序

高階程式語言隨著電腦硬體的進步而不斷推陳出新，經過多年來的百家爭鳴，C 語言依然受到程式設計師的喜愛。由於程式語言隨著功能的提升、都朝物件導向程式設計發展，所以 C++就成為 C 語言的主流。

APCS「大學程式設計先修檢測」，目的在檢測考生程式設計的能力，讓高中職學生有一個學習成果具公信力的檢驗，並提供大專院校選才的參考依據。APCS 題目分為程式設計「觀念題」與「實作題」兩大科目，「觀念題」為選擇題 40 題滿分 100 分、「實作題」共有四題程式實作滿分 400 分。目前資訊相關科系都將 APCS 成績，列入多元入學的重要篩選項目中，且電機、機械、數學、人工智慧、數位科技、資訊教育、醫學資訊、資訊傳播與財務金融等科系都開始採計 APCS。通常會要求觀念、實作成績要達 2 級分以上，而頂尖大學則會要求至少 4 級分。甚至大學入學甄選委員會擴大舉辦 APCS 招生名額，是進入資訊類學系另一個管道。

本書將 C++以詳細的說明、具代表性的簡例、學以致用的範例，和豐富的習題提供練習，協助讀者打下程式設計的堅實基礎。並在各章節中，將歷屆相關的觀念題做詳盡的解題說明，期望讀者能融會貫通。最後將 105、106 年四次實作題，從解題的演算法說明，到程式碼的實際撰寫，做深入淺出的解說，幫助讀者能獨立思考，養成具有獨立程式設計上的能力，並能順利高分通過 APCS 檢測。

為方便教學，本書另提供教學投影片，歡迎採用本書的授課教師向碁峰業務索取。同時系列書籍於「程式享樂趣 YouTube」頻道每週五分享補充教材與新知，以利初學者快速上手。有關本書的任何問題可來信至 itPCBook@gmail.com，我們會盡快答覆。本書雖經多次精心校對，難免百密一疏，尚祈讀者先進不吝指正，以期再版時能更趨紮實。感謝周家旬與廖美昭小姐細心校稿與提供寶貴的意見，以及碁峰同仁的鼓勵與協助，使得本書得以順利出書。在此聲明，書中所提及相關產品名稱皆為各所屬公司之註冊商標。

程式享樂趣 YouTube 頻道：https://www.youtube.com/@happycodingfun

微軟最有價值專家、僑光科技大學多媒體與遊戲設計系 助理教授　蔡文龍
張志成、何嘉益、張力元 編著

CHAPTER

9 遞迴

9.1 遞迴

函式間可以相互呼叫，除了呼叫別的函式外，也可呼叫自己本身，這種函式呼叫自己的方式稱為「遞迴」。遞迴是一種應用極廣的程式設計技術，在函式執行的過程不斷的呼叫函式自身，但每一次呼叫，皆會產生不一樣的效果，直到遇到終止再呼叫函式自身的條件或效果時，才會停止遞迴離開函式。如果遞迴的函式內沒有設定終止呼叫的條件，這樣的函式會形成無窮遞迴。

一個問題如果能拆成同形式且較小範圍時，就可以使用遞迴函式來設計。例如要計算 1 + 2 + … + 10 的總和時，可以拆成 1 和 2 + 3 + … + 10，而 2 + 3 + … + 10 又可以拆成 2 和 3 + 4 + … + 10，其餘類推，此時就可以設計成遞迴函式。遞迴函式常使用在具有規則性的程式設計中，其優點是具結構化可以增加程式的可讀性，以及能以簡潔的程式處理反覆的複雜問題。

遞迴在數學或電玩遊戲上常被使用，例如：數列、階乘、費氏數列、輾轉相除法、排列、組合、堆疊、河內塔、八個皇后、老鼠走迷宮…等。有些程式雖然使用 for、while…等重複結構也能處理，但使用遞迴函式會較為簡潔易懂，本章將針對遞迴的基本範例做設計的說明。

9.2 數列

本節提供兩個數列函式求總和的範例，說明如何使用遞迴解題的方式來設計程式。如下：

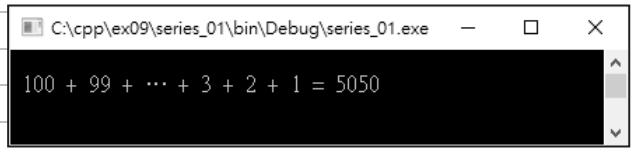
1. sum = n + (n-1) + ⋯ + 3 + 2 + 1
2. sum = 1 – 4 + 7 – 10 + 13 – ⋯ – (n-3) + n

範例 ：series_01.cpp

使用遞迴函式計算 n + (n-1) + (n-2) + … + 3 + 2 + 1 的結果。
其中 n = 100。

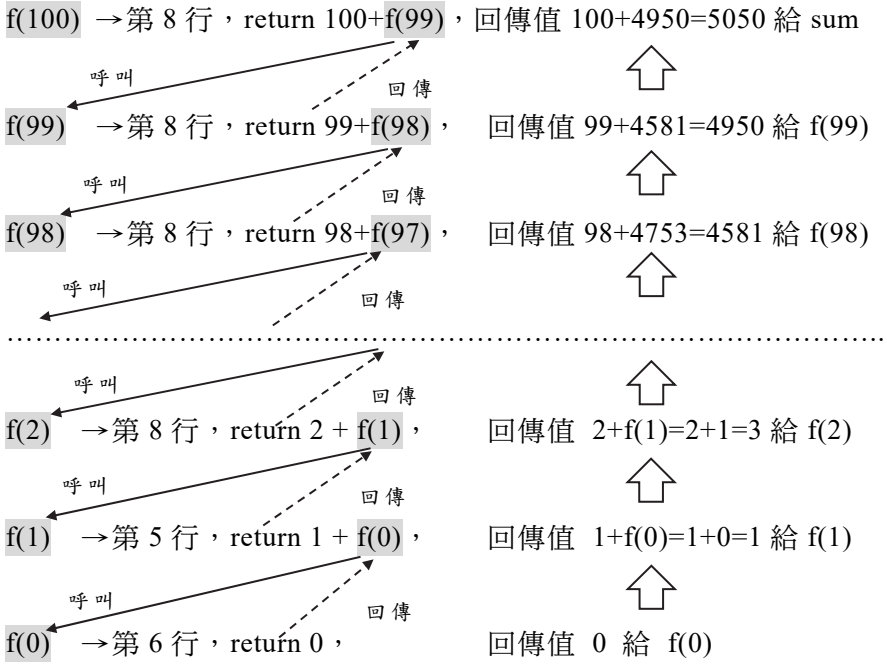
程式碼 FileName : series_01.cpp

```
01 #include <iostream>
02 using namespace std;
03
04 int f(int n) {
05     if (n <= 0)
06         return 0;
07     else                    // n > 0
08         return n + f(n-1);
09 }
10
11 int main()
12 {
13     int n = 100, sum;
14     sum = f(n);
15     cout << "\n " << n << " + " << n-1 << " + … + 3 + 2 + 1 = " << sum;
16     cout << "\n " ;
17     return 0;
18 }
```

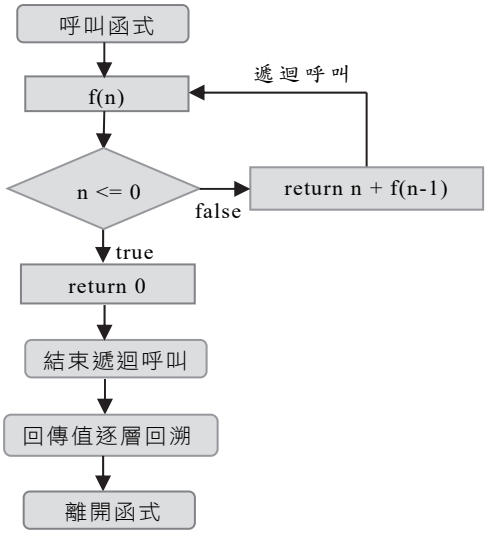


說明

1. 第 4~9 行：建立 f(n) 遞迴函式。該函式被呼叫 f(100) 的流程如下所示：



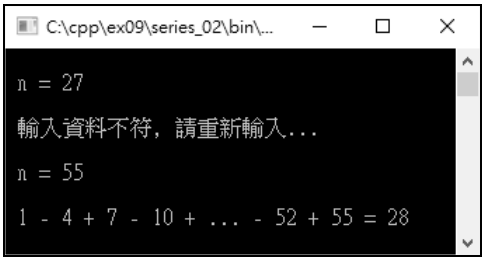
2. 第 14 行：將 f(100) 的回傳值 5050 指定給 sum 變數。
3. 遞迴函式一定要有結束遞迴的敘述。當第 5 行的條件式 (n<=0) 成立時，執行第 6 行 return 0，就不再遞迴呼叫，而將回傳值逐層回溯給原呼叫敘述。
4. 遞迴函式的流程圖如右：

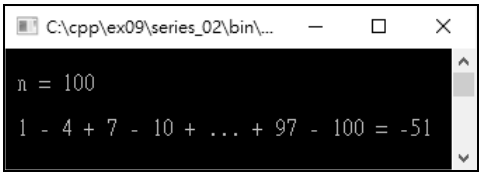


範例 ：series_02.cpp

使用遞迴函式計算 1 – 4 + 7 – 10 + 13 – … n 的結果，其中 n 由使用者輸入。n 的輸入值必須符合 (n % 3 == 1) 條件，即 n 為 3 的倍數加 1，如 1、4、7、10、14…等。

執行結果





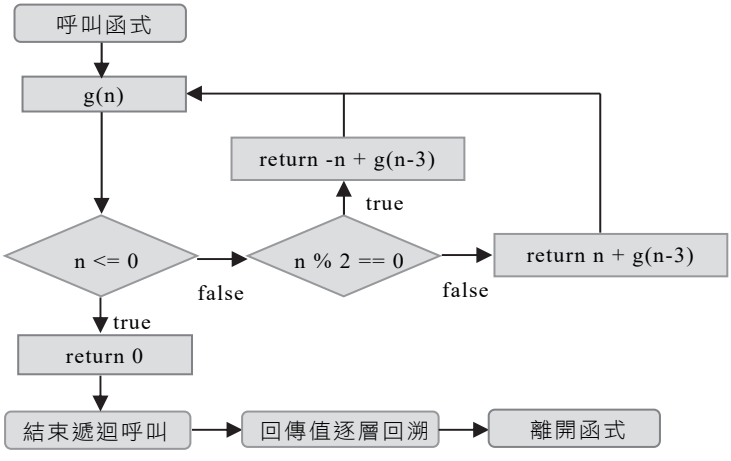
程式碼 FileName : series_02.cpp

```
01 #include <iostream>
02 using namespace std;
03
04 int g(int n) {
05    if (n <= 0) {
06       return 0;
07    } else if (n % 2 == 0) {      // n > 0,為偶數
08       return -n + g(n-3);
```

```
09    } else {                  // n > 0,為奇數
10       return n + g(n-3);
11    }
12 }
13
14 int main()
15 {
16    int n, sum;
17    while (true) {
18       cout << "\n n = ";
19       cin >> n;
20       if (n % 3 == 1)
21          break;             //輸入值符合條件跳離迴圈
22       else
23          cout << "\n 輸入資料不符, 請重新輸入...\n";
24    }
25
26    sum = g(n);
27    if (n % 2 == 1) {       // n輸入值為奇數
28       cout << "\n 1 - 4 + 7 - 10 + ... - " << n-3 << " + " << n << " = " << sum;
29    } else {                // n輸入值為偶數
30       cout << "\n 1 - 4 + 7 - 10 + ... + " << n-3 << " - " << n << " = " << sum;
31    }
32    cout << "\n\n";
33    return 0;
34 }
```

說明

1. 遞迴函式的流程圖如下所示：



2. 第 4~12 行：建立 g(n) 遞迴函式。該函式被呼叫 g(100) 的流程如下：

   g(100)

   → return -100 + g(97)

   → -100 + return 97 + g(94)

   → -100 + 97 + return -94 + g(91)

   ……

   → -100 + 97 – 94 + … -10 + return 7 + g(4)

   → -100 + 97 – 94 + … -10 + 7 + return -4 + g(1)

   → -100 + 97 – 94 + … -10 + 7 - 4 + return 1 + g(0)

   → -100 + 97 – 94 + … -10 + 7 - 4 + 1 + 0

   → -51 (回傳值)

3. 第 20~23 行：篩選使用者的輸入值是否符合 (n % 3 == 1) 的條件。
4. 第 26 行：呼叫 g(n) 的遞迴計算結果回傳指定給 sum 變數。
5. 第 27,28 行：若輸入值為奇數時，印出遞迴函式執行的加減過程，其中最後兩數是先減後加。
6. 第 29,30 行：若輸入值為偶數時，印出遞迴函式執行的加減過程，其中最後兩數是先加後減。

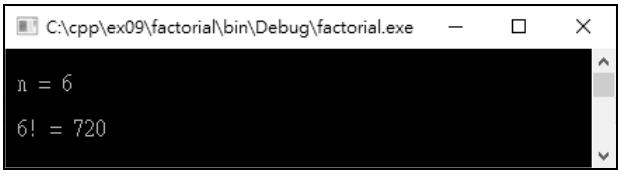
9.3 階乘

在數學中，正整數的「階乘」是所有小於及等於該數 n 的正整數的乘積，以 n! 表示。階乘的公式為 n! = n * (n-1) * (n-2) * … * 3 * 2 * 1，例如：
5! = 5 * 4 * 3 *2 * 1 = 120。

範例 ：factorial.cpp

使用階乘函式計算 n! = 1 * 2 * 3 * (n-1) * n 的結果，其中 n 由使用者輸入。
n 的輸入值必須大於等於 1。

執行結果



程式碼 FileName : factorial.cpp

```
01 #include <iostream>
02 using namespace std;
03
04 int d(int n){
05     if (n <= 1) {
06         return 1;
07     } else {                    // n>1
08         return n * d(n-1);
09     }
10 }
11
12 int main()
13 {
14     int n, fac;
15     while (true) {
16         cout << "\n n = ";
17         cin >> n;
18         if (n>=1)
19             break;
20         else
21             cout << "\n 輸入資料不符, 請重新輸入... \n";
22     }
23     fac = d(n);
24     cout << "\n " << n << "! = " << fac;
25
26     cout << "\n\n";
27     return 0;
28 }
```

說明

1. 第 4~10 行：建立 d(n) 遞迴函式。該函式被呼叫 d(6) 的流程如下所示：

   d(6)

   → return 6 * d(5)

   → return 6 * 5 * d(4)

   → return 6 * 5 * 4 * d(3)

   → return 6 * 5 * 4 * 3 * d(2)

   → return 6 * 5 * 4 * 3 * 2 * d(1)

   → return 6 * 5 * 4 * 3 * 2 * 1

   → return 720　(回傳值)

2. 第 18~21 行：篩選使用者的輸入值是否符合 (n >= 1) 的條件。

3. 第 23 行：呼叫 d(6) 的遞迴計算結果，回傳指定給 fac 變數。

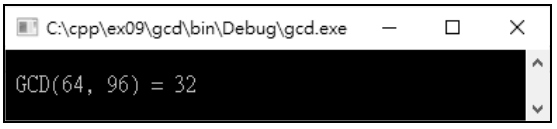
9.4 最大公因數

使用遞迴求兩整數 p、q 之最大公因數 GCD(p, q) 函式的設計，就是將兩數進行輾轉相除法的數學運算。所謂「輾轉相除法」就是將兩數相除，若能整除則除數為最大公因數；若不能整除，則除數變為被除數，餘數變為除數，再將兩數相除，… 以此類推。

範例 ：gcd.cpp

使用輾轉相除法的遞迴運算設計 GCD(p, q) 函式，求出兩整數 64、96 的最大公因數。

執行結果



程式碼 FileName : gcd.cpp

```
01 #include <iostream>
02 using namespace std;
03
04 int GCD(int p, int q) {
05     int rem;
06     if (p == 0 or q == 0) {
07         return 0;
08     } else {
09         rem = p % q;
10         if (rem == 0)
11             return q;
12         else
13             return GCD(q, rem);
14     }
15 }
16
17 int main()
18 {
19     int n1, n2, res;
20     n1 = 64;
21     n2 = 96;
22     res = GCD(n1, n2);
23     cout << "\n GCD(" << n1 << ", " << n2 << ") = " << res;
24
25     cout << "\n\n";
26     return 0;
27 }
```

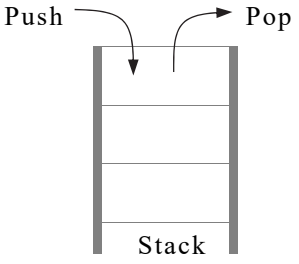
= [1 + 1 + C(1,0) + C(1,1)] + [1 + C(1,0) + C(1,1) + C(1,0) + C(1,1) + 1]
= [1 + 1 + 1 + 1] + [1 + 1 + 1 + 1 + 1 + 1]
= 10 (回傳值)

2. 第 19~22 行：篩選使用者的 n 和 m 的輸入值是否符合條件要求。
3. 第 24 行：呼叫 C(5, 2) 的遞迴計算結果，回傳指定給 ans 變數。

9.7 堆疊

「堆疊」(Stack) 是後進先出 (LIFO) 的概念。就像將一些物品依序放入有底的袋子，因為袋子的出入口只有一個，將物品一一拿出的順序就是後進先出。也像是一疊盤子，最後放上去的盤子會置於最上面，而要使用時，最後放上去的盤子最先取用。

參與堆疊的物品是屬於一個線性串列資料結構，它加入 (Push) 資料時會疊在串列的最上 (前) 端，而刪除 (Pop) 的資料也是從串列的最上 (前) 端開始移除。即最早存放的資料被擺在串列的最下或最末端，會是最晚被移除；最慢放入的資料，會是最先被移除。



範例 ：stack.cpp

陣列 arr(4) = {"AAA", "@@@", "$$$", "MMM"}，將其元素依序放入只有一個出入口的袋子，再一一取出。在放入和取出的過程中，觀察這些元素值出現在最上面的順序為何？

執行結果

程式碼 FileName : stack.cpp

```
01 #include <iostream>
02 using namespace std;
03
04 void stack(int m, string arr[]) {
05     static int n=0;
06     cout << "\n " << arr[n] << "  ";
07     if (n >= (m-1)) {
08         return ;
09     } else {
```

```
10        n = n + 1;
11        stack(m, arr);
12    }
13    n = n - 1;
14    cout << "\n " << arr[n] << "  ";
15 }
16
17 int main()
18 {
19    string arr[4] = {"AAA", "@@@", "$$$", "MMM"};
20    stack(4, arr);
21
22    cout << "\n\n";
23    return 0;
24 }
```

說明

1. 第 4 行：m = 4
2. 當 n = 0 時 →第 6 行顯示 arr[0] →第 10 行 n=n+1=1 →第 11 行遞迴呼叫
   當 n = 1 時 →第 6 行顯示 arr[1] →第 10 行 n=n+1=2 →第 11 行遞迴呼叫
   當 n = 2 時 →第 6 行顯示 arr[2] →第 10 行 n=n+1=3 →第 11 行遞迴呼叫
   當 n = 3 時 →第 6 行顯示 arr[3] →第 7 行 →第 8 行 return →回原呼叫行
   →第 13 行，n=n-1=3-1=2 →第 14 行顯示 arr[2] →回原呼叫行
   →第 13 行，n=n-1=2-1=1 →第 14 行顯示 arr[1] →回原呼叫行
   →第 13 行，n=n-1=1-1=0 →第 11 行顯示 arr[0] →回主程式原呼叫行

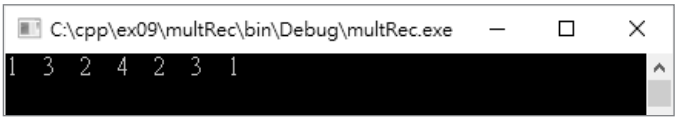
9.8 多遞迴

所謂「多遞迴」是指兩個以上的遞迴函式彼此呼叫。本節就以兩個函式 F1()、F2() 相互呼叫的例子來做說明。

範例 ：multRec.cpp

給定兩個函式 F1(m) 及 F2(n)，F1(m) 函式若 m<3 就結束；否則就以參數值 m+2 呼叫 F2() 函式。F2(n) 函式若 n<3 就結束；否則就以參數值 n-1 呼叫 F1() 函式。由主程式先呼叫 F1(1) 函式，接著再讓兩函式彼此呼叫，並列出呼叫時所顯示的資料。

執行結果



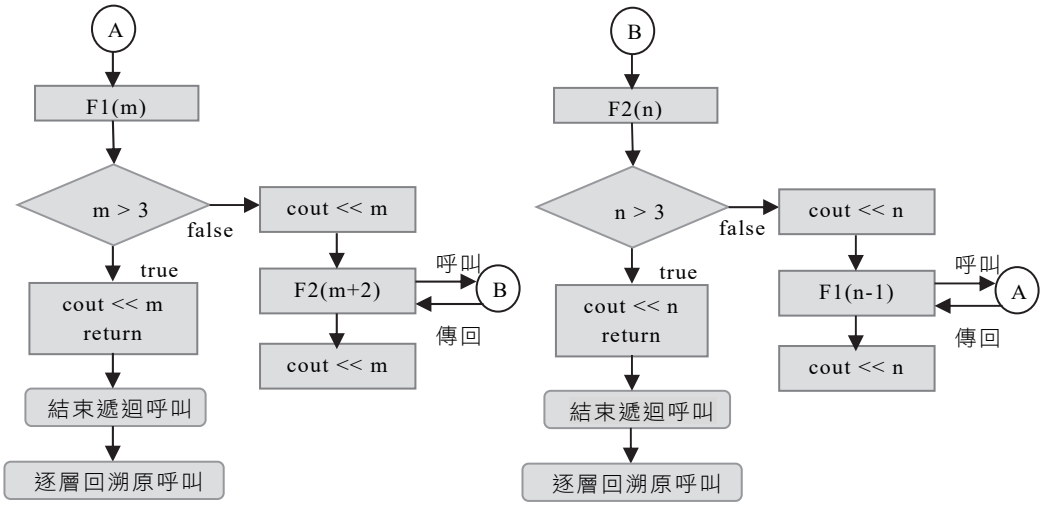
程式碼 FileName : multRec.cpp

```
01 #include <iostream>
02 using namespace std;
03
04 void F1(int);
05 void F2(int);
06
07 void F1(int m) {
08     if (m>3) {
09         cout << m << "  ";
10         return;
11     } else {
12         cout << m << "  ";
13         F2(m+2);
14         cout << m << "  ";
15     }
16 }
17
18 void F2(int n) {
19     if (n>3) {
20         cout << n << "  ";
21         return;
22     } else {
23         cout << n << "  ";
24         F1(n-1);
25         cout << n << "  ";
26     }
27 }
28
29 int main()
30 {
31     F1(1);
32     cout << "\n\n";
33     return 0;
34 }
```

說明

1. 第 31 行 F1(1)
   → m=1 → 第 12 行 顯示 1 → 第 13 行 F2(m+2) → F2(3)
   → n=3 → 第 23 行 顯示 3 → 第 24 行 F1(n-1) → F1(2)
   → m=2 → 第 12 行 顯示 2 → 第 13 行 F2(m+2) → F2(4)
   → n=4 → 第 20 行 顯示 4 → 第 21 行 return → 回原呼叫行
   → m=2 → 第 14 行 顯示 2 → 回原呼叫行
   → n=3 → 第 23 行 顯示 3 → 回原呼叫行
   → m=1 → 第 14 行 顯示 1 → 回主程式原呼叫行

2. 多遞迴函式的流程圖如下所示：



9.9 ACPS 觀念題攻略

題目 (一)

給定右側函式 f ( )，當執行 f(10) 時，最終回傳為何？

(A) 1　　(B) 3840　　(C) -3840

(D) 執行時導致無窮迴圈，不會停止執行

```
01 int f(int i) {
02    if (i>0)
03       if(((i/2)%2 == 0))
04          return f(i-2)*i;
05       else
06          return f(i-2)*(-i);
07    else
08       return 1;
09 }
```

說明

1. 答案是(C)，程式檔請參考 test09_1.cpp。
2. 當 i 為奇數時，會符合第 3 行 if(((i/2)%2 == 0)) 條件而執行第 4 行。
   當 i 為偶數時，不會符合第 3 行 if(((i/2)%2 == 0)) 條件而執行第 6 行。
3. f(10) = f(i-2)*(-i) = f(8)*(-10)　　// i = 10
   = (f(6)*(-8)) *(-10)　　// i = 8
   = (f(4)*(-6)) *(-8)*(-10)　　// i = 6
   = (f(2)*(-4)) *(-6)*(-8)*(-10)　　// i = 4
   = (f(0)*(-2)) *(-4)*(-6)*(-8)*(-10)　　// i = 2
   = (1* (-2)) *(-4)*(-6)*(-8)*(-10)　　// i = 0, 第 8 行
   = -3840

題 目 (四)

請問以 a(13,15) 呼叫右側 a()函式，函式執行完後其回傳值為何？

(A) 90　　(B) 103

(C) 93　　(D) 60

```
01 int a(int n, int m) {
02   if (n < 10) {
03     if (m < 10) {
04       return n + m ;
05     }
06     else {
07       return a(n, m-2) + m ;
08     }
09   }
10   else {
11     return a(n-1, m) + n ;
12   }
13 }
```

說明

1. 答案是(B)，程式檔請參考 test09_4.cpp。
2. a(13,15) = a(12,15)+13　　// 第 11 行
   = a(11,15)+12+13 = a(11,15)+25　　// 第 11 行
   = a(10,15)+11+25 = a(10,15)+36　　// 第 11 行
   = a(9,15)+10+36 = a(9,15)+46　　// 第 11 行
   = a(9,13)+15+46 = a(9,13)+61　　// 第 7 行
   = a(9,11)+13+61 = a(9,11)+74　　// 第 7 行
   = a(9,9)+11+74 = a(9,9)+85　　// 第 7 行
   = (9 + 9) +85 = 103　　// 第 4 行

題 目 (五)

給定 g() 函式如下，則 g(13) 的傳回值為何？

(A) 16　　(B) 18

(C) 19　　(D) 22

```
01 int g(int a) {
02    if (a>1) {
03       return g(a-2)+3;
04    }
05    return a;
06 }
```

說明

1. 答案是(C)，程式檔請參考 test09_5.cpp。
2. g(13) = g(11)+3 = g(9)+3+3 = g(7)+3+6
   = g(5)+3+9 = g(3)+3+12 = g(1)+3+15
   = 1+18 = 19

題目 (二十一)

若以 B(5,2) 呼叫右側 B() 函式，總共會印出幾次 "base case"？

(A) 1 (B) 5

(C) 10 (D) 19

```
int B (int n, int k){
    if (k == 0 || k == n){
        printf("base case\n");
        return 1;
    }
    return B(n-1,k-1)+B(n-1,k);
}
```

說明

1. 答案是(C)。
2. B(5,2) = B(4,1)+B(4,2) = B(3,0)+B(3,1) + B(3,1)+B(3,2) = 1 + 2*B(3,1) + B(3,2) 次
   B(3,1) = B(2,0)+B(2,1) = 1 + B(1,0) + B(1,1) = 3 次
   B(3,2) = B(2,1)+B(2,2) = B(1,0) + B(1,1) + 1 = 3 次
   所以 B(5,2) = 1 + 2*B(3,1) + B(3,2) 次 = 1 + 2*3 + 3 次 = 10 次

題目 (二十二)

若以 G(100) 呼叫右側函式後，n 的值為何？

(A) 25 (B) 75

(C) 150 (D) 250

```
01 int n = 0;
02 void K (int b) {
03     n = n + 1;
04     if (b % 4)
05         K(b+1);
06 }
07 void G (int m) {
08     for (int i=0; i<m; i=i+1) {
09         K(i);
10     }
11 }
```

說明

1. 答案是(D)。
2. 呼叫 G(100) 執行第 7 行，會依序呼叫 K(0) ~ K(99)。
3. 參數 b 以 0~99 呼叫 K(b) 時，b % 4 = r
   若 r = 0，即 b 為 4 的倍數，則以 b 為參數呼叫 K() 函式 1 次，n 累加 1。
   若 r = 1，則分別以 b, b+1, b+2, b+3 為參數呼叫 K() 函式 4 次，n 累加 4。
   若 r = 2，則分別以 b, b+1, b+2 為參數呼叫 K() 函式 3 次，n 累加 3。
   若 r = 3，則分別以 b, b+1 為參數呼叫 K() 函式 2 次，n 累加 2。
4. b=0~99 呼叫 K(b)，用 4 取餘數相除的過程有 25 個循環，每一次 n 共累加 10，故共累加 25*10 = 250。

CHAPTER

16

APCS 106 年 10 月實作題解析

16.1 邏輯運算子

問題描述

小蘇最近在學三種邏輯運算子 AND、OR 和 XOR。這三種運算子都是二元運算子，也就是說在運算時需要兩個運算元，例如 a AND b。對於整數 a 與 b，以下三個二元運算子的運算結果定義如下列三個表格：

a AND b

	b為0	b不為0
a為0	**0**	**0**
a不為0	**0**	**1**

a OR b

	b為0	b不為0
a為0	**0**	**1**
a不為0	**1**	**1**

a XOR b

	b為0	b不為0
a為0	**0**	**1**
a不為0	**1**	**0**

舉例來說：

(1) 0 AND 0 的結果為 0，0 OR 0 以及 0 XOR 0 的結果也為 0。

(2) 0 AND 3 的結果為 0，0 OR 3 以及 0 XOR 3 的結果則為 1。

(3) 4 AND 9 的結果為 1，4 OR 9 的結果也為 1，但 4 XOR 9 的結果為 0。

請撰寫一個程式，讀入 a、b 以及邏輯運算的結果，輸出可能的邏輯運算為何。

輸入格式

輸入只有一行，共三個整數值，整數間以一個空白隔開。第一個整數代表 a，第二個整數代表 b，這兩數均為非負的整數。第三個整數代表邏輯運算的結果，只會是 0 或 1。

輸出格式

輸出可能得到指定結果的運算，若有多個，輸出順序為 AND、OR、XOR，每個可能的運算單獨輸出一行，每行結尾皆有換行。若不可能得到指定結果，輸出 IMPOSSIBLE。(注意輸出時所有英文字母均為大寫字母。)

範例一：輸入	範例二：輸入
0 0 0	1 1 1
範例一：正確輸出	**範例二：正確輸出**
AND OR XOR	AND OR

範例三：輸入	範例四：輸入
3 0 1	0 0 1
範例三：正確輸出	**範例四：正確輸出**
OR XOR	IMPOSSIBLE

評分說明

輸入包含若干筆測試資料，每一筆測試資料的執行時間限制 (time limit) 均為 1 秒，依正確通過測資筆數給分。其中：

第 1 子題組 80 分，a 和 b 的值只會是 0 或 1。

第 2 子題組 20 分，0≤a，b<10,000。

解題分析

1. 本題使用 & (AND、且)、| (OR、或) 和 ^ (XOR、互斥) 位元運算子，做法是先將運算元轉成二進制，然後根據位元運算子的運算規則做運算。下表為 AND、OR、XOR 位元運算子的運算結果：

a	b	a & b (AND)	a | b (OR)	a ^ b (XOR)
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

2. 宣告 a、b、c 三個整數變數，再使用 cin 取得鍵盤輸入的三個整數置入給 a、b、c 三個整數變數。

3. 因位元運算子是對 0 和 1 做運算，但如下題目舉例發現可進行大於 1 的值的運算，例如：0 AND 3 或 4 OR 9 …等。

   ① 0 AND 0 的結果為 0，0 OR 0 以及 0 XOR 0 的結果也為 0。

② 0 AND 3 的結果為 0，0 OR 3 以及 0 XOR 3 的結果則為 1。

③ 4 AND 9 的結果為 1，4 OR 9 的結果也為 1，但 4 XOR 9 的結果為 0。

為簡化程式可使用 if 選擇敘述判斷 a 或 b 兩整數變數是否大於 0，若大於 0 則另該變數值為 1。寫法如下：

```
if(a>0) a=1;     //a 大於 0 時,令 a 為 1
if(b>0) b=1;     //b 大於 0 時,令 b 為 1
```

4. 宣告 and_result、or_result、xor_result 三個整數變數用來存放 AND、OR、XOR 位元運算的結果，若結果為 1 表示該位元運算成立。

   再依序判斷 a、b 進行 AND、OR、XOR 位元運算的結果是否等於 c，若成立將對應的 and_result、or_result、xor_result 的值設為 1。寫法如下：

```
int and_result=0, or_result=0, xor_result=0;

// 依序判斷 a 和 b 進行 AND、OR、XOR 位元運算的結果是否等於 c
if((a & b) == c) and_result = 1;
if((a | b) == c) or_result = 1;
if((a ^ b) == c) xor_result = 1;
```

5. 依序判斷 and_result、or_result、xor_result 的值是否為 1，若成立即印出該位元運算子的英文字，輸出順序為 AND、OR、XOR，每行結尾皆有換行。寫法如下：

```
if(and_result == 1) cout << "AND\n";
if(or_result == 1) cout << "OR\n";
if(xor_result == 1) cout << "XOR\n";
```

6. 當 and_result、or_result、xor_result 三個變數值皆為 0 時，即印出 "IMPOSSIBLE"。寫法如下：

```
if(and_result==0 && or_result==0 && xor_result==0)
    cout << "IMPOSSIBLE\n";
```

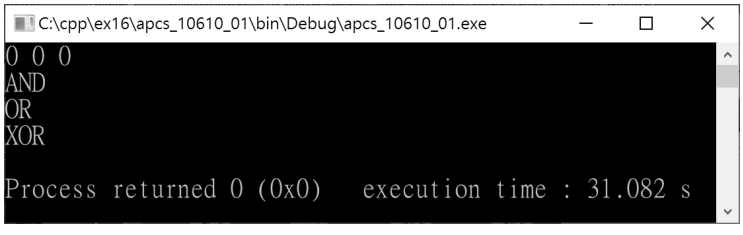
程式碼 FileName : apcs_10610_01.cpp

```
01 #include <iostream>
02 using namespace std;
03
04 int main()
05 {
06     // 宣告 a, b, c 三個整數變數
07     int a, b, c;
08     // 輸入三個整數置入 a, b, c 整數變數
09     cin >> a >> b >> c;
10
```

```
11      if(a>0) a=1;     //a 大於 0 時,令 a 為 1
12      if(b>0) b=1;     //b 大於 0 時,令 b 為 1
13
14      // 宣告 and_result, or_result, xor_result 三個變數
15      // 用來存放 AND、OR、XOR 位元運算子的運算結果
16      // 若 and_result, or_result, xor_result 的值為 1，表示該運算子成立
17      int and_result=0, or_result=0, xor_result=0;
18
19      // 依序判斷 a 和 b 進行 AND、OR、XOR 位元運算的結果是否等於 c
20      if((a & b) == c) and_result = 1;
21      if((a | b) == c) or_result = 1;
22      if((a ^ b) == c) xor_result = 1;
23
24      // 依序判斷 and_result, or_result, xor_result 的值是否為 1
25      // 若成立即印出該運算子的英文字
26      if(and_result == 1) cout << "AND\n";
27      if(or_result == 1) cout << "OR\n";
28      if(xor_result == 1) cout << "XOR\n";
29
30      // 當 and_result, or_result, xor_result 的值為 0 時即印出 IMPOSSIBLE
31      if(and_result==0 && or_result==0 && xor_result==0)
32          cout << "IMPOSSIBLE\n";
33      return 0;
34  }
```

執行結果

範例一：輸入 0△0△ 0 的執行結果。



範例二：輸入 1△1△1 的執行結果。