第七章

函式與巨集

前言：

- 在本章中，我們將介紹結構化程式設計的重點－『函式』。
- 除了函式之外，C語言還提供了具有類似函式功能的「巨集」，但巨集則會在編譯時期之前就先被處理完畢，因此在執行檔中，將不會出現巨集的程式碼。

大綱

- 7.1認識函式
  - 7.1.1　什麼是函式（function）
  - 7.1.2　函式的優點與特性
  - 7.1.3　呼叫函式的執行流程
- 7.2函式的宣告與定義
  - 7.2.1　函式宣告
  - 7.2.2　函式定義

大綱

- 7.3函式的使用
  - 7.3.1　函式呼叫
  - 7.3.2　函式的位置
  - 7.3.3　return敘述
- 7.4好用的亂數函式

大綱

- 7.5引數串列與引數傳遞
  - 7.5.1　傳值呼叫（Pass by value）
  - 7.5.2　傳指標呼叫（Pass by pointer）
  - 7.5.3　傳參考呼叫（Pass by reference）【C++補充】
  - 7.5.4　傳遞陣列
  - 7.5.5　搜尋演算法【補充】
  - 7.5.6　引數預設初值【C++補充】
  - 7.5.7　main函式的引數串列與回傳值

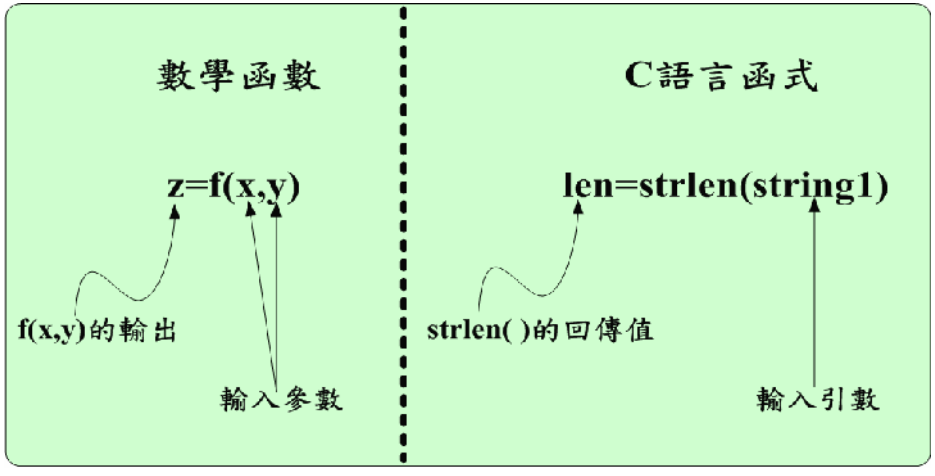
大綱

- 7.6自訂函式庫之引入標頭檔（#include）
- 7.7遞迴函式
- 7.8巨集
  - 7.8.1　前置處理(Preprocess)
  - 7.8.2　巨集指令：#define
  - 7.8.3　巨集、函式、常數的差別
- 7.9本章回顧

7.1 認識函式

7.1.1 什麼是函式（function）

- 在數學函數中，我們輸入函數的參數並經過函數處理後，將可以得到函數的輸出結果。
- 在C語言的函式中，同樣地也是如此，我們必須輸入函式的引數（Argument），經過函式的處理之後，可以獲得一個輸出結果（即函式回傳值），例如：strlen( ) 就是ANSI C定義用來計算字串長度的函式。
- 兩者的比較如下圖示意。



數學函數與C語言函式比較圖

7.1.1 什麼是函式（function）

- C語言的函式與數學函數類似，在數學函數中，我們會規定該數學函數的定義域範圍，例如：x,y為任意正數，
- 而C語言函式也必須限制輸入引數的資料型態，例如：string1必須為字串。
- 不一樣的地方
  - 是在C語言函式回傳值的資料型態方面，我們必須在函式宣告時加以定義回傳值的資料型別，
  - C語言也允許函式沒有回傳值。

7.1.2 函式的優點與特性

- C語言函式的特點整理如下：
  - 1. 函式是模組化的一大特色，將一個大的應用程式切割為數個副程式（即函式），就可以由許多的程式設計師分工撰寫各個副程式。
  - 2. 函式屬於應用程式的一部份，除了main函式之外，函式無法單獨執行。
  - 3. 類似於變數名稱，函式也擁有屬於自已的名稱，正常狀況下，不允許宣告兩個相同名稱的函式。（在C++中，則可以透過namespace或overload改善此一規定）。

7.1.2 函式的優點與特性

- 4. 函式內的變數，除非經過特別宣告，否則一律為『區域變數』，換句話說，在不同函式內可以使用相同的變數名稱，因為該變數只會在該函式中生效。

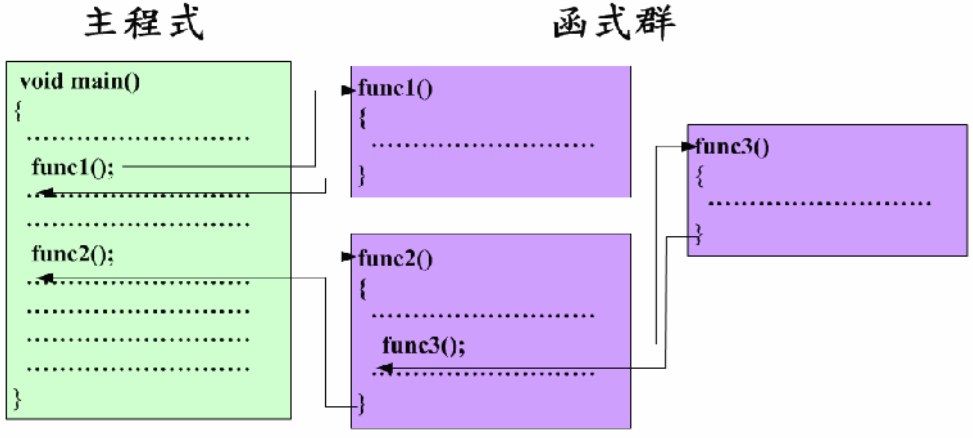
- 5. 函式最好具有特定功能，並且函式的程式碼應該越簡單越好，如此才能夠提高程式的可讀性並有利於除錯與日後的維護。

7.1.3 呼叫函式的執行流程

- 我們可以直接使用別人已經撰寫好的函式，只要在引入標頭檔時引入包含該函式的函式庫即可。例如：我們只要先引入ANSI C提供的string.h函式庫，就可以直接使用strlen( )函式，而不必自行撰寫strlen( )函式。

7.1.3 呼叫函式的執行流程

- 當主程式呼叫函式時，程式的控制權將會轉移到相對應的函式開頭處，然後執行函式中的程式碼，函式的程式碼執行完畢後，程式控制權將重新回到主程式碼（呼叫敘述）的下一個敘述，繼續往下執行。。



函式呼叫與返回示意圖

7.1.3 呼叫函式的執行流程

- 【觀念範例7-1】：藉由範例說明函式呼叫與返回的程式流程控制權之轉移。
- 範例7-1：ch7_01.c

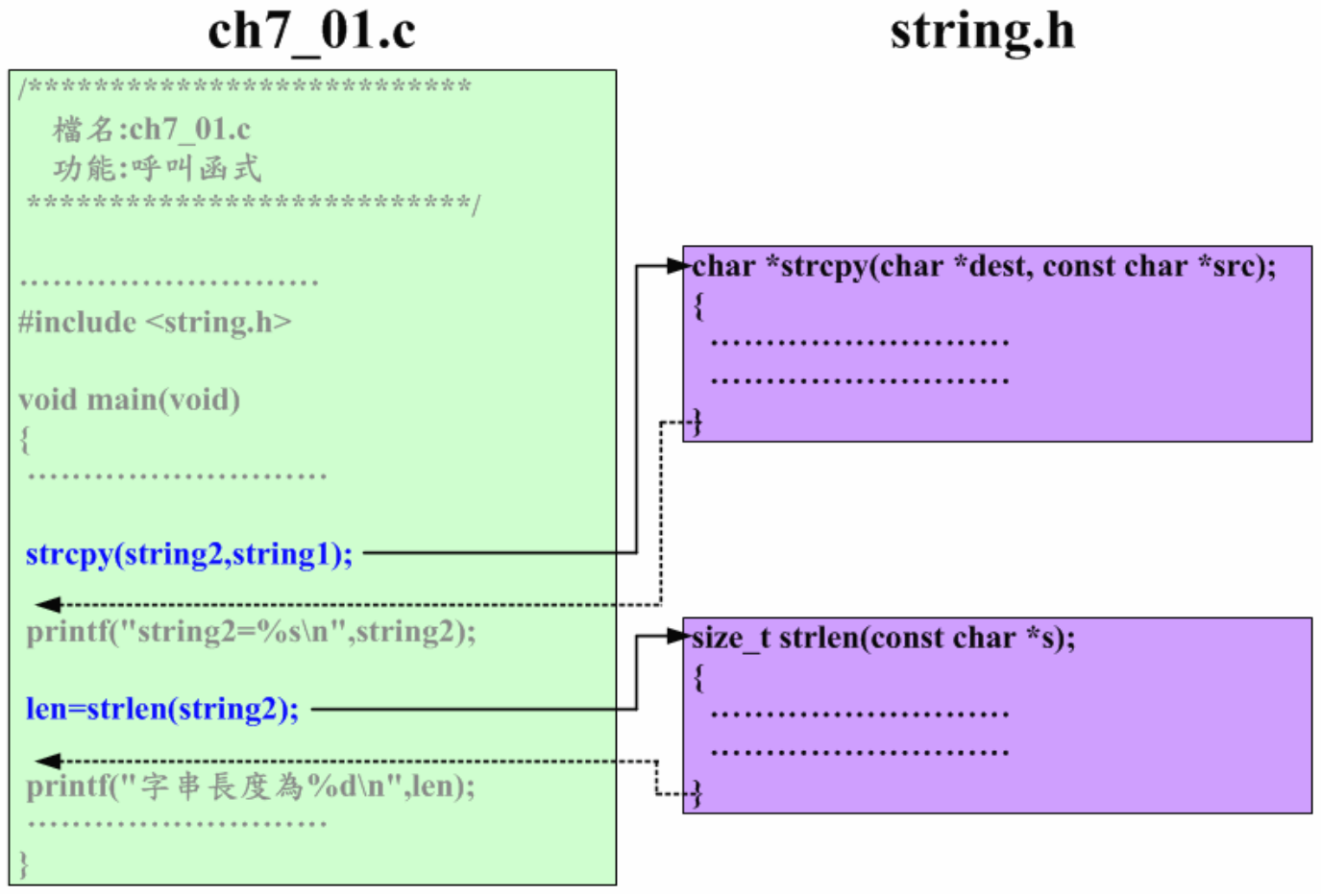
```
1  /*****************************
2      檔名:ch7_01.c
3      功能:呼叫函式
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8   void main(void)
9  {
10  char string1[60]="Welcome",string2[60];
11  int len;
12  strcpy(string2,string1);
13  printf("string2=%s\n",string2);
14  len=strlen(string2);
15  printf("字串長度為%d\n",len);
16  /*  system("pause");  */
17 }
```

- 執行結果：

```
string2=Welcome
字串長度為7
```

7.1.3 呼叫函式的執行流程

- 範例說明：
  - 整個程式的流程如圖所示



7.2 函式的宣告與定義

- 在使用任何一個函式之前，必須先宣告函式。
- 7.2.1 函式宣告
  - 在使用函式之前，我們必須先宣告函式，如此編譯器才能知道程式中具有該函式。
  - 函式的宣告語法如下：

```
函式回傳值型態 函式名稱(資料型態 [引數1],資料型態 [引數2],……);
```

  - 【語法說明】：
    - 函式可以有一個回傳值，而函式名稱前面的資料型態就是代表該回傳值的資料型態。
    - 若函式沒有回傳值（省略函式回傳值型態），在ANSI C中則規定必須宣告為void，來代表該函式無回傳值。
    - 2. 函式名稱小括號內則是引數群，每宣告一個輸入引數，都必須清楚地宣告該引數的資料型態，以及該輸入引數在函式中所代表的變數名稱。

7.2.1 函式宣告

- 引數的命名規則與一般變數的命名規則相同。
- 函式宣告必須出現在第一次呼叫函式之前。
- 函式宣告時，引數名稱可以省略，但引數的資料型態在函式宣告中則不可以省略。
- 合法的函式宣告範例。

宣告函式範例	解說
void func1( );	func1函式無回傳值，也不必輸入引數。
float func2(int a);	func2函式的回傳值為float資料型態，並且有一個整數型態的輸入引數。
int func3(int a,char b);	func3函式的回傳值為int資料型態，並且有兩個輸入引數，分別是整數型態的a，和字元型態的b。
int func4(int,char);	同func3，省略宣告引數名稱，引數資料型態不可省略。

7.2.2 函式定義

- 函式經過宣告後，代表編譯器得知該程式中存在一個這樣的函式，但只有函式宣告是不夠的，我們必須再定義函式的內容。
- 定義（實做）函式內容的語法如下：

```
函式回傳值型態        函式名稱(資料型態 引數1,資料型態 引數2,……)
{
   ……函式主體（程式碼）……
   [return … ;]
}
```

- 【語法說明】：
  - 函式定義的標頭和函式宣告差不多（但最後沒有分號『;』。
  - 不可省略引數名稱，函式定義（實做）還必須使用{ }包裝函式主體內容。

7.2.2 函式定義

- 輸入引數在函式主體內屬於合法的資料變數，也就是可以直接將這些輸入引數當作已宣告的變數使用。
- 具有回傳值的函式，在函式主體內應該包含一個return敘述，以便傳回資料。
- 不具回傳值的函式則可以沒有return敘述。
- 合法的函式定義範例。

函式定義範例	解說

void ShowWelcome(int print_times)
{
int a;
for(a=1;a<=print_times;a++)
printf("您好,歡迎光臨\n");
return;//可有可無
}
	(1)您可以在函式本體內使用輸入引數print_times。
	(2)函式無回傳值。

7.2.2 函式定義

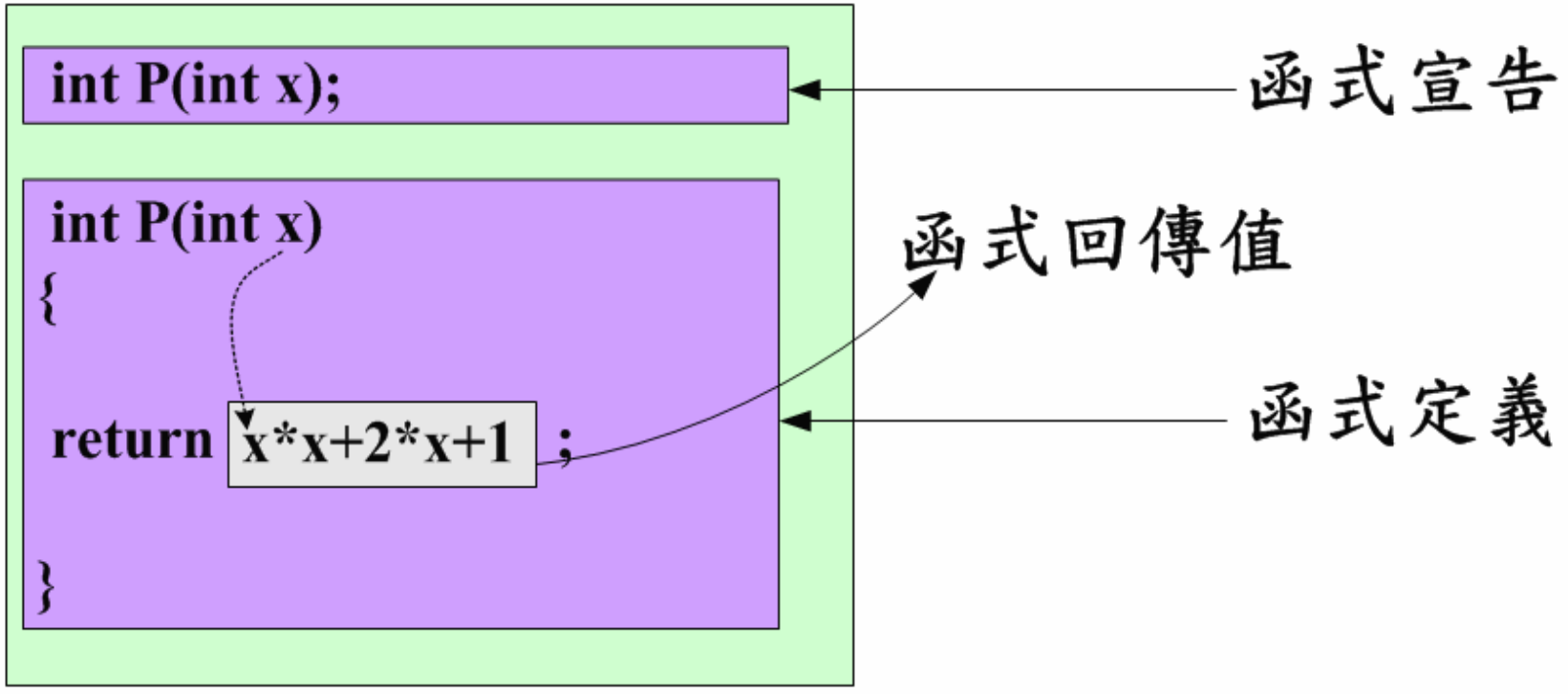
函式定義範例	解說

int Mul(int a,int b)
{
int result;
result = a*b;
return result;
}
	(1)您可以在函式本體內使用輸入引數a,b。
	(2)函式回傳值的資料型態為int。
	(3)使用return回傳資料，result為Mul函式的回傳值。
	(4)return敘述執行完畢，控制權將立刻返回原呼叫函式的下一個敘述。

double Add(double a,double b)
{
return (a+b);
}
	(1)您可以在函式本體內使用輸入引數a,b。
	(2)函式回傳值的資料型態為double。
	(3)使用return回傳資料，(a+b)運算式的結果為Add函式的回傳值。
	(4)執行完畢return敘述，控制權將立刻返回原呼叫函式的下一個敘述。

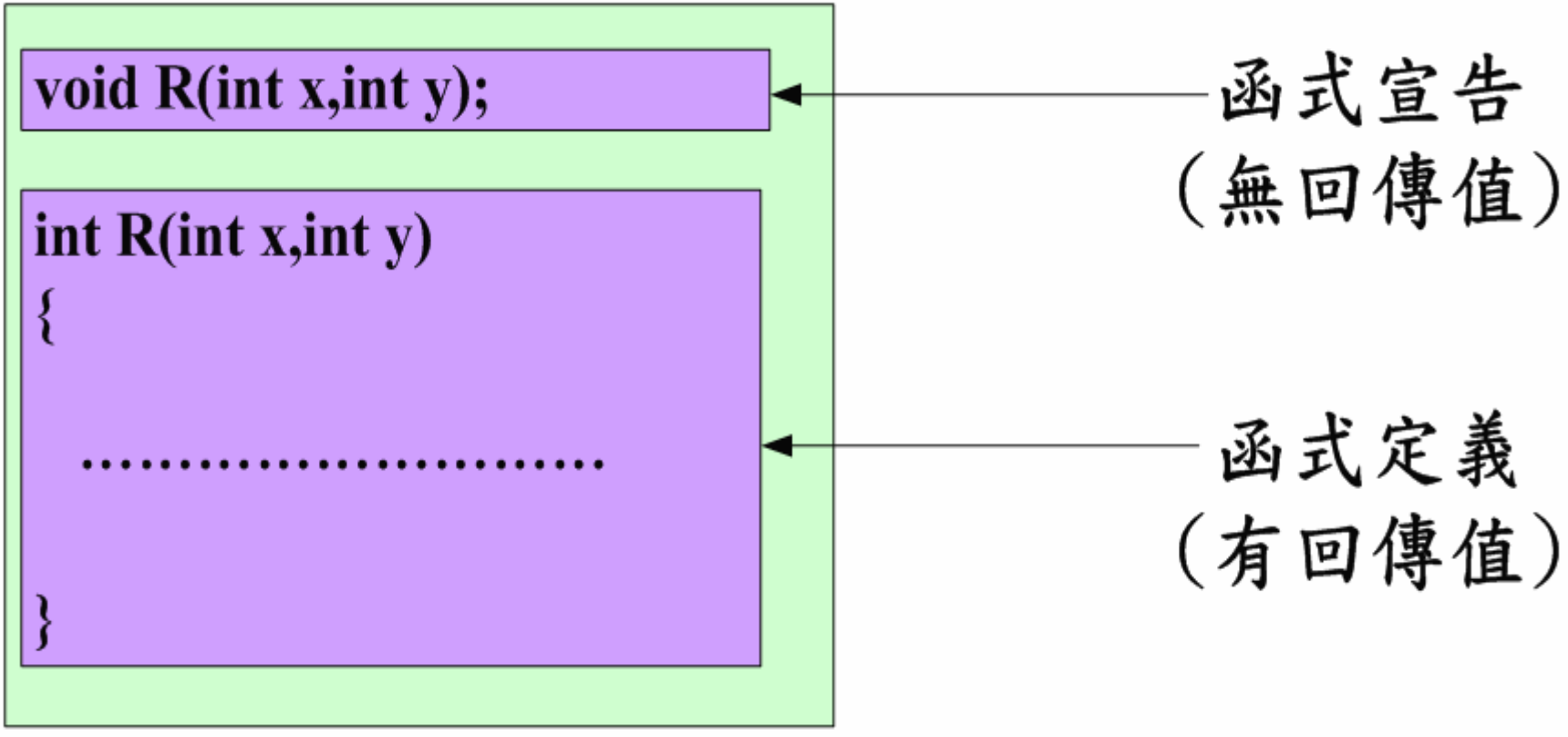
7.2.2 函式定義

- 【合法的函式宣告與定義範例】
  - (1)函式P可計算x^2+2x+1，其中x屬於整數。其函式宣告及定義如右：



7.2.2 函式定義

- 不合法的函式宣告與定義範例



7.3 函式的使用

- 在瞭解了C語言的函式宣告與定義後，在本節中，我們將實際設計合適的函式，並且呼叫這些函式來完成某些特定工作，並釐清函式呼叫的原理與相關規定。
- 7.3.1 函式呼叫
  - 函式經由宣告及定義後，必須透過函式呼叫（function call）才能實際應用該函式。
  - 函式呼叫可以視為一種轉移控制權的敘述。當程式執行過程中，遇到函式呼叫時，控制權將被轉移到被呼叫函式的起始點。
  - 當這些程式碼被執行完畢後，將會把控制權再交還給原來發生函式呼叫的程式執行點，執行下一個敘述。

7.3.1 函式呼叫

- 在C語言中，呼叫函式的語法如下：
  - 語法1（函式無回傳值）：

```
函式名稱(傳入引數串列);
```

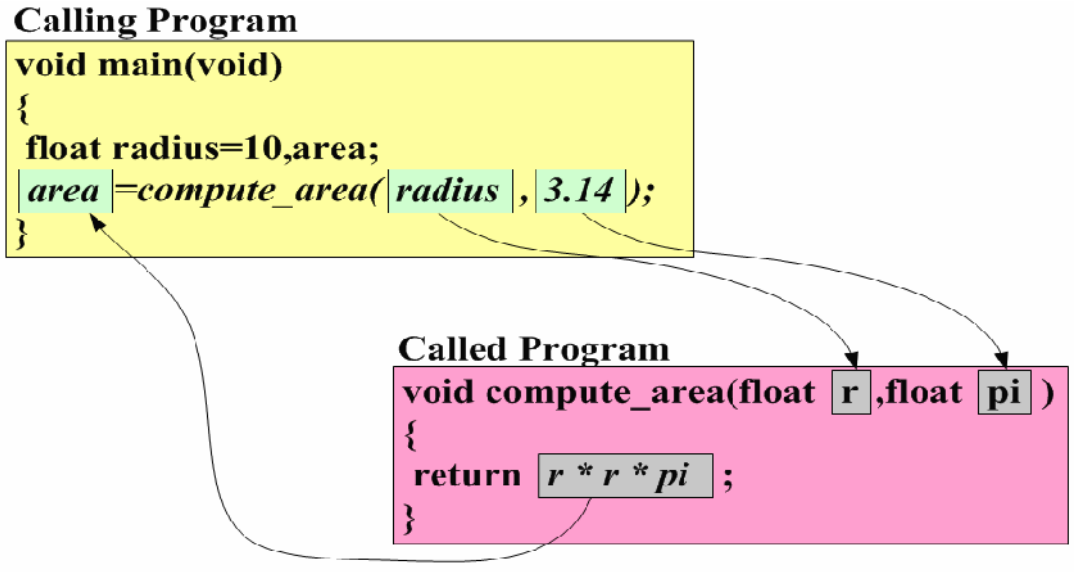
  - 語法2（函式有回傳值）：

```
變數=函式名稱(傳入引數串列);
```

  - 【語法說明】：
    - 呼叫敘述與被呼叫函式間若無資料需要傳遞，則只需要使用『函式名稱();』來呼叫函式即可，否則，必須要一一對應輸入所需要的引數。
    - 若函式有回傳值，則可以使用一個相容資料型態的變數來接收這個函式回傳值，也可以不接收。

7.3.1 函式呼叫

- 函式呼叫敘述必須與函式名稱相同，但兩者之引數名稱可以不同。
- 若呼叫者（Calling Program）有資料要傳遞給被呼叫者（Called Program），則必須藉由傳入引數串列將資料傳遞給函式，並且『傳入引數串列』的傳入變數會由『函式定義的引數串列』，如下圖示意：



實引數與虛引數的對應圖

7.3.1 函式呼叫

- 【觀念範例7-2】：製作一個函式（Power函式），功能為計算X^n。（X為實數、n為整數）。
- 範例7-2：ch7_02.c

```
1  /*****************************
2      檔名:ch7_02.c
3      功能:宣告,定義,呼叫函式
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  double Power(double,int);
8  double Power(double X,int n)
9  {
10  int i;
11  double PowerXn=1;
12  for(i=1;i<=n;i++)
13      PowerXn=PowerXn*X;
14  return PowerXn;
15 }
```

```
16
17 void main(void)
18 {
19  int k;
20  double Ans;
21  printf("計算3.5的k次方?請輸入k=");
22  scanf("%d",&k);
23  Ans=Power(3.5,k);/*  呼叫函式  */
24  printf("3.5的%d次方=%f\n",k,Ans);
 }
```

7.3.1 函式呼叫

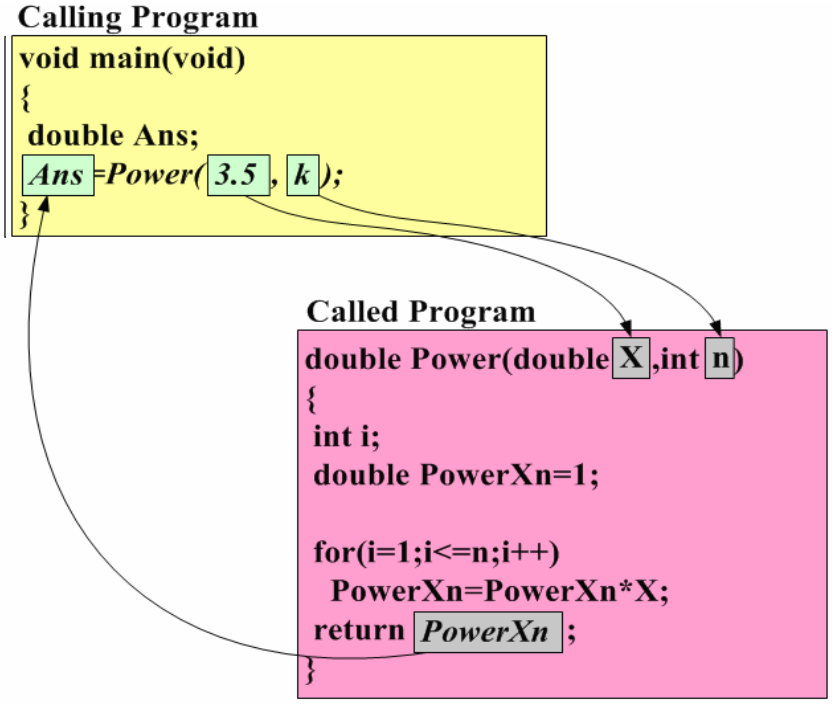
- 執行結果：

```
計算3.5的k次方?請輸入k=5
3.5的5次方=525.218750
```

- 範例說明：
  - 第9行：宣告函式Power，將函式宣告放在最前面，使得編譯器得知程式中含有Power函式。回傳值的資料型態是double，接受兩個傳入引數，資料型態分別是double,int。
  - 第11~19行：Power函式的定義，用來計算X^n。
  - 第28行：呼叫Power函式，傳入的引數為3.5,k，與函式定義的引數名稱不相同，其實這並不重要，只要傳入符合資料型態的數值或變數即可。
  - 使用Ans變數來存放函式回傳值。

7.3.1 函式呼叫

- 函式呼叫之引數傳遞與回傳值如下圖。



範例7-2的函式呼叫與回傳值

7.3.1 函式呼叫

- 【實用及觀念範例7-3】：製作三個函式，功能分別為計算奇數和、偶數和、整數和。
- 範例7-3：ch7_03.c

```
1  /***************************
2      檔名:ch7_03.c
3      功能:函式應用
4   ***************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  int Odd(int U);
8  int Even(int U);
9  int TotalSum(int U);
10 int main(void)
11 {
12  int n,Sum;
13  char AddChoice;
14  printf("1+2+...+n=?請輸入n=");
15  scanf("%d",&n);
16  fflush(stdin);
17  printf("請問要做奇數和(O),偶數和(E),還是整數和(I)?請選擇:");
18  scanf("%c",&AddChoice);
```

```
 switch(AddChoice)
{
   case ’O’: Sum=Odd(n); break;
   case ’E’: Sum=Even(n); break;
   case ’I’: Sum=TotalSum(n); break;
   default:  printf("選擇錯誤\n"); return -1;
}
printf("總和為%d\n",Sum);
}
int Odd(int U)
{
int i,total=0;
 for(i=1;i<=U;i++)
     if(i%2 == 1) total = total + i;
 return total;
}
int Even(int U)
{
 int i,total=0;
 for(i=1;i<=U;i++)
     if(i%2 == 0) total = total + i;
 return total;
}
int TotalSum(int U)
{ return Odd(U)+Even(U); }
```

7.3.1 函式呼叫

- 執行結果：

```
1+2+...+n=?請輸入n=10
請問要做奇數和(O),偶數和(E),還是整數和(I)?請選擇:I
總和為55
```

7.3.1 函式呼叫

- 【實用範例7-4】：製作階層函式（factorial函式），功能為計算某一正整數的階層k!。並利用該函式求出　　的值，m、n為任意正整數，C_n^m　。

$$C_n^m = \frac{m!}{n!(m-n)!}$$

- 範例7-4：ch7_04.c

7.3.1 函式呼叫

```
1  /*************************
2      檔名:ch7_04.c
3      功能:函式應用
4   *************************/
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  long int factorial(int p); /* 函式宣告 */
10 long int factorial(int p)  /* 函式定義 */
11 {
12  int count;
13  long int result = 1;
14
15  for(count=1;count<=p;count++)
16  {
17    result = result * count;
18  }
19  return result;
20 }
```

7.3.1 函式呼叫

```
21
22  void main(void)
23  {
24   int m,n;
25   long int ans;
26   long int temp[3];
27
28   printf("求排列組合C(m,n)\n");
29   printf("m = ");
30   scanf("%d",&m);
31   printf("n = ");
32   scanf("%d",&n);
33
34   temp[0] = factorial(m);      /*  計算 m! 的值     */
35   temp[1] = factorial(n);      /*  計算 n! 的值     */
36   temp[2] = factorial(m-n);    /*  計算 (m-n)! 的值 */
37   ans = (temp[0])/(temp[1]*temp[2]);  /* C(m,n) = (m!)/(n!*(m-n)!) */
38   printf("C(%d,%d) = %d\n",m,n,ans);
39   /*  system("pause");  */
40  }
```

7.3.1 函式呼叫

- 執行結果：

```
求排列組合C(m,n)
m = 10
n = 8
C(10,8) = 45
```

- 範例說明：
  - 在這個範例中，factorial()函式一共被呼叫了3次（第34、35、36行），充分利用了模組化特性，提高程式碼的重複使用率。

7.3.2 函式的位置

- 【觀念範例7-6】：函式宣告的重要性
- 範例7-6：ch7_06.c

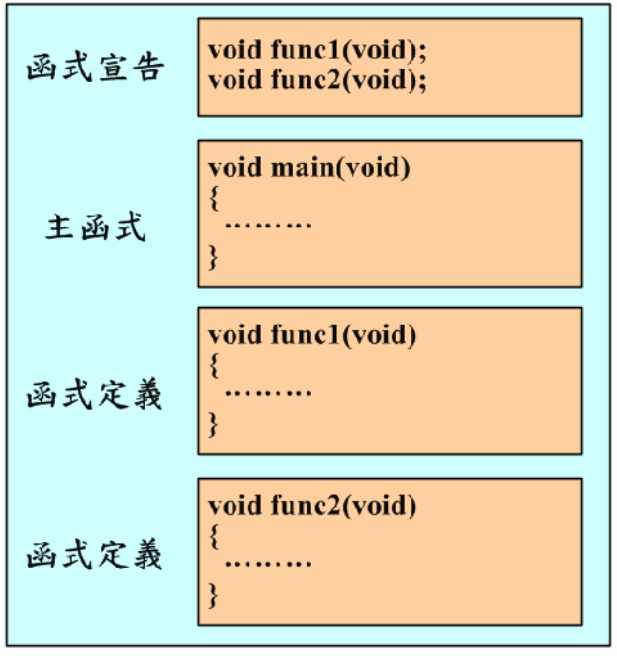
```
1  /***************************************
2      檔名:ch7_06.c
3      功能:函式宣告,定義,呼叫的相對位置
4   ***************************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void func1(void); /* func1函式宣告 */
8  void func2(void); /* func2函式宣告 */
9  void main(void)
10 {
11  func1();
12  func2();}
13  void func1(void)  /* func1函式定義 */
14 {
15   printf("func1函式正在執行中...\n");
16 }
17  void func2(void)  /* func2函式定義 */
18 {
19   printf("func2函式正在執行中...\n");
20 }
```

- 執行結果

```
func1函式正在執行中...
func2函式正在執行中...
```

7.3.2 函式的位置

- 範例說明：
  - (1)在程式範例中，我們將函式func1( )、func2的宣告放在主函式及其他函式之前。
  - (2)在程式開頭的函式宣告（第9、10行），使得編譯器得知程式中存在func1及func2函式，因此我們可以在程式的任何地方呼叫這兩個函式。：



7.3.2 函式的位置

- 無函式宣告型【合併函式宣告與定義】
  - 其實對於C語言來說，所有的程式都是以函式作為基本單位。
  - main是一個特殊的函式，在文字模式下，它是執行程式的入口處，換句話說，沒有了main函式，則程式無法執行。
  - 函式宣告與定義可以寫在一起，因此可以省略函式宣告，但會造成函式呼叫上的某些限制。

7.3.2 函式的位置

- 【觀念範例7-7】：函式宣告與定義的合併。
- 範例7-7：ch7_07.c

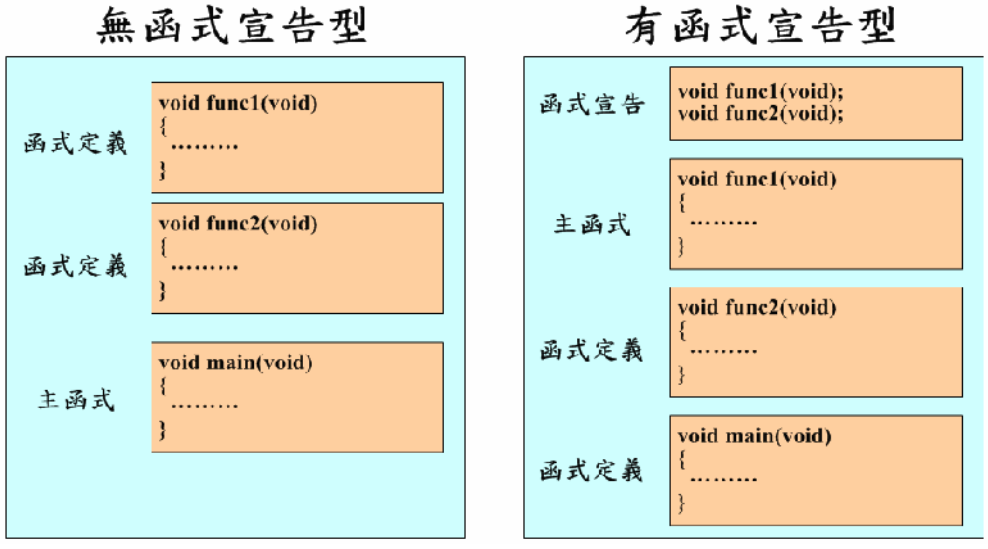
```
1  /*******************************
2      檔名:ch7_07.c
3      功能:函式宣告與定義的合併
4   *******************************/
5   #include <stdio.h>
6  #include <stdlib.h>
7  void func1(void)  /* func1函式定義(省略函式宣告) */
8  {
9     printf("func1函式正在執行中...\n");
10 }
11 void func2(void)  /* func2函式定義(省略函式宣告) */
12 {
13    printf("func2函式正在執行中...\n");
14 }
15 void main(void)
16 {
17  func1();
18  func2();
19 }
```

- 執行結果：

```
func1函式正在執行中...
func2函式正在執行中...
```

7.3.2 函式的位置

- 範例說明：
  - 在程式範例中，我們將函式func1( )、func2( )的宣告省略，但將函式定義移到主函式之前，因此在main( )函式內仍可使用『func1();』、『func2();』敘述來呼叫這兩個函式。



函式宣告與函式定義的合併

7.3.3 return敘述

- return敘述一共有2個功用：(1)回傳函式資料及(2)函式返回。
- return回傳資料
  - 使用return回傳資料的語法如下：

return 常數、變數、運算式或其他具有結果值的敘述；

  - Ex：

```
int func1(.....)
{
 int a;
 .....
 return a;
}
```

7.3.3 return敘述

- 使用return返回
  - 使用return返回呼叫函式敘述如下：

```
return;
```

或

```
return  函式回傳值;
```

- 【語法說明】：
  - 使用return將返回呼叫函式處，並由呼叫函式的下一個敘述開始執行。
  - 一個函式的return並不限定為一個，不過一但執行return敘述後，其餘函式內未被執行的敘述將不會被執行。
  - 無回傳值的函式，不需要用return敘述返回，而若使用return敘述返回，則在return之後的敘述不會被執行。

7.3.3 return敘述

- 【觀念範例7-8】：設計一個包含有多個return敘述的函式，觀察其執行過程。
- 範例7-8：ch7_08.c

```
1  /***************************
2     檔名:ch7_08.c
3     功能:return返回
4  ***************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  int func1(void)
8  {
9  int a=5,b=7;
10 a++;
11 return a+b;
12 a++;
13 return a+b;}
14 void main(void)
15 {
16 int k;
17 k=func1();
18 printf("k=%d\n",k);
19 }
20
```

- 執行結果：

```
k=13
```

7.4 好用的亂數函式

- 在程式設計中，我們可以使用亂數函式來模擬大量的隨機資料，例如統計與實驗、電腦遊戲、樂透開獎等等。
- ANSI C <stdlib>函式庫所提供的srand( )與rand( )函式即可。

7.4 好用的亂數函式

- rand( )【取亂數】

  標頭檔：#include <stdlib.h>
  語法：int rand(void);
  功能：產生亂數

  - 【語法說明】：
    - 回傳值為隨機產生的一個整數數值。
- srand( )【設定亂數產生器種子】

  標頭檔：#include <stdlib.h>
  語法：void srand(unsigned int seed);
  功能：設定亂數產生器種子

  - 【語法說明】：
    - 必須在使用rand函式之前。

7.4 好用的亂數函式

- 【觀念範例7-10】：使用rand亂數函式產生6個隨機亂數。（事先不設定亂數種子）。
- 範例7-10：ch7_10.c

```
1  /*****************************
2      檔名:ch7_10.c
3      功能:rand函式練習
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   void main(void)
8  {
9   int i;
10  for (i=1;i<=6;i++)
11  {
12    printf("第%d個隨機亂數為%d\n",i,rand());
13  }
14 }
```

7.4 好用的亂數函式

- 第一次執行結果：

```
第1個隨機亂數為41
第2個隨機亂數為18467
第3個隨機亂數為6334
第4個隨機亂數為26500
第5個隨機亂數為19169
第6個隨機亂數為15724
```

- 第二次執行結果：

```
第1個隨機亂數為41
第2個隨機亂數為18467
第3個隨機亂數為6334
第4個隨機亂數為26500
第5個隨機亂數為19169
第6個隨機亂數為15724
```

7.4 好用的亂數函式

- 【觀念範例7-11】：先使用srand( )函式設定亂數種子，再使用rand亂數函數產生6個隨機亂數。
- 範例7-11：ch7_11.c

```
1  /*******************************
2      檔名:ch7_11.c
3      功能:srand與rand函式練習
4   *******************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <time.h>
8  void main(void)
9  {
10  int i;
11  srand((unsigned) time(NULL));
12  for (i=1;i<=6;i++)
13  {
14    printf("第%d個隨機亂數為%d\n",i,rand());
15  }
16 }
```

7.4 好用的亂數函式

- 第一次執行結果：

```
第1個隨機亂數為16213
第2個隨機亂數為16729
第3個隨機亂數為119
第4個隨機亂數為20433
第5個隨機亂數為26390
第6個隨機亂數為20219
```

- 第二次執行結果：

```
第1個隨機亂數為16285
第2個隨機亂數為23818
第3個隨機亂數為32682
第4個隨機亂數為25538
第5個隨機亂數為23935
第6個隨機亂數為8303
```

- 範例說明：
  - 由於我們想要設定不同的亂數種子，因此，可以配合time函式取得系統時間作為種子的依據。

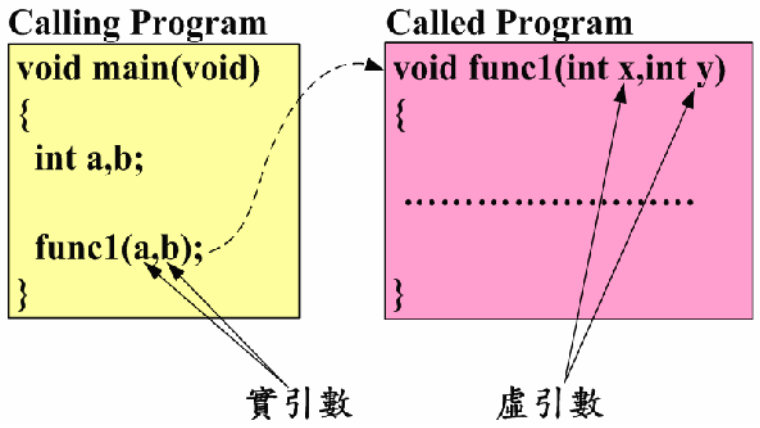
7.4 好用的亂數函式

- time( )

標頭檔：#include <time.h>
語法：time_t time(time_t *timrptr);
功能：取得由格林威治時間1970/1/1 00:00:00至今經過的秒數

- 【語法說明】：
  - *timrptr是存放函式結果的時間指標，一般設為NULL即可使用time函式。

7.5 引數串列與引數傳遞



- 關於引數的傳遞，一般程式語言會將之分為傳值呼叫（Call by value）與傳址呼叫（Call by address）兩種，其個別意義如下：
  - 傳值呼叫：將實引數的值傳遞給虛引數，各自獨立，不互相影響。
  - 傳址呼叫：將實引數的位址傳遞給虛引數，使得彼此互相影響。

7.5 引數串列與引數傳遞

- C語言支援上述兩種傳遞引數的方法，在C語言中將之稱為
  - 傳值呼叫（Pass by value）。
  - 傳指標呼叫（Pass by pointer），其中傳指標呼叫屬於『傳址呼叫』類型。

7.5.1 傳值呼叫（Pass by value）

- 什麼是「傳值呼叫」（Pass by value）呢？傳值呼叫就是在呼叫函式時，只會將實引數『數值』傳遞給函式中相對應的虛引數，所以不論被呼叫函式在執行過程中如何改變虛引數的變數值，都不會影響原本呼叫方實引數的變數值，這種引數傳遞方式稱為「傳值呼叫」。
  - 換句話說，當傳值呼叫發生時，只是把『值』傳送過去，並沒有把記憶體位址也傳送過去。
  - 在本節之前的所有範例，就是使用這種傳遞引數的方式，我們藉由下面一個範例重新加以闡述何謂「傳值呼叫」（Pass by value）。

7.5.1 傳值呼叫（Pass by value）

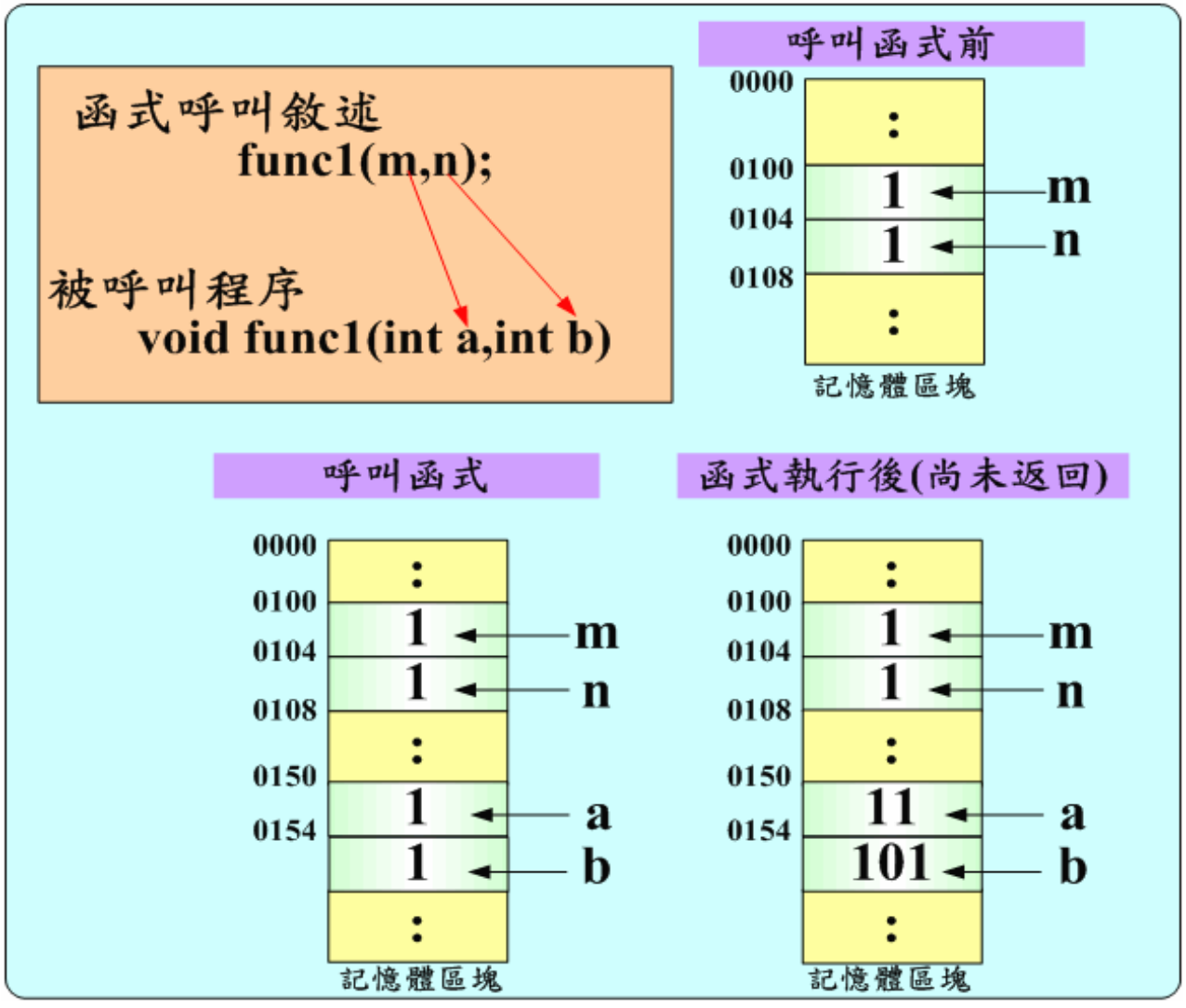
- 【觀念範例7-13】：透過觀察變數內容，了解傳值呼叫傳遞原理。
- 範例7-13：ch7_13.c

```
1  /*****************************
2      檔名:ch7_13.c
3      功能:傳值呼叫
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void func1(int a,int b)
8  {
9   a=a+10;
10  b=b+100;
11  printf("func1()的a=%d\n",a);
12  printf("func1()的b=%d\n",b);
13 }
14  void main(void)
15 {
16  int m=1,n=1;
17  func1(m,n);
18  printf("main( )的m=%d\n",m);
19  printf("main( )的n=%d\n",n);
20  /*  system("pause");  */
21 }
```

- 執行結果：

```
func1()的a=11
func1()的b=101
main( )的m=1
main( )的n=1
```

7.5.1 傳值呼叫（Pass by value）



傳值呼叫示意圖

7.5.2 傳指標呼叫（Pass by pointer）

- 傳指標呼叫（Pass by pointer）是「傳址呼叫」的一種，換句話說，在呼叫發生時，將會把引數的『位址』傳遞給被呼叫的函式。而『位址』就是『變數的記憶體位址』。

- 因此，兩個函式（呼叫與被呼叫者）將共有這些記憶體位址，所以會互相影響，因而可以用來做為需要取得被呼叫者超過一個以上的運算結果時使用。

7.5.2 傳指標呼叫（Pass by pointer）

- 傳指標呼叫（Pass by pointer）的語法如下：
  - 函式宣告語法：

```
函式回傳值型態  函式名稱(資料型態 *指標引數1,資料型態 *指標引數2);
```

  - 函式呼叫語法：

```
函式名稱(&變數1, &變數2);
```

  - 【語法說明】：
    - 『&』運算子具有取記憶體位址的功用。而『*指標引數』代表接收端使用指標來接收傳送過來的資料，換句話說，該指標將指向變數的記憶體位址。函式內若要更改變數內容，必須使用指標運算。
    - 使用『&』運算子來傳遞位址，最常見到的範例為scanf()函式。若忘了加上『&』，將造成無法預期的後果。

7.5.2傳指標呼叫（Pass by pointer）

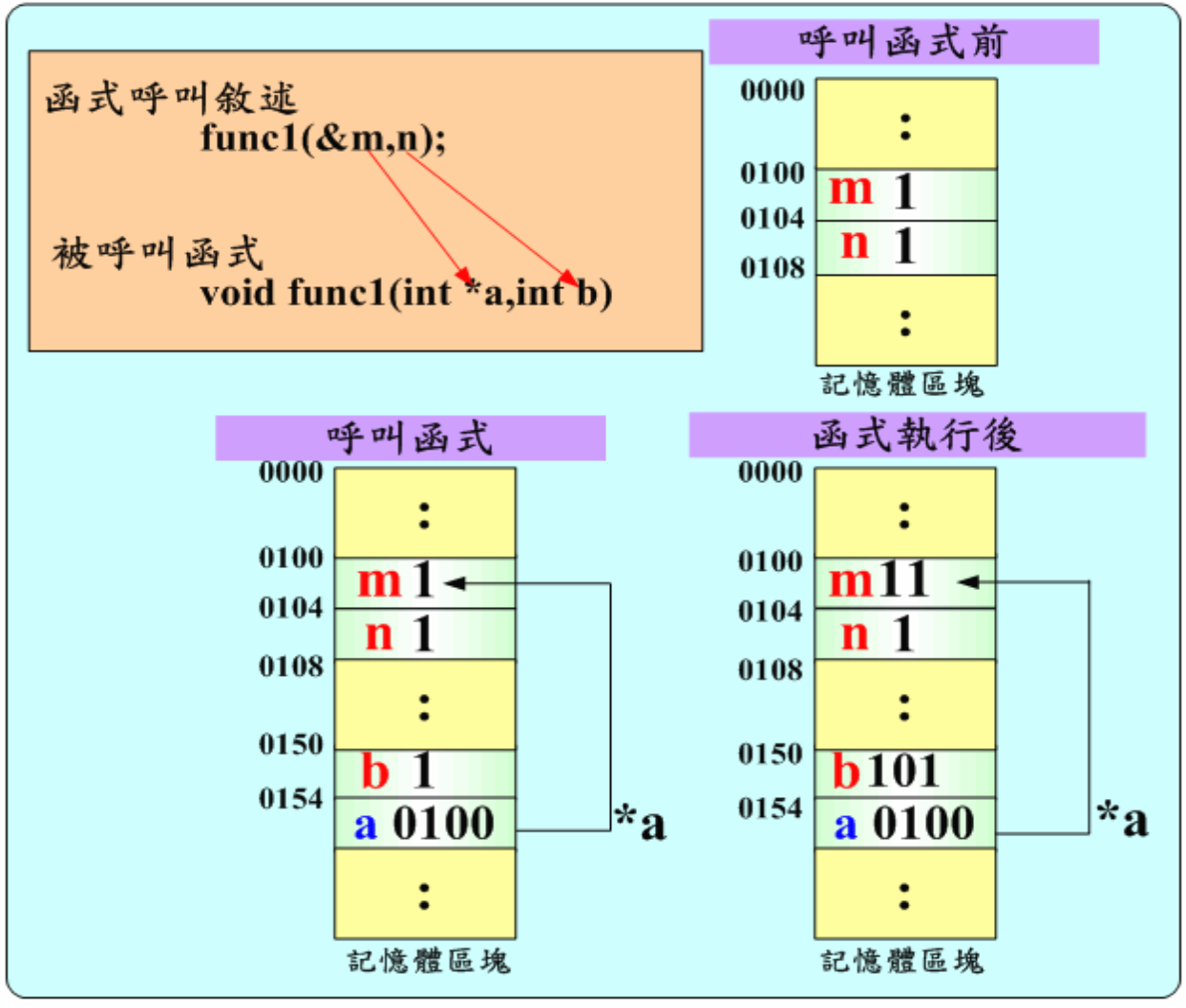
- 【觀念範例7-14】：透過觀察指標及變數內容，了解傳指標呼叫的引數傳遞原理。
- 範例7-14：ch7_14.c

```
1  /*****************************
2      檔名:ch7_14.c
3      功能:傳址呼叫(傳指標呼叫)
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   void func1(int *a,int b)
8  {
9   *a=*a + 10;
10  b=b+100;
11  printf("func1()的*a=%d\n",*a);
12  printf("func1()的b=%d\n",b);
13 }
14  void main(void)
15 {
16  int m=1,n=1;
17  func1(&m,n);
18  printf("main( )的m=%d\n",m);
19  printf("main( )的n=%d\n",n);
20  /*  system("pause");  */
21 }
```

- 執行結果：

```
func1()的*a=11
func1()的b=101
main( )的m=11
main( )的n=1
```

7.5.2傳指標呼叫（Pass by pointer）



傳指標呼叫示意圖(引數a)

7.5.4 傳遞陣列

- 函式宣告語法一：

```
函式回傳值資料型態  函式名稱(int *,int);
```

- 函式宣告語法二：

```
函式回傳值資料型態  函式名稱(int [],int);
```

- 函式宣告語法三：

```
函式回傳值資料型態  函式名稱(int [n],int);
```

  - 【語法說明】：
    - 以上三種都可以傳遞整數一維陣列（第一個引數是陣列名稱、第2個引數是陣列大小）。但語法三（n為陣列索引值）常常會讓程式設計師對於陣列索引的限制產生疑問，事實上，該陣列索引值只是給程式設計師看的，編譯器並不會對陣列索引值做任何檢查，因此不會有錯誤。

7.5.4 傳遞陣列

- 【觀念範例7-17】：利用傳址呼叫傳遞陣列。
- 範例7-17：ch7_17.c

```
1  /*************************
2      檔名:ch7_17.c
3      功能:傳址呼叫傳遞參數
4   *************************/
5
6   #include <stdio.h>
7  #include <stdlib.h>
8  #include <time.h>
9   void generate_lotto(int *arr,int arr_index)
10 {
11  int i;
12  srand((unsigned) time(NULL));
13  printf("樂透號碼開獎中.....\n");
14  for (i=0;i<arr_index;i++)
15  {
16   arr[i]=rand()%42+1;
17   printf("第%d個號碼為%d\n",i+1,arr[i]);
18  }
19 }
20 void main(void)
21 {
22  int lotto[6],i;
23  generate_lotto(lotto,6);
24  printf("樂透號碼如下.....\n");
25  for (i=0;i<6;i++)
26     printf("%d\t",lotto[i]);
27  printf("\n");
   }
```

7.5.4 傳遞陣列

- 執行結果：

```
樂透號碼開獎中.....
第1個號碼為3
第2個號碼為26
第3個號碼為12
第4個號碼為30
第5個號碼為22
第6個號碼為1
樂透號碼如下.....
3        26        12        30        22        1
```

- 範例說明：
  - 您可以將第10行改寫如下三種格式，但都具有相同效果。
    - void generate_lotto(int arr[],int arr_index)
    - void generate_lotto(int arr[3],int arr_index)
    - void generate_lotto(int arr[6],int arr_index)
  - 上述的陣列索引3或6並不重要，陣列大小將由arr_index決定。

7.5.4 傳遞陣列

- 從執行結果中，我們可以得知lotto與arr都指向同一個陣列的起始位址，所以在generate_lotto函式中對arr陣列元素的修改都將會影響main函式的lotto陣列元素值。

- 【回傳值的指標傳遞】：
  - 除了引數的傳遞可以使用指標傳遞之外，函式回傳值也可以使用指標傳遞，不過一般在使用上的意義不大，因為C的函式只能回傳一個值。

7.5.5 搜尋演算法【補充】

- 排序的主要目的其實是方便於搜尋資料，至於搜尋的方法，其實也是分成許多種，每一種的難度與效率皆不相同，以下是兩種常用的搜尋法：
  - 循序搜尋法
  - 二分搜尋法

- 循序搜尋法
  - 『循序搜尋』是一種簡單到不能再簡單的搜尋方法，也就是從第一筆資料開始尋找，然後是第二筆資料、、、一直到找到所要的資料或全部資料被找完為止。

7.5.5 搜尋演算法【補充】

- 【實用範例7-18】：使用循序搜尋法在未排序的資料中，尋找所需要的資料『57』。。
- 範例7-18：ch7_18.c

```
1  /*****************************
2      檔名:ch7_18.c
3      功能:循序搜尋法
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  int SeqSearch(int Target,int *arr,int arr_index)
8  {
9   int i;
10  for(i=0;i<arr_index;i++)
11     if(Target == arr[i])        /* 找到了 */
12        return i;
13  return -1;                    /* 完全找不到 */
14 }
```

7.5.5 搜尋演算法【補充】

```
15
16  void main(void)
17  {
18   int    work[11]={43,23,67,27,39,15,39,37,57,26,14};
19   int FindNumber,location;
20    printf("請輸入您要找的數值:");
21   scanf("%d",&FindNumber);
22   location=SeqSearch(FindNumber,work,11);
23   if(location==-1)
24      printf("在陣列中找不到要找的數值\n");
25   else
26      printf("數值%d位於work[%d]\n",FindNumber,location);
27   }
28
```

7.5.5 搜尋演算法【補充】

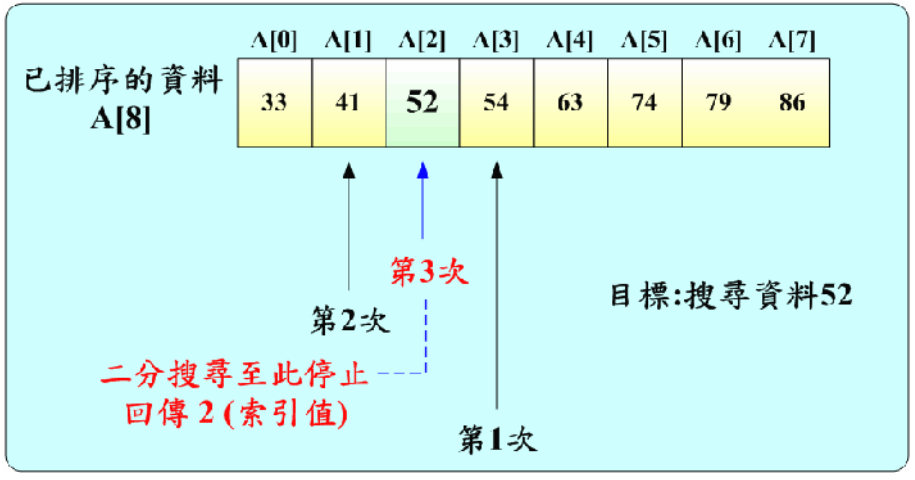
- 執行結果：

```
請輸入您要找的數值:57
數值57位於work[8]
```

- 範例說明：
  - 循序搜尋函式:可以接受一個尋找目標（整數資料型態）及一個工作陣列。
  - 主使用『傳值呼叫』傳遞FindNumber→Target引數。使用『傳指標呼叫』，傳送work[ ] →arr[ ]陣列（11是陣列大小）。

7.5.5 搜尋演算法【補充】

- 二分搜尋法
  - 二分搜尋法比循序搜尋法來得有效率許多。
  - 雖然速度比較快，但使用二分搜尋法找尋資料必須先將資料經過排序之後，才可以使用二分搜尋法。
  - 以下是用二分搜尋法的原理及步驟：



二分搜尋法

```
/*****************************
    檔名:ch7_19.c
    功能:二分搜尋法
 *****************************/
#include <stdio.h>
#include <stdlib.h>
int BinarySearch(int Target,int *arr,int arr_index)
{
 int Low,Upper,m; Low=0; Upper=arr_index-1;
 while (Low<=Upper)
 {
    m=(Low+Upper)/2;      /* 計算中間位置 */
    if(arr[m]==Target)    /* 找到了 */
    {
       return m;
    }
    else
    {
      if(arr[m]>Target)   /* Target位於上半部 */
           Upper=m-1;
      else                /* Target位於下半部 */
           Low=m+1;
    }
 }
 return -1;
}
```

7.5.5 搜尋演算法【補充】

```
24  void main(void)
25  {
26   int work[8]={33,41,52,54,63,74,79,86};
27   int FindNumber,location;
28    printf("請輸入您要找的數值:");
29   scanf("%d",&FindNumber);
30   location=BinarySearch(FindNumber,work,8);
31   if(location==-1)
32      printf("在陣列中找不到要找的數值\n");
33   else
34      printf("數值%d位於
35  work[%d]\n",FindNumber,location);
36   /*  system("pause");  */
37  }
```

- 執行結果：

```
請輸入您要找的數值:52
數值52位於work[2]
```

7.5.7main函式的引數串列與回傳值

- main函式的引數一共有3個如下：
  - 第一個引數『argc』：整數型態，代表作業系統一共傳遞過來的命令列參數（指令本身也算一個）。
  - 第二個引數『argv』：字串陣列型態，存放作業系統傳遞過來幾個命令列參數字串（一個命令列參數代表一個參數）。
  - 第三個引數『env』：指標陣列，接收程式環境設定的指標陣列（很少用)。

7.5.7main函式的引數串列與回傳值

- 依據main()函式引數個數的不同，main()函式一般有下列4種不同的定義方式。
  - 定義方式一：
    - int main()
  - 定義方式二：
    - int main(int argc)
  - 定義方式三：
    - int main(int argc,char *argv[])
  - 定義方式四：
    - int main(int argc,char *argv[],char *env[])

7.5.7main函式的引數串列與回傳值

- 由於字串是一種特殊的字元陣列，而指標也可以用來代表陣列的起始位址，因此我們可以使用指標的指標（詳見下一章）來代替字串陣列，也可以使用二維字元陣列來代替字串陣列，例如上述的定義方式三可以改寫為下列三種形式：
  - 定義方式三（型式一）：
    - int main(int argc,char *argv[])
  - 定義方式三（型式二）：
    - int main(int argc,char argv[][]) /* 使用二維字元陣列（字串陣列） */
  - 定義方式三（型式三）：
    - int main(int argc,char **argv) /* 使用指標的指標 */

7.5.7main函式的引數串列與回傳值

- 【觀念範例7-22】：接收由命令列傳送過來的參數。
- 範例7-22：ch7_22.c

```
1  /*****************************
2      檔名:ch7_22.c
3      功能:main函式的引數
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   void main(int argc,char *argv[])
8  {
9   int i;
10  printf("本程式共接受到命令列%d個參數\n",argc);
11  for (i=0;i<=argc;i++)
12     printf("argv[%d]字串為%s\n",i,argv[i]);
13 }
```

7.5.7main函式的引數串列與回傳值

- 執行結果：

```
C:\C_language\ch07>ch7_22 Time C
language
本程式共接受到命令列4個參數
argv[0]字串為ch7_22
argv[1]字串為Time
argv[2]字串為C
argv[3]字串為language
argv[4]字串為(null)
```

7.5.7main函式的引數串列與回傳值

- 範例說明：
  - 在執行結果中，我們先將ch7_22.c編譯為ch7_22執行檔。然後執行ch7_22執行檔，並輸入『Time』、『C』、『language』等3個額外參數。
  - 由於輸入了3個額外參數，因此argc=4（因為執行檔本身也算一個參數）。
  - argv[0]存放的是第一個命令列參數，也就是執行檔路徑與名稱，argc[1]存放『Time』、argc[2]存放『C』、argc[3]存放『language』，argc[4]由於已經達到字串陣列的結尾，因此將是(null)。

7.6自訂函式庫之引入標頭檔（#include）

- 當我們把許多好用的函式寫好之後，可以將之分類並放置於一個副檔名為『.h』的函式庫檔案中，如此一來就可以在要使用該函式的時候，直接將該函式庫檔案引入即可。

- 引用方法
  - #include <檔名> /* 引入 C語言編譯器提供的標頭檔 */
  - #include "路徑及檔名"/* 引入非編譯器提供的標頭檔 */

7.6自訂函式庫之引入標頭檔（#include）

7.7 遞迴函式

- 在前面的章節中，我們已經練習很多在函式中呼叫另一個函式的方法。聰明的讀者不知是否曾經想過，當一個函式呼叫自己的時候會發生什麼狀況呢？這就是所謂函式的『遞迴呼叫』（recursive call）。

- 其實『遞迴呼叫』應該明確定義如下：
  - 遞迴呼叫：一個函式經由直接或間接呼叫函式本身，稱之為函式的『遞迴呼叫』。

7.7 遞迴函式

- 『費氏數列』：一個無限數列，該數列上的任一元素值皆為前兩項元素值的和，數列如下所示：
  - 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144.....
  - 當我們使用數學函數來表示費氏數列時，費式數列的定義本身就是一個遞迴定義如下：

費氏數列的遞迴定義式	
F(0) = 0	n ＝ 0
F(1) = 1	n ＝ 1
F(n) = F(n-1)+F(n-2)	n ≧ 2

7.7 遞迴函式

- 從上述定義中，我們可以發現，在實際計算時（例如計算F(10)），數學的遞迴函數必須不斷地重複呼叫自己，直到遇上非遞迴定義式為止，才能夠求出答案。
- 使用C語言的遞迴函式時，也必須對該函式做出某些限制條件，以避免函式無窮的執行下去，通常一個遞迴函式需符合下列兩個限制條件：
  - 遞迴函式必須有邊界條件，當函式符合邊界條件時，就應該返回（可使用return 強制返回）函式呼叫處，在費式數列中，F(0)=0與F(1)=1就是函式的邊界條件。
  - 遞迴函式在邏輯上，必須使得函式漸漸往邊界條件移動，否則該函式將無法停止呼叫，而無窮地執行下去，最後將造成記憶體不足的問題。

7.7 遞迴函式

- 【實用及觀念範例7-26】：使用遞迴，求出費氏數列0~25的元素值。
- 範例7-26：ch7_26.c

```
1  /*****************************
2      檔名:ch7_26.c
3      功能:使用遞迴求費氏數列
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  \int Fib(int n)
8  {
9   if((n==1) || (n==0))
10    return n;
11  else
12    return Fib(n-1)+Fib(n-2);
13 }
```

```
14
15 void main(void)
16 {
17  int i;
18  printf("費氏數列如下:");
19  for (i=0;i<=25;i++)
20  {
21    if (i%8==0)
22        printf("\n");
23    printf("%d\t",Fib(i));
24  }
25  printf("......\n");
26  /*  system("pause");  */
27 }
```

7.7 遞迴函式

- 執行結果：

```
費氏數列如下:
0       1       1       2       3       5       8       13
21      34      55      89      144     233     377     610
987     1597    2584    4181    6765    10946   17711   28657
46368   75025   ......
```

- 範例說明：
  - 在main函式中，大多數的程式碼都是為了處理列印的問題，實際上最重要的程式出現在第30行呼叫Fib(i)，以便計算費式數列的元素值。
  - 您是否驚訝於Fib()函式竟然如此簡潔有力，幾乎只是將數學定義式轉換為C語言程式語法而已。事實的確如此，這就是遞迴函式的優點。舉例來說，當main()的函式呼叫敘述Fib(4)執行時，函式呼叫與返回狀況如下圖：

7.7 遞迴函式

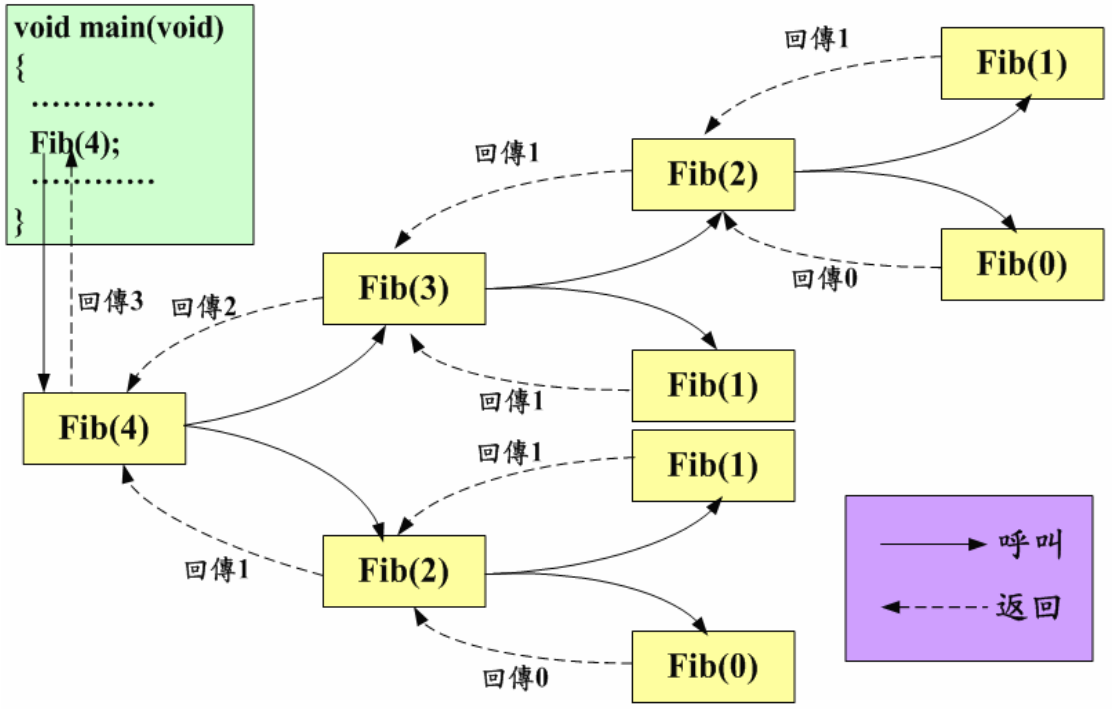


圖7-14 Fib(4)的遞迴呼叫與返回

7.7 遞迴函式

```
1  /*遞迴求最大公因數*/
2  #include <stdio.h>
3  #include <stdlib.h>
4  int gcd(int a, int b);
5  int main(int argc, char *argv[])
6  {
7    int x, y, gcdanswer;
8    printf("Please enter 2 numbers.\n");
9    scanf("%d%d", &x, &y);
10   gcdanswer =  gcd(x, y);
11   printf("The GCD is %d.\n", gcdanswer);
12   system("PAUSE");
13   return 0;
14 }
15 int gcd(int a, int b)
16 {
17   int c;
18   c = a % b;
19   if( c != 0 )
20         return gcd(b, c);
21   else
22         return b;
23 }
24
```

7.7 遞迴函式

	優點	缺點
遞迴	程式簡潔明確 節省記憶體空間	因參數的存取較費時
非遞迴	節省執行時間	程式較長浪費記憶體空間

7.8.1　前置處理(Preprocess)

- 前置處理指令在編譯之前就會被置換成某些程式碼，這個處理動作動作稱為『前置處理』。

- 如#include指令會被需要引入的標頭檔函式庫所替換，而本節要說明的巨集指令也是一種前置處理指令。

7.8.2 巨集指令：#define

- 『巨集指令』又稱為『替換指令』，可替換內容的有「常數」、「字串」或「函式」，語法格式則有下列兩種：
- 語法格式一：

```
#define 巨集名稱        替換內容
```

  - 範例：

```
#define PI      3.14159
#define e       2.71828
#define loop8   for(i=1;i<=8;i++){\
                printf("%d\n",i);\
}
```

- 語法格式二：

```
#define 巨集名稱(參數列)  替換內容
```

  - 範例：

```
#define Sum(x,y)        x+y
#define Div(x,y)        (x)/(y)
```

7.8.2 巨集指令：#define

- 當編譯器的前置處理器在前置處理階段遇到巨集指令時，就會將巨集名稱所在位置的常數、字串或函式替換成『替換內容』。
- 巨集取代常數
  - #define可以定義『巨集名稱』來取代『常數』，所有在該程式中使用此巨集名稱的地方將會被替換成對應的常數，再進行編譯。所以和使用const定義一樣。
  - 範例：巨集取代常數

```
#define a       5
#define b       10
c=a+b;
```

    - 編譯時，『c=a+b;』會變成『c=5+10;』。

7.8.2 巨集指令：#define

- 【觀念及實用範例7-28】：使用巨集替換常數PI、半徑R。
- 範例7-28：ch7_28.c

```
1  /*****************************
2      檔名:ch7_28.c
3      功能:巨集取代常數
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #define PI 3.1416
8  #define R 3
9  void main(void)
10 {
11  float area,length;
12  area=PI*R*R;
13  length=2*PI*R;
14   printf("半徑為3的圓面積為%.4f\n",area);
15  printf("半徑為3的圓周長為%.4f\n",length);
16  /*  system("pause");  */
17 }
```

- 執行結果：

```
半徑為3的圓面積為28.2744
半徑為3的圓周長為18.8496
```

7.8.2 巨集指令：#define

- 巨集取代函式
  - 巨集取代函式時，必須加上引數的宣告，語法格式如下：

```
#define  M1(a,b,c…)  F1(a,b,c…)
```

7.8.2 巨集指令：#define

- 【觀念範例7-30】：使用巨集替換函式。
- 範例7-30：ch7_30.c

```
1  /*****************************
2       檔名:ch7_30.c
3       功能:巨集取代函式
4   *****************************/
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  #define Sum(a,b) a+b
10
11 void main(void)
12 {
13  int a=5,b=10,c;
14
15  c=Sum(a,b);
16  printf("c=%d\n",c);
17  /*  system("pause");  */
18 }
```

- 執行結果：

```
c=15
```

7.8.2 巨集指令：#define

- 範例說明：
  - 第15行看起來好像是執行一個函式呼叫，從執行結果中，也看不出有何端倪。可是，巨集和函式的差別可是很大的，上面的範例程式碼（第15行）會被前置處理器轉換為『c=a+b;』，然後才進行編譯，因此根本不會有函式呼叫及返回的動作。

- 巨集的運算式
  - 當使用巨集代替函式並包含有運算式時，必須特別小心『運算子優先權』的問題，否則將發生下面這個範例的非預期狀況。

7.8.2 巨集指令：#define

- 【觀念範例7-31】：巨集內容包含運算式常犯的錯誤。
- 範例7-31：ch7_31.c

```
1  /*****************************
2      檔名:ch7_31.c
3      功能:巨集包含運算式
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   #define Sum(a,b) a+b
8  #define Mul(a,b) a*b
9  void main(void)
10 {
11  int a=1,b=2,c=3,d=4;
12  int result1,result2;
13   result1=Sum(a*b,c*d);
14  result2=Mul(a+b,c+d);
15  printf("result1=%d\n",result1);
16  printf("result2=%d\n",result2);
17  /*  system("pause");  */
18 }
```

- 執行結果：

```
result1=14
result2=11
```

7.8.2 巨集指令：#define

- 範例說明：
  - 從執行結果中，我們可以發現result2不是我們所預期的結果（3*7=21）。
  - 因為巨集名稱只會笨笨的被取代為替換內容，因此，第18行會被前置處理器替換為『result2=a+b*c+d』，執行結果則為『result2=1+2*3+4=1+6+4=11』。
- 在上面這個範例中，我們得知result2執行結果不是我們所要的(a+b)*(c+d)=21，這是因為運算子優先權所造成的結果，所以解決之道就是在巨集定義時加上小括號，以便替換後保證一定會先被執行，如下修正範例。

7.8.2 巨集指令：#define

- 【觀念範例7-32】：使用小括號修正範例。
- 範例7-32：ch7_32.c

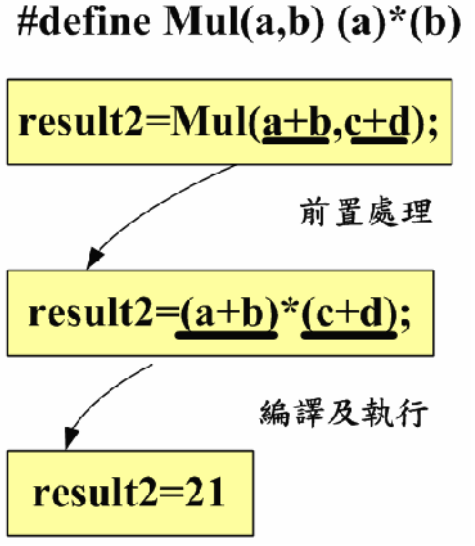
```
/*****************************
    檔名:ch7_32.c
    功能:巨集包含運算式
 *****************************/
#include <stdio.h>
#include <stdlib.h>
#define Sum(a,b) (a)+(b)
#define Mul(a,b) (a)*(b)
void main(void)
{
 int a=1,b=2,c=3,d=4;
 int result1,result2;
 result1=Sum(a*b,c*d);
 result2=Mul(a+b,c+d);
 printf("result1=%d\n",result1);
 printf("result2=%d\n",result2);
 /*  system("pause");  */
}
```

- 執行結果：

```
result1=14
result2=21
```

7.8.2 巨集指令：#define

- 範例說明：
  - 經由修正後的第18行將被前置處理器替換為『result2=(a+b)*(c+d)』，執行結果則為『result2=(1+2)*(3+4)=3*7=21』。



7.8.3 巨集、函式、常數的差別

- 巨集與常數變數
  - 我們可以使用#define來設定巨集名稱以便取代程式中的常數，也可以用const來宣告一個常數變數。如下範例：
    - #define PI 3.1415926
      或
    - const double PI=3.1415926;
  - 雖然兩者都可以達到目的，但對於處理及編譯過程則不太相同。
  - 對於使用const而言，由於在程式編譯的過程中，const宣告的常數變數會被編譯器檢查資料型態，因此建議盡量使用const來定義常數。

7.8.3 巨集、函式、常數的差別

- 巨集與函式
  - 從前面的範例中，讀者可以發現使用巨集的方式也可以做到部分函式的功能，事實上巨集函式與普通函式在處理及效率上有著極大的差異。
  - 『巨集』是前置處理階段的工作，而編譯『函式』則是編譯階段的工作。在效能上，則可以分為時間與空間上的差別：
    - 空間上的差異：使用巨集時，巨集將被取代為程式碼加入到程式中，因此，程式使用越多次巨集，經由前置處理後，編譯器看到的程式碼也就越長，自然編譯出來的執行檔也就越大，執行時所佔用的**記憶體空間也越大**。
    - **時間上的差異：由於呼叫函式及返回函式時，需要對堆疊進行疊入（push）與疊出（pop）的動作，因此會多花費一些時間。而使用巨集時，由於沒有這些動作，因此執行速度比較快。**

7.9 本章回顧

- 在本章中，我們認識了結構化程式語言的重點－『函式』。使用函式來設計程式可以將程式以『模組化』方式切割，製作功能不同的各種函式，並且提高程式的重複使用率。
  - 除了介紹『函式宣告』與『函式定義』之外，我們還介紹了2種C語言提供的函式呼叫，分別是『傳數值呼叫』、『傳指標呼叫』，並且還額外介紹了C++新提供的『傳參考呼叫』。其中傳指標呼叫、傳參考呼叫所依靠的資料傳遞都是記憶體位址，傳指標呼叫使用了指標來接收引數，而傳參考呼叫則使用了參考運算子來接收引數。讀者若對於指標尚未熟悉，應該於下一章閱讀完畢後，重新回頭檢視本章關於指標的內容，將會有更清楚的體認。

7.9 本章回顧

- 具有類似函式功能的另外一種技術稱為『巨集』，但兩者在處理流程上則大不相同，『巨集』只能夠算是一種『前置處理』技術，在編譯階段中，將不會看到任何的巨集，因此，當您使用『巨集』取代『函式』之後，雖然執行速度上會加快許多，但執行檔將佔用較大的磁碟空間，因此並不建議大量使用巨集。

- 就如同我們大量使用ANSI C函式庫所提供的眾多函式，當我們製作完某些常用的『函式』之後，也可以將之包含在特定的自訂『函式庫』中，提供他人日後使用，如此一來，將可以使您的程式被重複使用的機率大增，以便加速完成更大型的程式開發。

本章习题