函數

函數

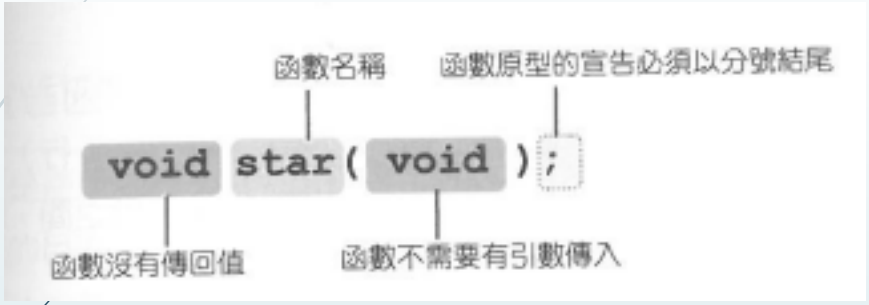
- 函數(function)
- 可以簡化主程式的結構，也可以節省撰寫相同程式碼的時間，達到程式模組化的目的
- 利用前置處理器不但可以完成簡單函數的定義，同時還可以含括所需的檔案到程式中
- C語言使用方式
  - 函數原型(prototype)宣告
  - 函數定義
  - 呼叫函數

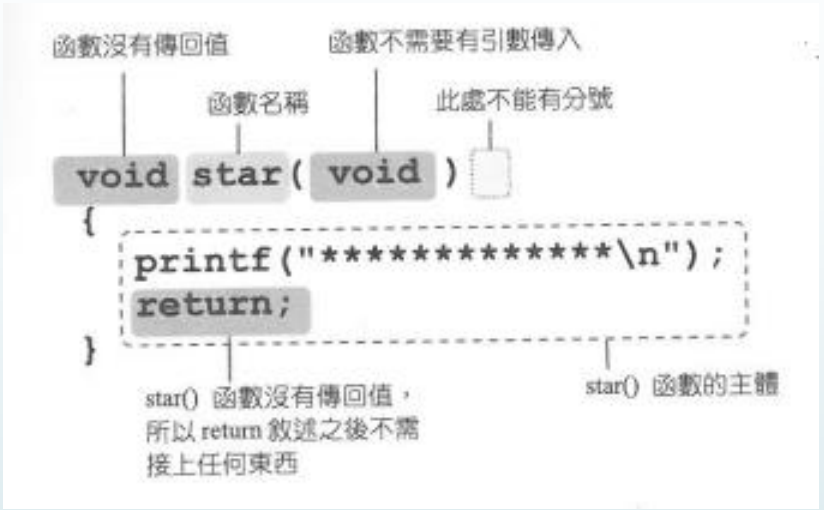
範例-函數範例

```c
#include <stdio.h>
#include <stdlib.h>
void star(void);                    /* star()函數的原型 */
int main(void)
{
    star();                         /* 呼叫star函數 */
    printf("歡迎使用C語言\n");
    star();                         /* 呼叫star函數 */
    system("pause");
    return 0;
}

void star(void)
{
    printf("**************\n");     /* 印出13個星號 */
    return;
}
```

範例詳解





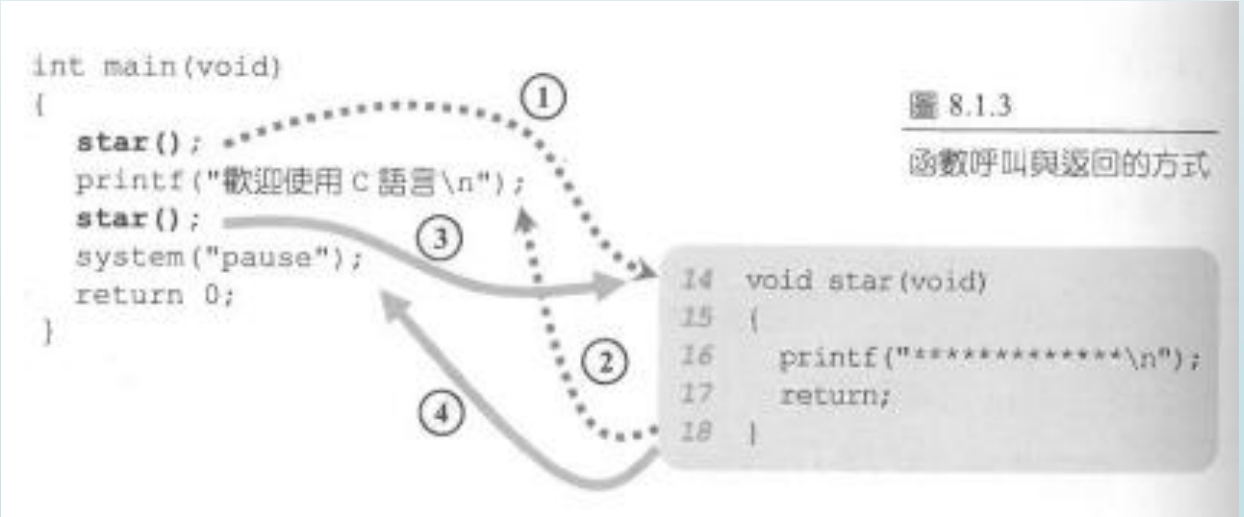


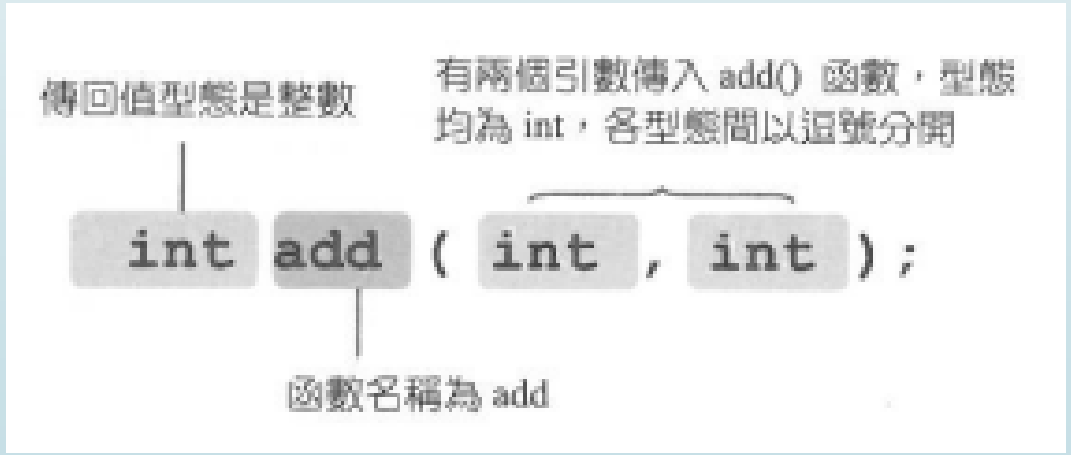
圖 8.1.3
函數呼叫與返回的方式

函數原型宣告

- 宣告要有分號結尾
- 可放在main函數內或外皆可，內表示為只能作用在main函數中，外則表示再本檔案中皆可使用
- 語法

```
傳回值型態 函數名稱(引數型態 1, 引數型態 2,...);
```

- void表示無須傳入參數



函數定義

- 放在main函數外
  - 若放在main前面則不需要宣告函數原型
- return可審略
  - 回傳值可以是一個運算式或值，須注意只能有一個回傳值
- 注意參數數量及資料型態是否與呼叫函數時的參數數量及資料型態一致
- 語法

```
傳回值型態 函數名稱(型態 1 引數 1, …, 型態 n 引數 n)
{
    變數宣告;
    敘述主體;
    return 運算式;     /* 傳回運算式的值 */
}
```

呼叫函數

- 若不需要傳參數的話引數內不需要傳任何內容
- 語法

```
函數名稱(引數);
```

```
變數 = 函數名稱(引數);
```

範例 - 函數加總

```c
#include <stdio.h>
#include <stdlib.h>
int add(int,int);                   /* add()函數的原型 */
int main(void)
{
    int sum, a=5, b=3;
    sum=add(a,b);                   /* 呼叫add()函數，並把傳回值設給sum */
    printf("%d+%d=%d\n",a,b,sum);

    system("pause");
    return 0;
}

int add(int num1, int num2)         /* add()函數的定義 */
{
    int a;                          /* 於add()函數裡宣告變數a */
    a=num1+num2;
    return a;                       /* 傳回num1+num2 的值 */
}
```

範例 - 字元列印函數

```c
#include <stdio.h>
#include <stdlib.h>
void display(char,int); /* display()函數的原型 */
int main(void)
{
    int n;
    char ch;
    printf("請輸入欲列印的字元:");
    scanf("%c",&ch);
    printf("請問要印出幾個字元:");
    scanf("%d",&n);
    display(ch,n);          /* 呼叫自訂的函數，印出n個ch字元 */

    system("pause");
    return 0;
}

void display(char ch,int n) /* 自訂的函數display() */
{
    int i;
    for(i=1;i<=n;i++)               /* for迴圈，可印出n個ch字元 */
        printf("%c",ch);            /* 印出ch字元 */
    printf("\n");
    return;
}
```

範例 - 絕對值函數

```c
#include <stdio.h>
#include <stdlib.h>
int abs(int);              /* 宣告函數abs()的原型 */
int main(void)
{
    int i;
    printf("Input an integer:");      /* 輸入整數 */
    scanf("%d",&i);
    printf("abs(%d)=%d\n",i,abs(i)); /* 印出絕對值 */

    system("pause");
    return 0;
}

int abs(int n)  /* 自訂的函數abs(),傳回絕對值 */
{
    if (n<0)
        return -n;
    else
        return n;
}
```

範例 - 次方函數

```c
#include <stdio.h>
#include <stdlib.h>
double power(double, int);  /* 宣告函數power()的原型 */
int main(void)
{
    double x;          /* x為底數 */
    int n;         /* n是次方 */

    printf("請輸入底數與次方:");
    scanf("%lf,%d",&x,&n);    /* 輸入底數與次方 */
    printf("%lf的%d次方=%lf\n",x,n,power(x,n));

    system("pause");
    return 0;
}

double power(double base, int n)    /* power()函數的定義 */
{
    int i;
    double pow=1.0;
    for(i=1;i<=n;i++)          /* for() 迴圈，用來將底數連乘n次 */
        pow=pow*base;
    return pow;
}
```

範例 - 質數測試函數

```c
#include <stdio.h>
#include <stdlib.h>
int is_prime(int);              /* 宣告函數is_prime()的原型 */
int main(void)
{
    int i;
    for(i=2;i<=30;i++)          /* 找出小於30的所有質數 */
        if(is_prime(i))         /* 呼叫is_prime()函數 */
            printf("%3d",i);    /* 如果是質數,便把此數印出來 */
    printf("\n");
    system("pause");
    return 0;
}

int is_prime(int num)    /* is_prime()函數,可測試num是否為質數 */
{
    int i;
    for(i=2;i<=num-1;i++)
        if(num%i==0)
            return 0;
    return 1;
}
```

質數測試函數-變數變化

表 8.3.2　num 分別等於 7 和 9 時，is_prime() 函數內變數變化的情形

num=7

i	num%i
2	7%2=1
3	7%3=1
4	7%4=3
5	7%5=2
6	7%6=1
7	

2~6 都無法整除 7，所以 7 是質數

不符合 for 迴圈的判斷條件（i<=num-1），跳出 for 迴圈，執行第 22 行，傳回 1，代表 7 是質數

num=9

i	num%i
2	9%2=1
3	9%3=0

3 可以整除 9，所以 9 不是質數

符合 20 行的判斷條件，傳回 0，代表 9 不是質數

範例 - 同時使用多個函數

```c
#include <stdio.h>
#include <stdlib.h>
void sum(int), fac(int);          /* 定義函數的原型 */
int main(void)
{
   fac(5);              /* 呼叫fac()函數 */
   sum(5);              /* 呼叫sum()函數 */

   system("pause");
   return 0;
}

void fac(int a)      /* 自訂函數fac()，計算a! */
{
   int i,total=1;
   for(i=1;i<=a;i++)
      total*=i;
   printf("1*2*...*%d=%d\n",a,total);    /* 印出a!的結果 */
}

void sum(int a)      /* 自訂函數sum()，計算1+2+...+a的結果*/
{
   int i,total=0;
   for(i=1;i<=a;i++)
      total+=i;
   printf("1+2+...+%d=%d\n",a,total);    /* 印出加總的結果 */
}
```

萊布尼茲公式

- 萊布尼茲公式

$$\pi = 4\sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{2k-1} = 4\left(\frac{(-1)^{1-1}}{2(1)-1}+\frac{(-1)^{2-1}}{2(2)-1}+\frac{(-1)^{3-1}}{2(3)-1}+\frac{(-1)^{4-1}}{2(4)-1}+\frac{(-1)^{5-1}}{2(5)-1}+\ldots\right)$$

$$= 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\ldots\right)$$

$$\text{Leibniz(n)} = 4\sum_{k=1}^{n}\frac{(-1)^{k-1}}{2k-1}$$

```
20    sum=sum+power(-1.0,k-1)/(2*k-1);        /* 萊布尼茲公式 */
```

範例 - 函數相互呼叫

```c
#include <stdio.h>
#include <stdlib.h>
double Leibniz(int);                    /* 宣告函數Leibniz()的原型 */
double power(double, int);          /* 宣告函數power()的原型 */
int main(void)
{
    int i;
    for(i=1;i<=10000;i++)               /* 找出前10000個π 的估算值 */
        printf("Leibniz(%d)=%12.10f\n",i,Leibniz(i));
    system("pause");
    return 0;
}

double Leibniz(int n)    /* Leibniz()函數，可估算π 值到第n項*/
{
    int k;
    double sum=0.;
    for(k=1;k<=n;k++)
        sum=sum+power(-1.0,k-1)/(2*k-1);        /* 萊布尼茲公式 */
    return 4*sum;
}

double power(double base, int n) /* power()函數,可計算base的n次方 */
{
  int i;
    double pow=1.0;
    for(i=1;i<=n;i++)
        pow=pow*base;
    return pow;
}
```

遞迴函數

- 遞迴(recursive)
  - 函數本身呼叫自己
  - 可以讓程式碼變得簡潔
  - 一定要可以結束函數執行的終止條件，容易造成無窮迴圈
  - 佔用較大記憶體空間
    - 每一個未執行完畢的函數與區域變數，會佔用記憶體空間來存放他們，等到開始返回時再由記憶體空間取出未完成的部分繼續執行，被佔用的記憶體空間才會一一釋放

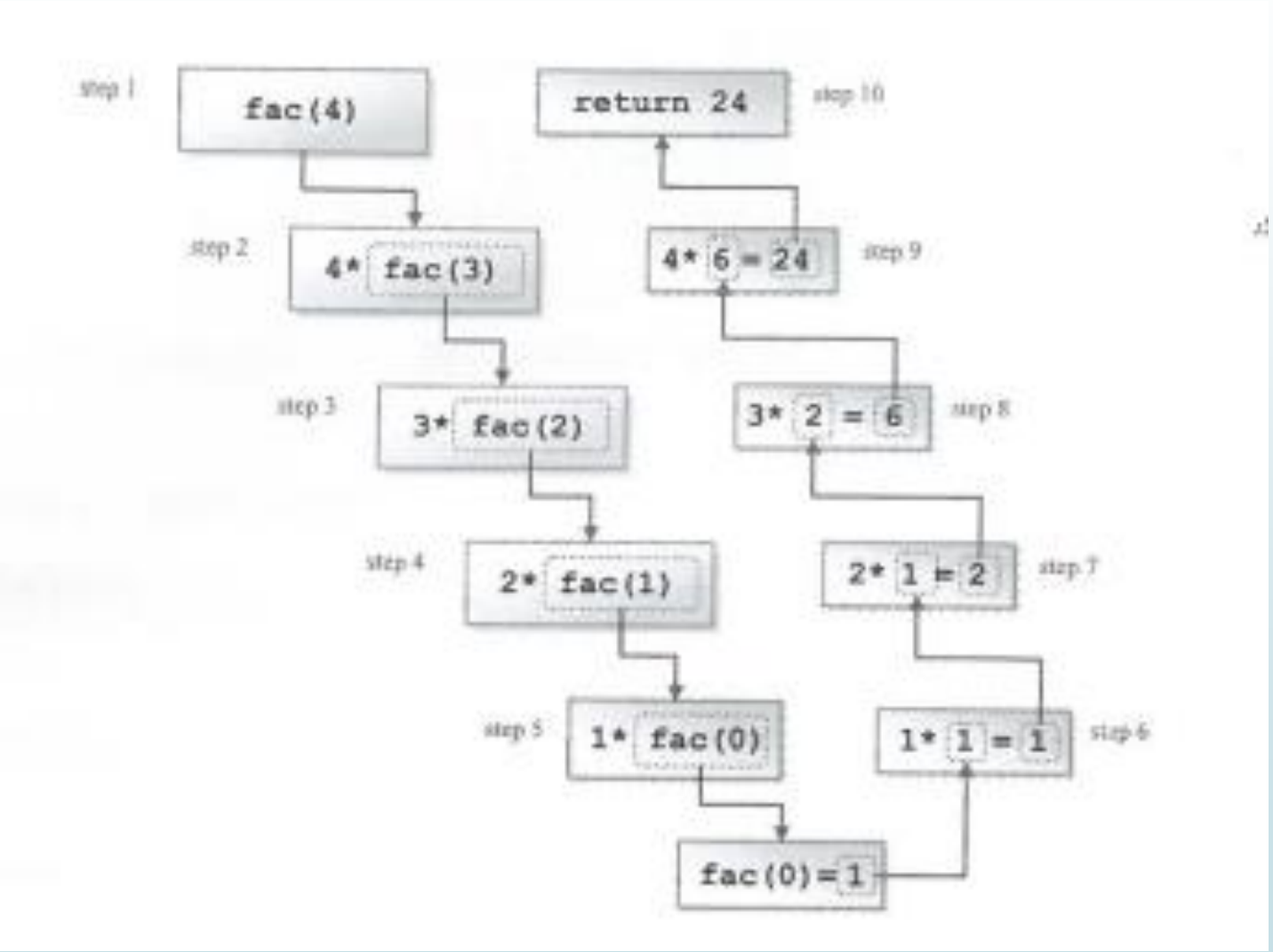
範例 - 遞迴函數 - 計算階乘

```c
#include <stdio.h>
#include <stdlib.h>
int fac(int);      /* 遞迴函數fac()的原型 */
int main(void)
{
    printf("fac(4)=%d\n", fac(4));    /* 呼叫遞迴函數fac() */

    system("pause");
    return 0;
}

int fac(int n)  /* 自訂函數fac()，計算n! */
{
    if(n>0)
        return (n*fac(n-1));
    else
        return 1;
}
```

範例 - 遞迴函數- 計算階乘圖解

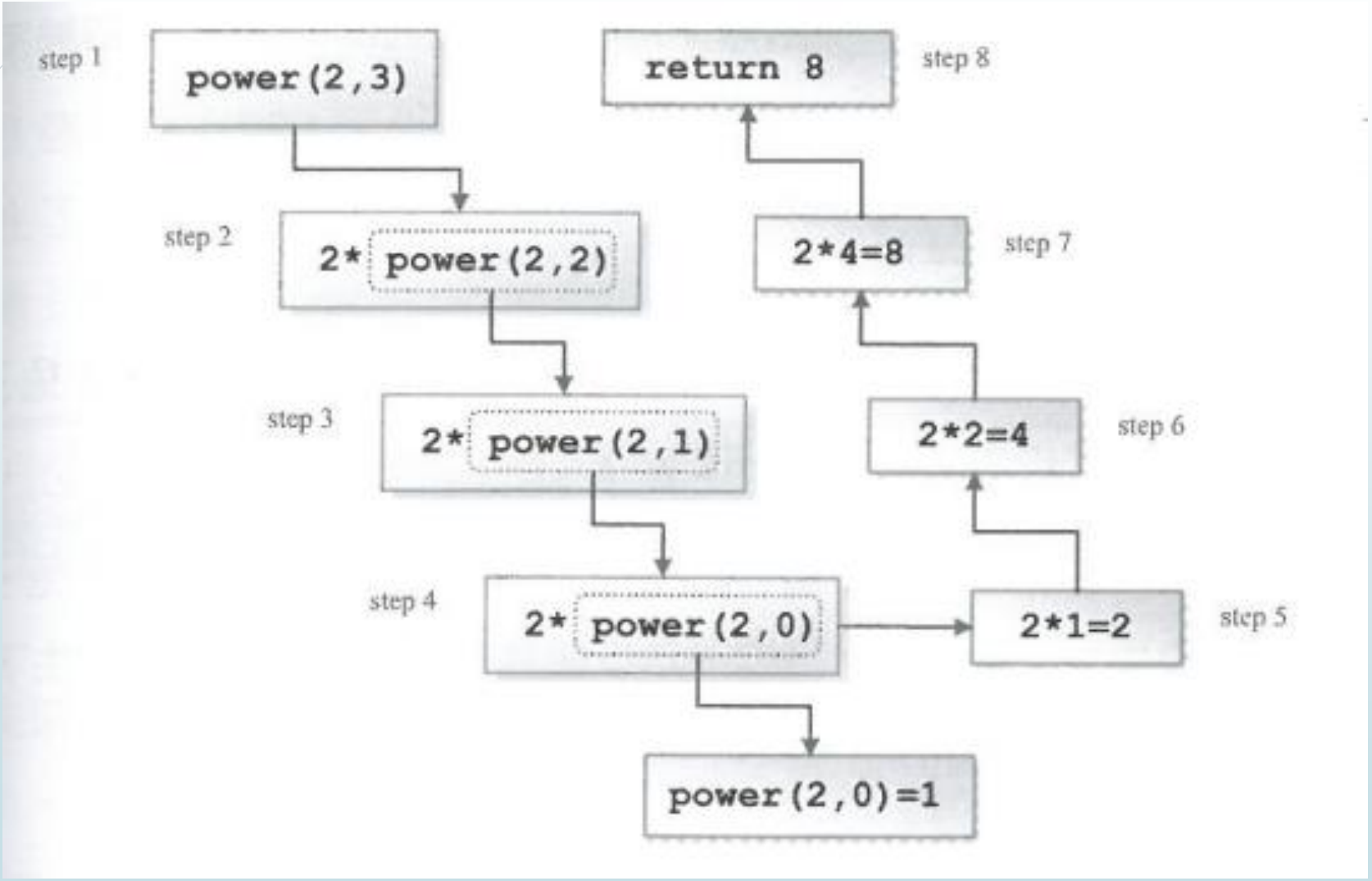


範例 - 遞迴函數 – 求次方

```c
#include <stdio.h>
#include <stdlib.h>
int power(int,int);      /* 迴函數power()的原型 */
int main(void)
{
   printf("power(2,3)=%d\n", power(2,3));
   system("pause");
   return 0;
}

int power(int b,int n)  /* 自訂函數power()，計算b的n次方 */
{
   if(n==0)
      return 1;
   else
      return (b*power(b,n-1));
}
```

範例 - 遞迴函數-求次方圖解



費氏數列

- 1202 Fibonacci 義大利數學家
- Liber Abbaci 兔子問題
  - 一開始有一對小兔子關在籠子裡
  - 一個月後，這一對兔子可長大成為成兔。長大之後，一個月就可以生一對小兔子，而且在以後每個月都會再生一對小兔子。
  - 每一對小兔子在出生後滿一個月便會變長大，再一個月後便可生下一對小兔，而且之後每個月都會再生一對小兔，請問一年以後共有多少對兔子(假設生下來的兔子都不會死)？

費氏數列-兔子問題圖解

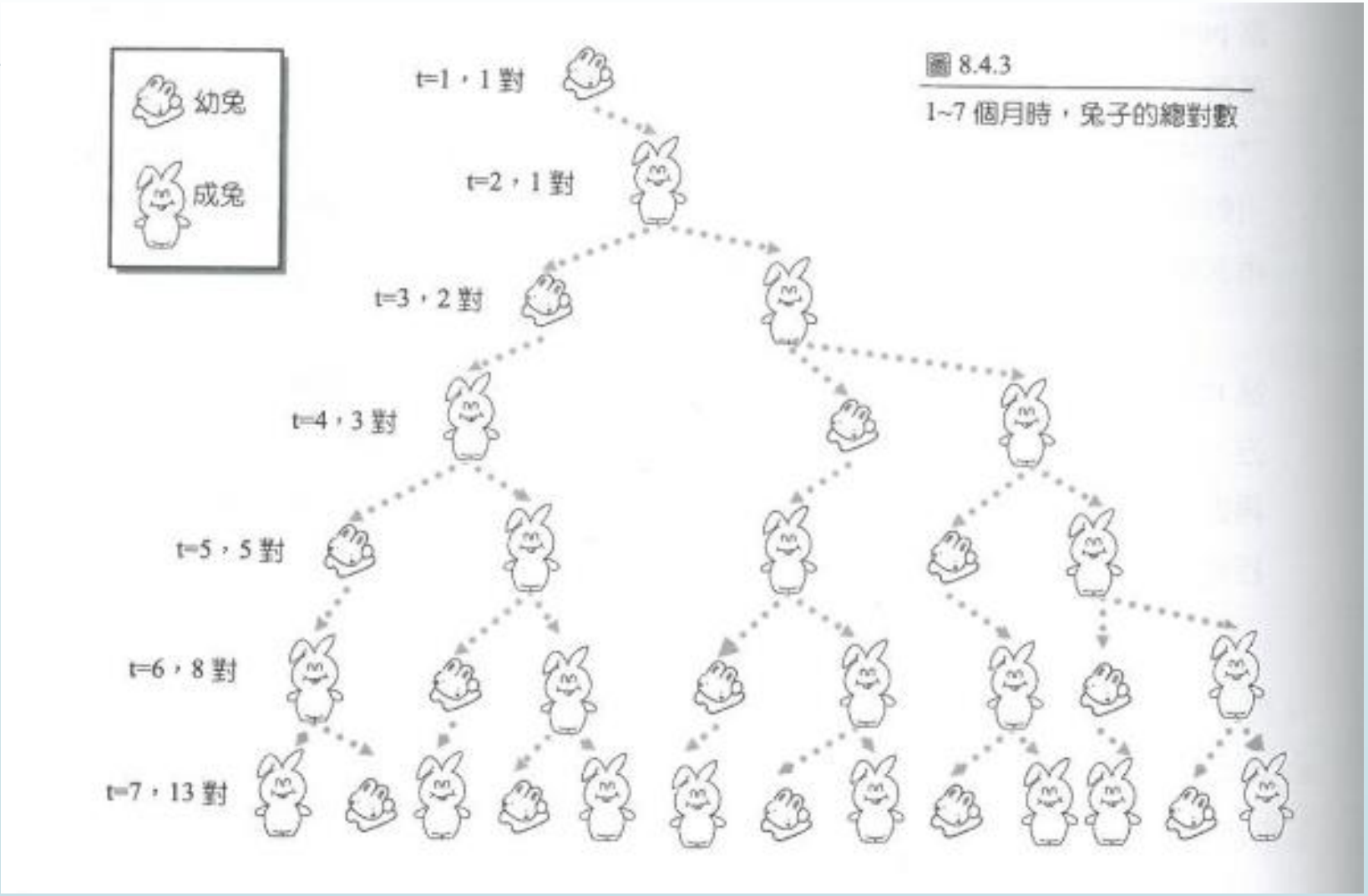


圖 8.4.3
1~7 個月時，兔子的總對數

兔子問題➔費氏數列

- 兔子問題

月份	1	2	3	4	5	6	7	8	9	10
兔子對數	1	1	2	3	5	8	13	21	34	55

- 費氏數列
  - 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , ...
  - 除了第一個與第二個數字都是1之後，其餘的數字都是前兩個數字的和
  - 公式

$$\text{fib}(n) = \begin{cases} 1; & n = 1, 2 \\ \text{fib}(n-1) + \text{fib}(n-2); & n \geq 3 \end{cases}$$

範例 - 費氏數列

```c
#include <stdio.h>
#include <stdlib.h>
int fib(int);              /* fib()函數的原型 */
int main(void)
{
    int n;
    for(n=1;n<=10;n++)             /* 計算前10個費氏數列 */
        printf("fib(%d)=%d\n",n,fib(n));

    system("pause");
    return 0;
}

int fib(int n)        /* 定義函數fib()，計算費氏數列的第n個數 */
{
    if(n==1 || n==2) /* 如果n=1或n=2，則傳回1 */
        return 1;
    else                    /* 否則傳回 fib(n-1)+fib(n-2) */
        return (fib(n-1)+fib(n-2));
}
```

範例 -費氏數列圖解

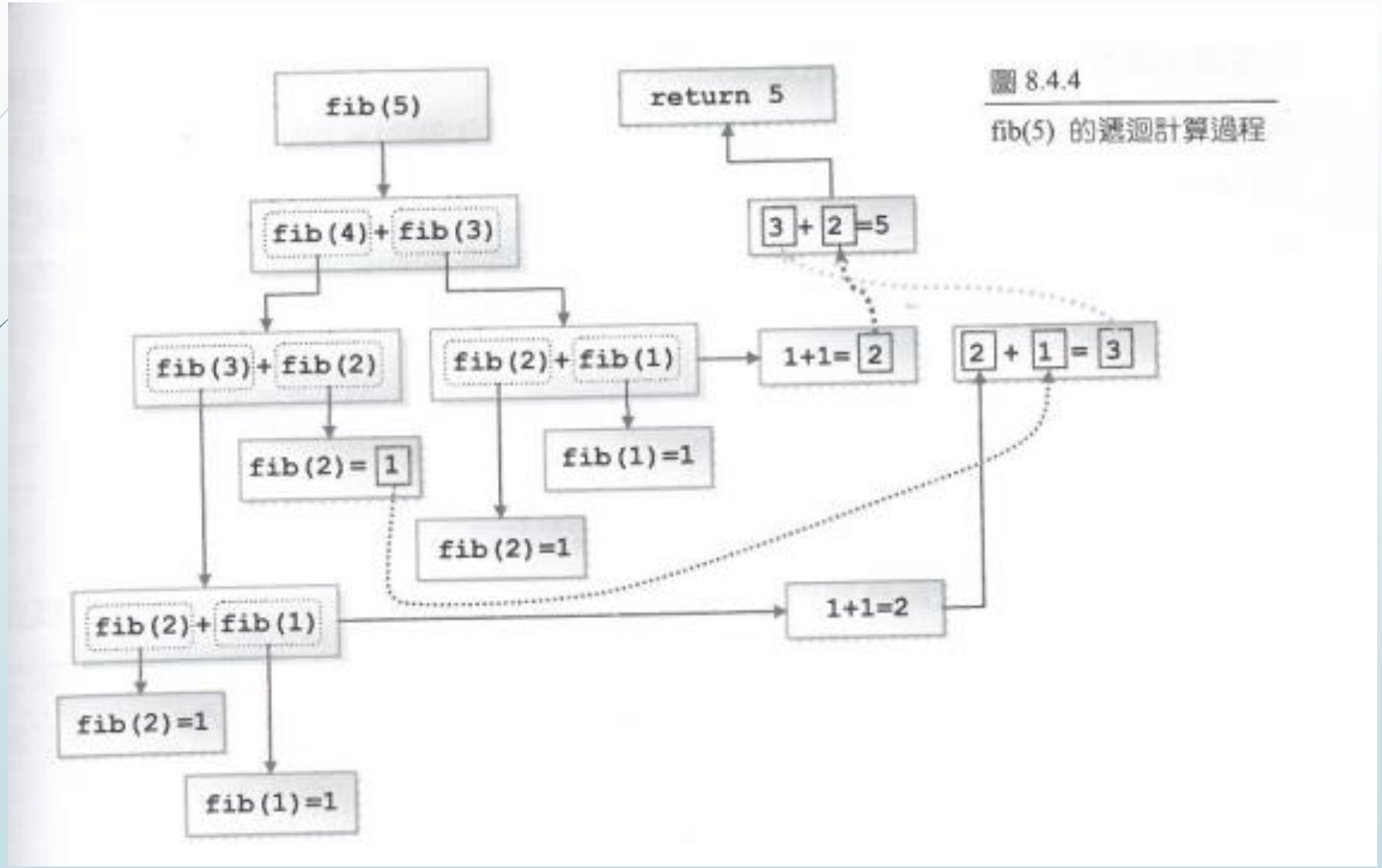


圖 8.4.4
fib(5) 的遞迴計算過程

變數

- 區域變數 (local variable)
  - main函數或是自己寫的函數內的變數皆為區域變數
  - 區域變數的生命週期(變數保留在記憶體的時間)只在於執行函數時存在，執行完函數時則會被銷毀，因此無法在其他地方做存取
- 全域變數 (global variable)
  - 將變數輸入在函數外面
- 靜態變數 (static variable)
  - static 宣告在變數型態前

```
static float num;
```

  - 和區域變數活動範圍相同
  - 配置一個固定的記憶體位置，函數執行後不會銷毀，值會被保留下來，若是再次呼叫到該函數時，則會將靜態變數存放在記憶體的值拿來使用，而非宣告的初值

範例－區域變數

```c
#include <stdio.h>
#include <stdlib.h>
int fac(int);                    /* fac()函數的原型 */
int main(void)
{
    int ans;
    ans=fac(5);
    printf("fac(5)=%d\n",ans);

    system("pause");
    return 0;
}

int fac(int n)
{
    int i, total=1;
     for(i=1; i<=n; i++)
        total=total*i;
    return total;
}
```

區域變數作用範圍

```
    int ans;
    ans=fac(5);
    printf("fac(5)=%d\n",ans);

    system("pause");
    return 0;
}
```

區域變數 ans 的活動範圍

```
int fac(int n)
{
    int i, total=1;
    for(i=1; i<=n; i++)
        total=total*i;
    return total;
}
```

區域變數 i 與 total 的活動範圍

區域變數 n 的活動範圍

範例－區域變數 2

```c
#include <stdio.h>
#include <stdlib.h>
void func(void);
int main(void)
{
    int a=100;      /* 宣告main()函數裡的區域變數a */

    printf("呼叫func()之前,a=%d\n",a);    /* 印出main()中a的值 */
    func();       /* 呼叫自訂的函數 */
    printf("呼叫func()之後,a=%d\n",a);    /* 印出a的值 */

    system("pause");
    return 0;
}


void func(void)      /* 函數func() */
{
    int a=300;          /* 宣告func()函數裡的區域變數a */
    printf("於func()函數裡,a=%d\n",a);    /* 印出func函數中a的值 */
}
```

範例 - 全域變數

```c
#include <stdio.h>
#include <stdlib.h>
void func(void);          /* 函數func()的原型 */
int a;                    /* 宣告全域變數a */
int main(void)
{
   a=100;                 /* 設定全域變數a的值為100 */
   printf("呼叫func()之前,a=%d\n",a);
   func();                /* 呼叫自訂的函數 */
   printf("呼叫func()之後,a=%d\n",a);

   system("pause");
   return 0;
}

void func(void)           /* 自訂的函數func() */
{
   a=300;                 /* 設定全域變數a的值為300 */
   printf("於func()函數裡,a=%d\n",a);
}
```

全域變數作用範圍

```
int a;                          /* 宣告全域變數 a */
int main(void)
{
   a=100;                       /* 設定全域變數 a 的值為 100 */
   printf("呼叫 func()之前,a=%d\n",a);
   func();                      /* 呼叫自訂的函數 */
   printf("呼叫 func()之後,a=%d\n",a);

   system("pause");
   return 0;
}

void func(void)                 /* 自訂的函數 func() */
{
   a=300;                       /* 設定全域變數 a 的值為 300 */
   printf("於 func()函數裡,a=%d\n",a);
}
```

全域變數 a 的活動範圍

範例 - 全域變數 2

```c
#include <stdio.h>
#include <stdlib.h>
void func(void);
int a=50;                          /* 定義全域變數a */

int main(void)
{
    int a=100;                     /* 定義區域變數a */
    printf("呼叫func()之前,a=%d\n",a);
    func();          /* 呼叫自訂的函數 */
    printf("呼叫func()之後,a=%d\n",a);

    system("pause");
    return 0;
}

void func(void)
{
    a=a+300;                       /* 這是全域變數a */
     printf("於func()函數裡,a=%d\n",a);
}
```

全域變數2作用範圍

```
int a=50;                          /* 宣告全域變數 a */

int main(void)
{
   int a=100;                      /* 宣告區域變數 a */
   printf("呼叫 func()之前,a=%d\n",a);
   func();          /* 呼叫自訂的函數 */
   printf("呼叫 func()之後,a=%d\n",a);

   system("pause");
   return 0;
}

void func(void)
{
   a=a+300;                        /* 這是全域變數 a */
   printf("於 func()函數裡,a=%d\n",a);
}
```

全域變數 a 的活動範圍

區域變數 a 的活動範圍

全域變數 a 的活動範圍

範例 - 全域變數 3

```c
#include <stdio.h>
#include <stdlib.h>
double pi=3.14;                    /* 宣告全域變數pi */
void peri(double);
void area(double);
int main(void)
{
   double r=1.0;
   printf("pi=%.2f\n",pi);         /* 於main()裡使用全域變數 pi*/
   printf("radius=%.2f\n",r);
   peri(r);      /* 呼叫自訂的函數 */
   area(r);

   system("pause");
   return 0;
}

void peri(double r) /* 自訂的函數peri()，印出圓周 */
{
   printf("圓周長=%.2f\n",2*pi*r);  /* 於peri()裡使用全域變數pi */
}

void area(double r) /* 自訂的函數area()，印出圓面積 */
{
   printf("圓面積=%.2f\n",pi*r*r);  /* 於area()裡使用全域變數pi */
}
```

範例－靜態變數

```c
#include <stdio.h>
#include <stdlib.h>
void func(void);          /* 宣告func()函數的原型 */
int main(void)
{
    func();               /* 呼叫函數func() */
    func();               /* 呼叫函數func() */
    func();               /* 呼叫函數func() */

    system("pause");
    return 0;
}

void func(void)
{
    static int a=100;                    /* 宣告靜態變數a */
    printf("In func(),a=%d\n",a);       /* 印出func()函數中a的值 */
    a+=200;
}
```

引數傳遞機制

- 傳值(call by value)
  - 在引數傳遞時，編譯器會將欲傳入函數的引數複製一份，供呼叫的函數使用，因此不管在被呼叫函數內如何改變這個傳進來的引數值，都不會改到原先變數的值

範例 -引數傳遞機制

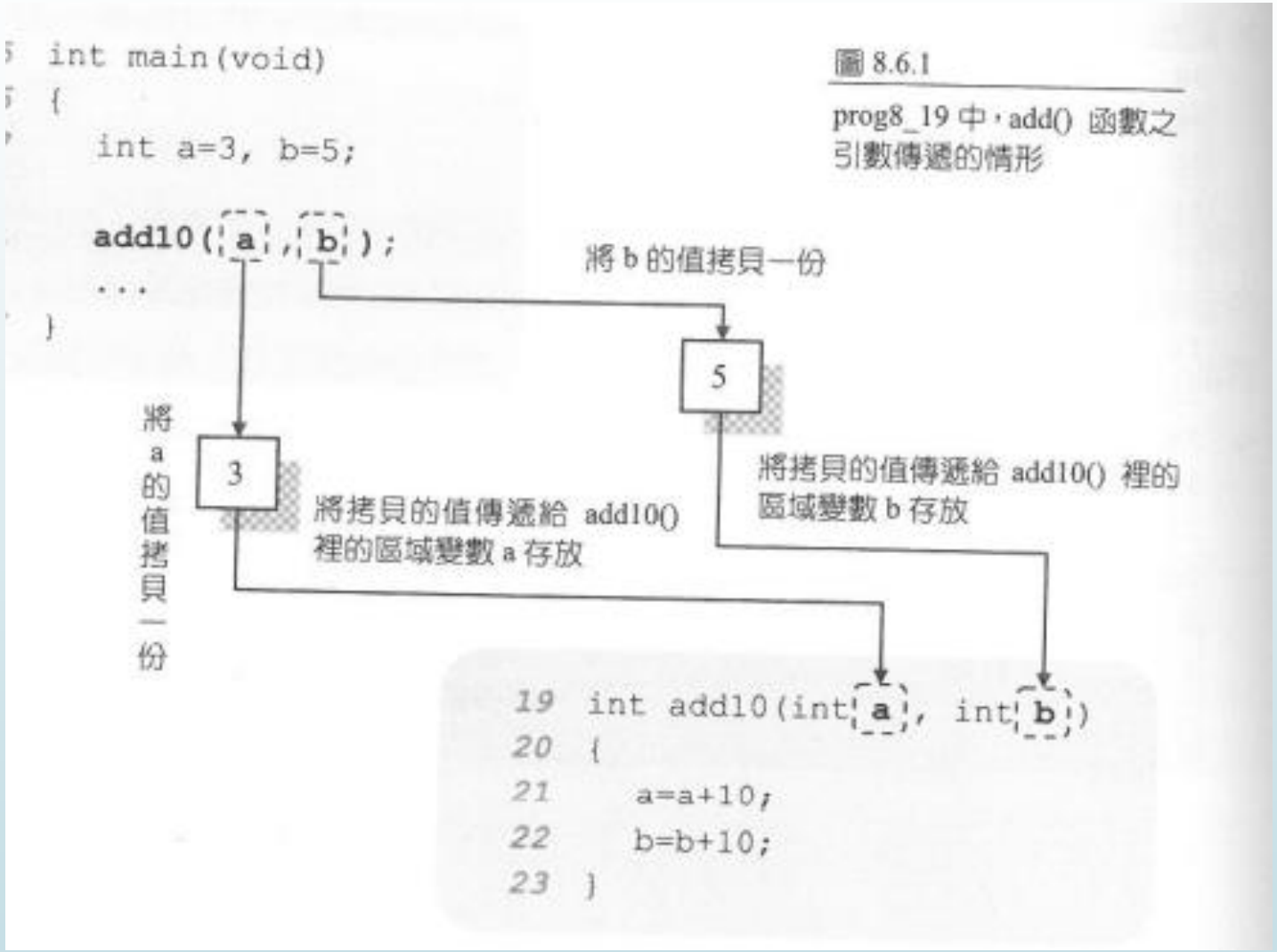
```c
#include <stdio.h>
#include <stdlib.h>
void add10(int,int);                    /* add10()的原型 */
int main(void)
{
    int a=3, b=5;                       /* 宣告區域變數a與b */

    printf("呼叫函數add10()之前: ");
    printf("a=%d, b=%d\n",a,b);         /* 印出a、b的值 */
    add10(a,b);
    printf("呼叫函數add10()之後: ");
    printf("a=%d, b=%d\n",a,b);         /* 印出a、b的值 */

    system("pause");
    return 0;
}

void add10(int a,int b)
{
    a=a+10;                 /* 將變數a的值加10之後，設回給a */
    b=b+10;                 /* 將變數b的值加10之後，設回給b */
}
```

引數傳遞機制情形


int main(void)
{
int a=3, b=5;
add10(a,b);
...
}
圖 8.6.1
prog8_19 中，add() 函數之引數傳遞的情形
將 b 的值拷貝一份
5
將 a 的值拷貝一份
3
將拷貝的值傳遞給 add10() 裡的區域變數 a 存放
將拷貝的值傳遞給 add10() 裡的區域變數 b 存放
19 int add10(int a, int b)
20 {
21 a=a+10;
22 b=b+10;
23 }

前置處理器#define

- #前置處理器
  - 這些指令在編譯之前會先被進行處理，再把處理後的結果與程式碼一起送給編譯器編譯
- #define
  - 可方便的將常用的常數，字串替換成一個自訂的識別名稱，也可取代簡單的函數
  - 變數必須佔用記憶體空間，執行時必須從記憶體取值，而#define則再編譯前以常數置換，故較節省記憶體空間
  - 語法

範例 - #define

```c
#include <stdio.h>
#include <stdlib.h>
#define BEGIN {                 /* 定義識別名稱BEGIN為左大括號{ */
#define END }                   /* 定義識別名稱END為右大括號} */
int main(void)
BEGIN                           /* 此行的BEGIN相當於左大括號 { */
   int i,j;

   for(i=1;i<=5;i++)
   BEGIN                        /* 此行的BEGIN相當於左大括號 { */
      for(j=1;j<=i;j++)
         printf("*");
      printf("\n");
   END                          /* 此行的END相當於右大括號 } */

   system("pause");
   return 0;
END                             /* 此行的END相當於右大括號 } */
```

範例 - #define 2

```c
#include <stdio.h>
#include <stdlib.h>
#define WORD "Think of all the things \
we've shared and seen.\n"
int main(void)
{
    printf(WORD);

    system("pause");
    return 0;
}
```

範例 - #define 3

```c
#include <stdio.h>
#include <stdlib.h>
#define PI 3.14             /* 定義PI為3.14 */
double area(double);
int main(void)
{
    printf("PI=%4.2f, area()=%6.2f\n",PI,area(2.0));

    system("pause");
    return 0;
}

double area(double r)
{
    return PI*r*r;
}
```

範例 - 使用const修飾子

```c
#include <stdio.h>
#include <stdlib.h>
const double pi=3.14;          /* 宣告pi為double型態的常數 */
double area(double);
int main(void)
{
   /* 若在此處設定pi=3.1416，則編譯時會發生錯誤 */
   printf("pi=%4.2f, area()=%6.2f\n",pi,area(2.0));

   system("pause");
   return 0;
}

double area(double r)
{
   return pi*r*r;
}
```

範例 - #define取代簡單的函數

```c
#include <stdio.h>
#include <stdlib.h>
#define SQUARE n*n        /* 定義巨集SQUARE為n*n */
int main(void)
{
    int n;
    printf("Input an integer:");
    scanf("%d",&n);
    printf("%d*%d=%d\n",n,n,SQUARE);  /* 計算並印出n的平方 */

    system("pause");
    return 0;
}
```

範例 - 使用有引數的巨集

```c
#include <stdio.h>
#include <stdlib.h>
#define SQUARE(X) X*X    /* 定義巨集SQUARE(X)為X*X */
int main(void)
{
    int n;
    printf("Input an integer:");
    scanf("%d",&n);
    printf("%d*%d=%d\n",n,n,SQUARE(n));   /* 計算並印出n的平方 */

    system("pause");
    return 0;
}
```

範例 - 使用巨集常見的錯誤

```c
#include <stdio.h>
#include <stdlib.h>
#define SQUARE(X) X*X              /* 定義巨集SQUARE(X)為X*X */
int main(void)
{
    int n;
    printf("Input an integer:");
    scanf("%d",&n);
    printf("%d*%d=%d\n",n+1,n+1,SQUARE(n+1));     /* 印出n+1的平方 */

    system("pause");
    return 0;
}
```

詳解使用巨集常見的錯誤

- 將巨集替換後的程式碼

```
printf("%d*%d=%d\n",n+1,n+1, n+1*n+1);
```

- n 的值為12

```
n+1*n+1=n+(1*n)+1=12+(1*12)+1=25
```

- 前置處理器並不會先行計算引數內的值，而是直接將引數傳到巨集後再於程式中替換，所以會造成乘法的優先權高於加法，得到的結果將不正確
- 解決此問題只需要加上括號即可

```
#define SQUARE(X)  (X)*(X)
```

函數 VS 巨集

- 巨集
  - 不需要像函數一樣要宣告、定義回傳值及引數的型態
  - 處理的只是字串，前置處理器將「代換標記」的內容直接替代掉再程式碼出現的識別名稱
  - 不用特別考慮變數型態的問題，可以處理變數、浮點數、各種型態及替代簡單的函數
  - 巨集佔用的記憶體較多，但是程式的執行流程不用移轉，因而程式執行速度較快；函數程式碼較短，佔用的記憶體較少，但是程式的執行流程必須交給函數使用，所以執行速度較慢

習題

- 試寫int mod(int x,int y)函數，計算x/y的餘數，並利用此函數來計算mod(17,5)即計算17/5的餘數。

習題

- 試寫一函數 int prime(int n)，可以用來找出第n個質數(第一個質數為2，第二個質數為3，以此類推)，以此函數找出第100個質數。