國立屏東大學 資訊工程學系 程式設計(一)

12. 函式與函式庫

函式(function)可視為一個“ 小程式 ”，它可以接收輸入、進行特定的處理並且輸出資料。如同程式可以使用IPO模型分析，“ 小程式 ”(也就是函式)當然也可以用IPO模型加以分析，請參考figure 1

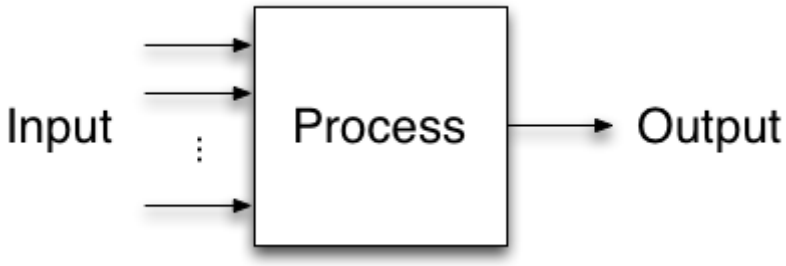

Fig. 1
要特別注意的是：「一個函式可以有0個或多個輸入，但只能有0個或一個輸出。」，我們將函式的輸入稱為「參數(parameter)」將其輸出稱為傳回值(return value)

<note tip>C語言的函式在某方面來說與數學的函數十分類似，且兩者的英文皆為function本書針對在數學與C語言，將function一詞分別譯做函數與函式，以視區別</note>

12.1 函式定義

函式在被使用之前必須先加以定義，其定義語法如下：

```
return_type functionName ( inputParamemters )
{
   declarations
   statements
}
```

每個函式必須提供其傳回值的型態定義，所謂的傳回值即為IPO模型中之Output部份。我們以return_type是定義函式的傳回值的資料型態，有以下的定義規則：

- 一個函式至多只能有一個輸出，除了不能傳回陣列外，所有資料型態皆可。
- 若沒有要傳回的值(沒有輸出)，則return_type必須寫做void表示無型態。

接著functionName是定義函式的名稱，建議依函式的功能使用較具意義的名稱，以增進程式的可讀性(readability)函式名稱如同變數名稱一樣，在語法結構上都是「識別字」，其命名規則如下：

- 只能使用英文大小寫字母、數字與底線(_)

- C語言是case-sensitive的語言，意即大小寫會被視為不同的字元
- 不能使用數字開頭
- 不能與C語言的保留字相同

在函式名稱後以一組小括號「()」來定義輸入(有時亦稱為傳入)的參數，也就是IPO中的Input部份，其語法定義如下：

```
[type variableName]?[,type variableName]*
```

<note tip> 在上面的語法說明中，「[]」為選擇性的語法單元，其後接續「*」表示該語法單元可出現0次或多次；「?」表示出現0次或1次。另外還有「+」代表1次或多次 </note>

- 函式可以有0個或多個輸入參數
- 若超過一個以上的參數，則任意兩個參數間必須要使用「,」隔開
- 與變數宣告相同，每個輸入參數必須定義其名稱與其所屬之資料型態。
  - 參數名稱亦為識別字，同識別字命名規則
  - 參數的資料型態並無限制
- 參數的值於使用函式(函式被呼叫)時傳入，可在函式內部的處理敘述中使用

<note important>在變數宣告時我們可以使用int i,j;一次宣告兩個整數，但在函式的參數定義時，不可以使用int i, j這種方式。如果需要傳入兩個整數型態的參數，那麼您必須分別宣告，如：

```
void foo(int i, int j)
{
}
```

是正確的定義；但下面這個則是錯誤的定義:

```
void foo(int i,j)
{
}
```

</note>

最後，在一組大括號「{}」內定義函式的處理敘述，也就是IPO模型中的Process部份。由於使用了大括號「{}」，因此此部份又被稱為是函式內容區塊。可分成兩部份，首先是宣告的部份，您可以在此為函式的處理宣告所須的變數，其規則與一般變數宣告一致；接著才是處理敘述的部份，您可以使用任意的C語言敘述。有以下的規則：

- 若有宣告需求，則declarations可以省略
- 若無處理需求且函式無傳回值(return_type定義為void)則statements亦可省略
- 若函式有傳回值，則其statements中至少須包含一行return敘述
  - return敘述語法為return expression;expression的運算結果其型態必須與return_type相同

以下是一個典型的函式範例：

```
int sum( int x, int y)
```

```
{
   int result;
   result = x+y;
   return result;
}

int main()
{
    int a, b;
    a=10;
    b=45;
    int x=33,y=77;

    sum(a,b);
    sum(x,y);
    sum(a,sum(a,b));
    sum(3,5);
}
```

以這個函式為例，其functionName為sum也就是加總的意思，其傳回值型態定義為int傳入的參數為兩個整數，分別命名為x與y此函式內容先定義了所需的變數result然後再計算x+y的值並以return敘述將結果傳回。

以下是一個印出程式資訊的函式定義範例

```
void showInfo()
{
   printf("This program is written by Jun Wu.\n");
   printf("All right reserved.\n");
}
```

12.2 main()函式

打從我們一開始學習C語言，我們就已經開始使用函式定義了，請回想一開始的hello.c

h hello.c

```
/* This is my first C program */
#include <stdio.h>

int main()
{
     printf("Hello World!\n");
```

```
}
```

在這個程式中，我們已經自己定義了一個函式 – main()不過main()函式是一個特別的函式，它是程式的進入點，同時它有以下的特別規則：

- main()函式的傳回值可以為int或void
  - 傳回值定義為int時：
    - 可使用return敘述傳回整數以說明程式執行狀態
    - return敘述可以省略，此時會傳回預設值0
  - 傳回值定義為void時：
    - 不可使用return敘述
    - main()函式還是會傳回預設的傳回值0

參考上述規則，以下的main()函式都是正確的:

```
int main()
{
}
```

```
int main()
{
   return 0;
}
```

```
void main()
{
}
```

但下面這個main()函式是錯誤的

```
void main()
{
   return 0;
}
```

每當程式被系統載入加以執行時main()函式是程式首先也是唯一會被加以執行的程式區塊，一旦執行完main()函式的內容區塊，程式最會結束。關於main()函式的傳回值，是回傳給誰呢？由於我們是在console模式下執行程式，所以可以在Console模式下取得main()函式的傳回值，請參考下面的例子：

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

```
int main()
{
   printf("Hello!\n");
   return 1;
}
```

當程式執行完成後，main()函式的傳回值會被存放在系統的環境變數中，您可以使用以下的指令取得其值：

```
[16:41 user@ws example] ./a.out
Hello!
[16:41 user@ws example] echo $?
1
[16:41 user@ws example]
```

還要注意，系統預設取回的main()函式傳回值為一個8bits的unsigned int，若傳回值為超出此範圍時，數值會有溢位或不正確的問題。

12.3 函式呼叫

先讓我們回顧一下，本書到目前為止，已為您介紹過的函式，如table 1:

函式名稱	標頭檔
printf()	stdio.h
scanf()	stdio.h
sizeof()	stdio.h
getchar()	stdio.h
putchar()	stdio.h
srand()	stdlib.h
rand()	stdlib.h
time()	time.h

Tab. 1: 已使用過的函式列表

回想看看您是如何使用這些別人幫您寫好(C語言內建)的函式呢？其實函式的使用方法很簡單，只要在程式敘述中使用該函式的名稱，並在一組大括號「()」內傳入引數(arguments)給函式使用即可，要注意的是引數除了是值外，也可以是運算式。

<note tip>

Parameters vs.Arguments (到底是參數還是引數?)

所謂參數(parameters)是定義函式時所指定的輸入變數，引數則是指在呼叫使用函式時所傳入的值或運算式。

</note>

請參考下面的例子：

```
#include <stdio.h>

int sum( int x, int y )
{
   return x+y;
}

int main()
{
   int i=2, j=4, k=6;

   //使用=將函式的傳回值指定給變數i
   i = sum(10, 30);
   printf("10+30 = %d\n", i);

   //直接把函式的傳回值當成一般的值, 並傳給printf做為輸入
   printf("5+6 = %d\n", sum(5,6));

   //引數的部份也可以為變數
   printf("5+j = %d\n", sum(5,j));

   //傳回值可做為運算式的一部份
   printf("5+j+k = %d\n", 5+sum(j,k) );

   //引數的部份也可以為運算式
   printf("5+j+k = %d\n", sum( 5 , (j+k) ) );

   //引數的部份也可以為另一個函式呼叫
   printf("5+j+k = %d\n", sum( 5 , sum(j,k) ) );
}
```

函式呼叫時的引數也可以為陣列，例如下面的例子：

```
void showData(int data[])
{
   int i;
   for(i=0; i<(sizeof(data)/sizeof(data[0]));i++)
      printf("%d ", data[i]);
}

int main()
{
   int data[5]={2, 4, 6, 8, 10};
   showData(data);
}
```

12.4 變數範圍

只可以在其所屬的程式區塊內使用的變數稱為區域變數(local variables)，請參考下例，我們標示了每個區塊可使用的變數：

```
int foo(int i, int j)
{
 //這裡可以使用的變數為i,j
   return i+j;
}

int main()
{
 //這裡可以使用的變數為x,y
   int x=3, y=5;

   printf("%d\n", foo(x,y));
}
```

若在不同區塊內，有同樣的變數名稱，則以該變數所在的區塊為依據，請參考下例：

```
int foo(int x, int y)
{
 // 這裡可以使用的變數為x,y
 // 可以視為 foo.x與foo.y
   if(x>y)
      return x;
   else
      return y;
}

int main()
{
 // 這裡可以使用的變數為x,y
 // 可以視為 main.x 與main.y
   int x=3, y=5;

   printf("%d\n", foo(9,8));
}
```

宣告在函式外的變數稱為全域變數(global variables)，可以在程式的任何地方使用。但若某區塊內有同樣名稱的變數，則以該區塊為主。請參考下面的例子：

```
// 可以視為global.x與global.y

int x,y;

int foo(int x, int y)
{
 // 這裡可以使用的變數為x,y -> foo.x, foo.y

   if(x>y)
      return x;
   else
      return y;
}

int foo2(int i, int j)
{
 // 這裡可以使用的變數為x,y,i,j -> global.x, global.y, foo2.i, foo2.j

   if((i>x)&&(j>y))
      return i+j;
   else
      return x+y;
}


int main()
{
 // 這裡可以使用的變數為x,y -> global.x與global.y
   printf("%d\n", foo(9,8));
   printf("%d\n", foo2(3,6));
}
```

12.5 遞迴呼叫

以下是一個計算N!的函式定義範例

```
int factorial( int n)
{
   int result=1, i;

   if((n==0)||(n==1))
      return 1;
   else if (n>1)
   {
      for(i=2;i<=n;i++)
      {
         result*=i;
      }
```

```
        return result;
    }
    else
        return (-1);
}
```

以這個函式為例，其functionName為factorial也就是階乘的意思，其傳回值型態定義為int傳入的參數為整數，命名為n此函式內容先定義了所需的變數result與i然後再計算n!並以return敘述將結果傳回。其中若參數n的值為0或1時，依階乘定義0!與1!皆為1，使用return敘述傳回1。要注意的是一旦使用了return敘述，函式就會將指定的結果傳回，剩餘未執行的程式碼不會被執行。若n的值不為0或1，則比較n是否大於等於2，使用一個迴圈來計算n!並使用return敘述將結果傳回。若n的值不為0或1且不大於等於2，那麼唯一的可能是n為負數，此時傳回-1表示錯誤。
這個計算階乘的程式，可以遞迴的方式重新設計。所謂的遞迴(recursvie)指得是在函式的定義中呼叫自己。例如：

在一個遙遠的地方，有一座山上，山頂有一間破廟，廟裡有一個小和尚，他做了一個夢，夢到「

> 在一個遙遠的地方，有一座山上，山頂有一間破廟，廟裡有一個小和尚，他做了一個夢，夢到「
> > 在一個遙遠的地方，有一座山上，山頂有一間破廟，廟裡有一個小和尚，他做了一個夢，夢到「
> > > 在一個遙遠的地方，有一座山上，山頂有一間破廟，廟裡有一個小和尚，他做了一個夢，夢到「
> > > > 在一個遙遠的地方，有一座山上，山頂有一間破廟，廟裡有一個小和尚，他做了一個夢，夢到「
> > > > > 在一個遙遠的地方，有一座山上，山頂有一間破廟，廟裡有一個小和尚，他做了一個夢，夢到「
> > > > > > 在一個遙遠的地方，有一座山上，山頂有一間破廟，廟裡有一個小和尚，他做了一個夢，夢到「
> > > > > > > ...
> > > > > >
> > > > >
> > > > >
> > > >
> > > >
> > >
> > >
> >
> >
>
>

OK這就叫做遞迴。讓我們使用遞迴重新改寫計算階乘的程式：

```
int factorial(int i)
{
   if((i==0)||(i==1))
      return 1;
   else
      return i*factorial(i-1);
}
```

以計算5的階乘為例，呼叫factorial(5)的過程如下：

factorial(5) →

> 5 * factorial(4) →
>
> > 5 * 4 * factorial(3) →
> >
> > > 5 * 4 * 3 * factorial(2) →
> > >
> > > > 5 * 4 * 3 * 2 * facotrial(1) →
> > > >
> > > > > 5 * 4 * 3 * 2 * 1

12.6 函式原型與標頭檔

若是我們自行設計的函式也是一樣的使用方式，但如同變數的使用一樣，必須在使用前先進行過變數的宣告；函式的使用，一般稱之為函式呼叫(function call)在您可以呼叫一個函式前，您必須先提供該函式的定義。通常必須在程式碼的開頭處進行函式的定義，例如：

```
#include <stdio.h>

void foo()
{
   printf("This is foo.\n");
}

int main()
{
   foo();
   printf("This is main function.\n");
}
```

程式當然還是從main()函式開始執行，在main()函式的內容區塊中，我們使用了一個自行定義的函式foo()為了讓在main()函式內容區塊中，能正確地呼叫foo函式，我們將其函式定義在main()函式之前。但這種做法要求所有要使用到的函式都必須定義在使用之前，有時我們並不想這麼做，那麼你可以在程式開頭處先宣告函式的原型(prototype)然後在其它地方提供完整的函式定義，請參考下面的例子：

```
#include <stdio.h>

void foo();

int main()
{
   foo();
   printf("This is main function.\n");
}

void foo()
{
   printf("This is foo.\n");
}
```

在這個例子中，我們將函式定義在main()函式之後，並且為了讓main()函式內容區塊還是能呼叫foo函式，必須在main()函式前提供foo()函式的原型(prototype)說明。函式prototype的宣告語法如下：

```
return_type functionName ( inputParamemters );
```

與函式定義語法相同，只不過少了函式的內容區塊而已，並且要在結尾處加上一個「;」分號。

<note tip> 所謂的prototype是指函式的定義，但不包含其內容區塊。從prototype中，我們可以知道函式的名稱、輸入參數的個數與型態、還有傳回值的型態為何，這些資訊被稱為是函式的介面(interface)，而這些資訊已經足夠讓我們呼叫這個函式。至於函式的內容區塊，則是函式實際進行處理的地方，也就是說內容區塊決定了函式所提供的功能為何？相較於函式的介面，內容區塊被稱為函式的實作(implement)。
</note>

如果一個或一個以上的函式，常常會被不同程式使用，那麼將它的函式定義在所有使用到的程式中，應該是一個很糟的決定，不但麻煩而且日後很難維護。常見的做法是，將函式另外定義在獨立的程式檔案中，並且提供關於函式原型宣告的標頭檔案(header file)，日後使用函式的人，只要在其程式碼中使用#include指令來載入標頭檔，然後在編譯時將函式定義所在的檔案一起納入編譯，如此即可完成程式的開發。請參考以下的範例：

h sum.h

```
int sum(int x, int y);
```

h sum.c

```
int sum(int x, int y)
{
   return x+y;
```

```
}
```

h aTest.c

```
#include <stdio.h>
#include "sum.h"

int main()
{
   printf("3+5=%d\n", sum(3,5));
}
```

請分別使用以下指令來編譯與執行：

```
[12:21 user@ws example] cc -c sum.c -o sum.o // 編譯產生不可單獨執行的sum.o目的
檔
[12:21 user@ws example] cc aTest.c sum.o     // 結合sum.o一起編譯產生可執行檔
[12:21 user@ws example] ./a.out
3+5=8
[12:21 user@ws example]
```

如果您的工作是專責開發函式，供其它程式設計師使用，那麼您有可能不想公開函式的實作細節。此時您只要提供sum.h與sum.o給其它程式設計師，他們就可以使用您所設計的函式而不知道其實作的方法。

12.7 函式庫

延續上一節的討論，您可以開發一些函式供別人使用。以下提供更進一步的做法：

假設除了sum.o之外，您還有easyPrint.o目的檔

- 將它們合併成一個靜態連結函式庫myLib.a

```
[12:22 user@ws example] ar -r myLib.a sum.o easyPrint.o
```

- 將它們合併成一個動態連結函式庫myLib.so

```
[12:22 user@ws example] gcc -Wall -fpic -shared sum.c easyPrint.c -o
myLib.so
```

這樣一來，你只要提供標頭檔與myLib.a或myLib.so給其它人。假設現在有一個程式main.c使用到了您所設計的函式，其編譯方法如下：

```
[12:22 user@ws example] cc main.c myLib.a
[12:22 user@ws example] cc main.c myLib.so
```