開金鑰加密）中，不再只有一個金鑰 Key 了。在非對稱加密演算法中，金鑰被分解為一對，一個稱為公開金鑰（簡稱公開金鑰 PK），另一個稱為私有金鑰（簡稱私密金鑰 SK）。對於公開金鑰，可以透過非保密方式向他人公開，而私密金鑰則由解密方保存，不用對別人公開。

發送資訊的一方透過公開金鑰對資料進行加密，然後發送給接收方。接收方透過私密金鑰對密文進行解密，加、解密的過程如圖所示。



由於非對稱加密方式可以使通訊雙方無須事先交換金鑰就可以建立安全通訊，因此被廣泛應用於身份認證、數位簽章等資訊交換領域。

非對稱加密體系一般是建立在某些已知的數學難題之上，是電腦複雜性理論發展的必然結果。最具有代表性的非對稱加密方式是 RSA 公開金鑰密碼體制。