第三節 技術主題 2 ・ 函數開發

3-1. 進行函數的宣告、定義、引用

所謂「函數 ・ Function」即是將一群具有"相關動向"的程式碼，採用「集中設計、彈性應用」於功能開發的進階技術語法。其作業原理是以配置"參考型式"訂作函數類型及識別名稱，再利用"關係參數"進行資料互動存取及更新。其過程中先將"功能執行碼"編寫於自訂申請的"函數別名"內部，再依作業時機需求進行線上"呼叫引用"。如果是「單向・無回傳」執行狀態，通常採用「程序模組」類型進行開發，但要「雙向・具回傳」互動狀態時，就要使用「功能函式」類型進行設計。依照不同「設計動向」，規劃出控制句型語法，參考如下...

㋮巨匠材哥・線上速解

㋮ 單向・無回傳.
```
void 函數別名(型式 參數,.....) {
    程式碼
}
```

㋮ 雙向・具回傳.
```
type 函數別名(型式 參數,.....) {
    程式碼
    retrun value;
}
```

最後要注意的是在使用「新型字串・String」及「原型指標・*」這兩種技術型式。尤其是在 Dev-C++ 5.11 這個創新版本工具中，進行編譯時就會出現警告訊息，希望可把 char* 轉換成 string 型別提示，當然這並不代表指標技術就此被取代而不再重要，我們應該將其看成更具人性化而發明的新式感觀作法，而指標的應用依舊是快捷更新資料的技術選擇！

3-2. 原型與指定型式應用句型比較

通常在「自訂函數」應用規則中，使用前都要進行「註冊宣告」行為，也就是所謂的「全型宣告」作業機制。因為這些開發介面是由設計師自行創作，無法由系統內建支援及引用預訂技術標題來執行，只有經過註冊型式空間行為，讓電腦經由宣告動作所預留的位址中找到線上執行目標，這樣子就能維持正常功能運作。所以執行註冊有其重要性，唯一例外的是如果將執行介面放在呼叫者之前，就可以省略此宣告動作，這也是現代人比較喜歡的示範設計方式，如要使用其行為表述式，參考如下...

㋮ 註冊・宣告式.
```
型式 函數別名(型式 參數,.....);
```

最後依照不同「呼叫動作」，規劃出控制句型語法，參考如下...

㋮ 單向・無回傳.
```
函數別名(指示值/參照址,.....);
```

㋮ 雙向・具回傳.
```
type 目標別名 =函數別名(指示值/參照址,.....);
```

3-3. 區域、全域、靜態宣告時機與動機

當我們在「自訂函數」與「呼叫動作」設計中進行資料傳送時，會先分析其過程作業關係變化，再決定使用何種宣告(declare)方式來申請記憶體空間最為恰當，也是業界所指的「生命週期」應用原理。首先介紹「區域宣告・Local」的用法及特性，這是最基本也最常見的設計方式，其目標只能在"單一功能介面"裡使用，隸屬電腦自動化機制，進入即時取得存取資料位址，離開自動釋放應用空間。再談「全域宣告・Global」為人工整體化申請機制，可支援整個檔案下的"所有設計介面"，通常放置於引用技術區塊中，會在所有執行介面應用前產生其作業空間值，直到整個程式終止才會釋放記憶體資源。最後要討論「靜態宣告・Static」為人工區域化申請機制，其功能在於保留區域中的重點資料，使該目標值不受自動譯放行為影響。

3-4. 參數傳送方法

單純呼叫「自訂函數」而不傳送任何資料時，就會省略參數(argument)架設。當我們要將資料送出到指定函數應用時，就要加設控制參數列。通常在呼叫端提供資料實值，通稱「實際參數」關係，而在設計端則負責對應接收工作，亦稱「虛擬參數」關係。進行參數傳送方法有「傳值呼叫・By Value」、「傳參考值呼叫・By Reference」及「傳址呼叫・By Address」等三種線上作法，參考如下...

- 傳值呼叫. | (值, ...) → (值型式 目標, ...)
  此種作法，將保留來源值，只是將值單向送出。

- 傳參考值呼叫. | (值, ...) → (值型式& 目標, ...)
  此種作法，將變更來源值，利用取址型式將值連結送出。

- 傳址呼叫. | (&值, ...) → (值型式 *目標, ...)
  此種作法，將變更來源值，利用資料取址將值連結送出。

到目前為止我們也掌握了許多函數資訊，再來做個範例體驗，把上述的技術語法整合在設計上吧！請各位完成下列專案，將程式碼內容對照本章前 4 節所說的應用關係變化吧！

【專案代號：Study3_01】

*執行畫面・線上研討...


```
--------【傳值作法】・線上回傳 ------
 取得參考原始值 -- a = 10 , b = 15
                   a + b = 25
---【傳參考值作法】・線上更新 ------
 參照型址異動後 -- a = 20 , b = 5
                   a - b = 15
--------【傳址作法】・線上更新 ------
 指標物件異動後 -- a = 3 , b = -25
                   a * b = -75
請按任意鍵繼續 . . .
```

3-5. 技術週邊研討

當然身為期待成為「Super Designer」族群的各位，除了已擁有基本函數開發能力外，還要朝向更高境界邁進，挑戰不可能的任務！啊哈，筆者將帶領大家一一突破業界所認證的各項函數技能，就讓我們繼續看下去...

首先挑戰的技能項目是「內插函數・Inline」。這個作法的技術特性在於可直接執行"嵌入程式碼"，不必在再做呼叫引用動作，故可有效提昇執行效率，如同變向式快取作法！使用本法是利用記憶體中指定空間維持碼值在長駐記錄、隨時備用狀態，確實有其記憶體負擔，只適用在碼值簡短且具演算行為為佳的時機，而非用在大量程式碼表述處理的場合，避免造成記憶空間過度消耗而產生後續相關問題，這點還請各位多加注意。

```
內插函數.  inline  型式  函數別名( 型式  參數 , ..... ){
                    程式碼描述
                    [ return value ; ]
                }
```

就讓我們架設一個參考範例，體驗設計手法。

【專案代號：Study3_02】

*執行畫面・線上研討...

```
設定半徑及圓周率...
12.34 3.14159

對應圓周長 -- 121.79

請按任意鍵繼續 . . .
```

叫函數本身程式碼，在執行過程中產生動態堆疊運算(stacking-operation)行為模式後再回頭整合求取解答的一種學術型作法。最主要的應用價值在於可讓設計人員減輕正規表述下複雜分析流程能力外，也不需要引用過多的擬人化行為指示動作。

雖然此法有其智慧技術考量，可使電腦模擬人工完整思維能力，但在實際執行過程中所要承受過高的硬體成本，卻也讓專業設計人員望其憂心而不敢輕言應用之，所以一般業界還是選用最符合成本的疊代(iteration)作法，也就是大家所熟悉的迴圈(loop)來取代。不過筆者認為天生我才必有用，還是有其一定的上場時機，所以就讓我們先行學會其設計手法，擇期再用！

```
遞迴函數・原型.   void 函數別名(型式 參數, .....) {
                      程式碼描述
                      函數別名(value, ..... );
                  }
```

```
遞迴函數・具型.   型式 函數別名(型式 參數, .....) {
                      程式碼描述
                      return 終止值 ;
                      return 函數別名(value , ..... );
                  }
```

就讓我們架設二個參考範例，體驗不同感受的設計手法。

```
12!=479001600
請按任意鍵繼續 . . .
```

【專案代號：Study3_03】

*執行畫面・線上研討...

```
設定正整數：159827
反向顯示值 -- 728951
請按任意鍵繼續 . . .
```

【專案代號：Study3_04】

巨匠材哥・線上速解

再來挑戰的技能項目是「覆載函數・Overloading」。這個作法的技術特性在於在使用相同函數名稱下，依據功能上的型式與參數上的變化差異，進行同類式功能設計，電腦也會依據其呼叫值及參考型式找到最佳執行對象，所以業界也另有「同名異式」或「函式多載」之說。

雖然此法通常都會有對應型式及參數目標為搜尋引用，一旦發現查無原型目標指示時，就會自動啟用「參考轉型」方式後再線上尋找相符者，如果依舊無法辨識時就會造成編譯錯誤！在參數列宣告時加入"const 關鍵字"來強調唯一引用關係，可確保執行的穩定性。

```
覆載函數.   多元型式 共同函數別名(多元型式 參數, .....) {
                程式碼描述
                [ return value ; ]
            }
```

就讓我們架設一個參考範例，體驗設計手法。

【專案代號：Study3_05】

*執行畫面・線上研討...

```
正方體・體積 -- 343
長方體・體積 -- 135
請按任意鍵繼續 . . .
```

再來挑戰的技能項目是「樣版函數・Template」。這個作法的技術特性在於會根據參數值來源型式來建構線上動態型式，進而產生參照型式函數。其使用時機在於具相同目標參數但不同型式的關聯式整合設計！雖然也可利用前述覆寫方式處理，但也會因其法編寫太多重複性函數而造成未來程式碼維護上無法同步編修問題，此時就可以適時運用樣版作法來解決！

```
樣版函數.   template <class 動態型式>
            動態型式 函數別名(動態型式 參數, .....) {
                程式碼描述
                [ return value ; ]
            }
```

就讓我們架設一個參考範例，體驗設計手法。 【專案代號：Study3_06】

*執行畫面・線上研討...


```
Set 兩字元 及 兩整數 ....
K b 381 492

        字元中較大者為 -- b
        整數裡較大值為 -- 492

請按任意鍵繼續 . . .
```

最後挑戰的終極技能項目就是「樣版覆載函數 · Template + Overloading」。這個作法的技術特性在於徹底解決了傳統樣版開發下，無法進行多元目標化的麻煩問題。其實本法作業模式很單純原始，就是把型式與參數關係分離開發後，再進行整合引用設計，其中型式問題由樣版負責，而參數引用就用覆載來決定，兩者技術各有所長！相關作法，請往上參考本節中線上語法說明。

就讓我們架設一個參考範例，體驗設計手法。

【專案代號：Study3_07】

*執行畫面 · 線上研討...

㋡巨匠材哥 · 線上速解

```
Change before -- Value1 = 10 , Value2 = 15

        Swap After -- Value1 = 15 , Value2 = 10

--------------------------------------------------

Change before -- Array-1 = 9 , Array-2 = 0

        Swap After -- Array-1 = 0 , Array-2 = 9

請按任意鍵繼續 . . .
```

巨匠真人・實境教學