《程式語言》

試題評析	第一題：屬於基本 BNF 題目，考繪製剖析樹及分辨模糊文法，屬於 93 年關務的相似題。對於認真準備的考生，應該不難拿到滿分。 第二題：儲存體繫結，比較不同細節方式的優缺點與生命期，亦屬於考古題，相似於100年鐵路與93年退役。若考生能充分解釋其優缺點，得到高分亦不難。 第三題：指標型態的優缺點，屬於程式語言的基本概念，可能因較爲簡單，配分較其他題目少，但若能了解指標的使用特性與限制，也可以穩定掌握分數。 第四題：參數傳遞的方式。此題與以往題目相對而言有變化，要求考生自己以程式舉例說明傳值與傳址呼叫的差別。雖然不曾出現此類考題，但考生練習中已會接受大量範例，也不難拿到高分。值得注意的是此題撰寫時，力求舉例簡單易懂，方便考官快速理解給分爲佳。 第五題：此題爲活動紀錄考題，與以往活動紀錄不同的是，此題必須說明活動紀錄中「控制權轉移」的過程，另外也出現多年沒考的靜態鏈。但考生若能掌握活動紀錄運作的流程及活動紀錄每個欄位的名稱、內容，應能對答自如。
考點命中	第一題：《高點程式語言第六回講義》，金乃傑編撰，頁 10-11 ； 《高點程式語言總複習講義》，金乃傑編撰，考點 27。 第二題：《高點程式語言第三回講義》，金乃傑編撰，頁 40； 《高點程式語言總複習講義》，金乃傑編撰，考點 1。 第三題：《高點程式語言第三回講義》，金乃傑編撰，頁 1； 《高點程式語言總複習講義》，金乃傑編撰，考點 4。 第四題：《高點程式語言第二回講義》，金乃傑編撰，頁 4-5； 《高點程式語言總複習講義》，金乃傑編撰，考點 10。 第五題：《高點程式語言第二回講義》，金乃傑編撰，頁 26； 《高點程式語言總複習講義》，金乃傑編撰，考點 13。

一、根據下列文法，其中非終端（non-terminal）以<>符號標示：

<S> → <A>

<A> → <A>+<A> | <ID>

<ID> → w | x | y | z

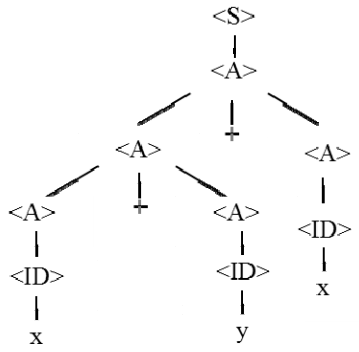
(一)請畫出 x+y+x 所對應之剖析樹（parse tree）。(10 分)

(二)請問此文法是否模糊（ambiguous）？請說明。(10 分)

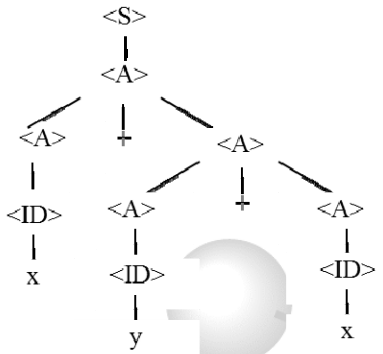
【擬答】

(一) x+y+x 所對應之剖析樹如下：

剖析樹一：



剖析樹二：



(二)此文法屬於模糊文法。模糊文法的定義爲：使用文法來分析語句，若一個語句可以產生兩個或以上的剖析樹，則稱此文法爲模糊文法。由語句 x+y+x 使用題目中的文法分析時，可以產生兩顆以上的剖析樹，故此文法屬於模糊文法。

二、程式語言中的變數依其記憶體配置（allocation）的不同可分為那三類？其生命期（lifetime）有何不同？其優缺點各為何？（24 分）

【擬答】

變數依其記憶體配置分類，可分爲靜態變數（static variable）、堆疊動態變數（stack-dynamic variable）及外顯堆積動態變數（explicit heap-dynamic variable）三類，其內容、優缺點與生命期以表格說明如下：

種類	靜態變數	堆疊動態變數	外顯堆積動態變數
內容	宣告成全域變數或以 static 修飾字宣告的變數。此變數在程式載入階段已經配置好記憶體，放在執行記憶體的 Data 或 BSS 區段。	宣告在函數中的一般變數或參數，此變數要在函數被執行建立活動紀錄時才會配置到區域變數或參數的記憶體區塊中。由於活動紀錄又稱爲 Stake Frame，會隨著函數被動態建立及釋放，故名爲堆疊動態變數。	使用動態配置記憶體的變數，例如：C++中使用 new 指令建立的記憶體空間。此變數儲存在執行記憶體的 Heap 區塊中，若程式中沒有以指令（如 delete）刪除，這些記憶體空間就很可能一直留在系統中直到程式結束。
優點	1.執行效率佳：記憶體位址直接定址，不需要再額外的考察運算就能存取其內容。 2.支援歷史相關特性：不會隨著函數釋放而清除內容，可以用來記錄函數執行次數。 3.提供函數、類別間溝通：不受物件釋放的影響，適合用來做資料分享與傳遞訊息。	1.支援遞迴程式：提供函數動態配置，因此可以支援無法預測執行次數的遞迴程式儲存變數資料。 2.節省記憶體：函數執行完後，記憶體空間會被自動釋放，提供其他函數配置使用。	1.支援動態結構：提供建立最適大小的陣列，而不需因不確定需求先建立過大空間的陣列。 2.支援大型資料傳送：通常配合指標，因此若函數間要傳送大量資料，也可以直接將指標傳遞給其他函數，不需要整筆變數內容完整複製。
缺點	1.記憶體運用缺乏彈性：配置固定空間，不能隨意增減，也不能在執行中釋放。 2.不支援遞迴程式：不能動態配置就無法提供不知道執行次數的遞迴程式儲存資料。	1.效能較差：執行時必須配置、釋除記憶體，花費許多系統額外的處理時間。 2.不支援歷史相關特性：無法儲存函數上次的執行結果，對某些需要記錄過程的程式設計相對不便。	1.容易產生人爲錯誤：如果程式語言沒有垃圾回收機制，可能會產生記憶體洩漏等問題。 2.執行效率較差：無法直接存取記憶體空間。
生命期	從配置執行記憶體開始便存在，直到整個程式結束釋放執行記憶體終止。	配置函數活動紀錄時開始，到函數結束活動紀錄釋放時一起釋放。	程式中建立記憶體空間語法處開始，到執行釋放指令或垃圾回收機制時釋放。若以上階段沒有釋放，

			則會到整個程式結束釋放執行記憶體時終止。

三、在C語言中，指標型態（pointer type）為其一大特色，

(一)請問其指標型態主要的優點為何？（8分）

(二)但是指標型態也帶來不少問題，請舉出指標的問題主要有那些？（8分）

【擬答】

在C語言中，指標是儲存記憶體位址的資料型態，透過指標可以存取特定的記憶體空間的內容，常用以進行陣列的操作、實作傳址呼叫，或用以實作高階函數的函數指標。以下說明指標的優點與問題：

(一)主要優點：

1.彈性大，增加程式撰寫靈活度。

使用指標可以直接存取記憶體空間，跳脫程式結構的限制，因此可以透過指標將資料以「傳址」方式在副程式呼叫中傳送（如下片段程式所示）。

```
void swap(int *a, int *b){        //副程式用指標存取特定位址的記憶體內容
        int temp = *a;                    //對資料進行互換
       *a = *b;
       *b = temp;
}
void main(void){
       int x = 3, y = 5;
       swap(&x, &y);              //傳遞 x 與 y 的爲址
}
```

也可以用指標存取陣列的特定元素（如下片段程式所示）。

```
int A[2][3] = {{2, 3, 5}, {7, 11, 13}};
int *p1 = &A[0][0];                    //將陣列首元素傳給指標
*(p1+2) = 5;                           //存取陣列 A[0][2]的資料
```

另外也可以建立函數指標，將函數作爲參數傳到另一個函數中。

2.避免傳值呼叫，減少記憶體使用空間。

一般的傳值呼叫會參數傳遞時必須要把實際參數的值複製到形式參數中，若參數資料內容龐大，例如大型陣列或物件，則複製會造成程式中儲存兩份相同的大型內容。使用傳址呼叫可以省去複製變數的階段，節省程式的執行空間。

3.直接存取記憶體位址，對大量資料存取速度較快。

如前所述，大量資料傳送時若需要先經過複製則相當耗時。使用指標可以直接存取到目標資料，故亦能節省大量資料的存取時間。

(二)指標的主要問題：

1.學習難度高，程式可讀性低。

由於指標使用方式相當具有彈性，例如用指標指向陣列在位址移動時，就有好幾種方式表達相同的位址意義；另外指標亦可指向儲存記憶體位置的空間（即雙指標）。如此變化性造成學習難度提升，並且也可能降低程式可讀性，使程式不易理解。

2.邏輯錯誤不易發現。

延續前述問題，因爲程式理解難度變高，若程式中有邏輯錯誤也不易檢查出來。例如若指標指向一個長度爲6的陣列，則在進行指標運算時，當位移超過陣列的範圍比如*(ptr+6)在程式編譯時不會出錯，只有執行時才會發生錯誤，使除錯變得更困難。

3.可能存取程式以外的記憶體，降低安全性。

某些安全性較低的作業系統無法限制程式能存取到的記憶體範圍，惡意程式就可以透過指標存取程式以外的記憶體資料（例如動到作業系統所使用的記憶體），對其進行修改或植入指令碼，破壞系統正常運作或盜取機密資料。

4.間接讀取資料，對簡單變數存取速度可能較慢。
雖然在大量資料時可以減少複製所花費的時間與空間，但若是簡單變數如 int、double 等，直接存取反而效率較高。

四、請用您熟習的語言，設計一程序（procedure）或函式（function），此程序或函式在參數傳遞方式不同時：以值傳遞（passed by value）、以參照傳遞（passed by reference），會產生不同的效果，並說明為何會有不同。（20 分）

【擬答】

以 C++設計程式如下：

```
int side_add(int n){
        n += 3;
        return n;
}
int main(){
        int a = 5;
        int b = side_add(a)+a;
        printf(“a=%d, b=%d”, a, b);
}
```

以上程式若採傳值呼叫，輸出爲：
a=5, b=13
原因：
變數 a 爲 5，傳入副程式 side_add()形式參數時，在副程式建立 a 的副本 n，此時 a 與 n 是完全獨立的記憶體（但 n 的值卻複製自 a）。在副程式 side_add()中，n 被設定爲 8，副程式並將 8 傳回主程式。主程式以 b 接收副程式回傳值，故 b = 8(副程式回傳)+5(主程式中沒被改變的 a)得到 13。

採傳參考呼叫輸出爲：
a=8, b=16
原因：
變數 a 原爲 5，傳入副程式 side_add()形式參數 n 與 a 參考到相同的記憶體空間。此時副程式敘述將 n 設爲 8，並傳回 n。副程式 n 與主程式 a 共用記憶體空間，因此 a 的值也被修改爲 8。故 b = 8(副程式回傳)+8(主程式被改變的 a)得到 16。

五、程式語言以堆疊結構實作副程式或函式呼叫的順序，堆疊中的元素稱為啟動紀錄（activation record）。
（一）假設函式 foo 呼叫函式 bar，請說明 foo 要將那些資訊存入 bar 的啟動紀錄中，才能將控制權交給 bar；bar 要將那些資訊存入啟動紀錄中，才能將控制權還給 foo？（10 分）
（二）有些語言在啟動紀錄裡儲存靜態連結（static link），靜態連結與動態連結（dynamic link）的用途有何不同？那種特性的程式語言需要靜態連結？（10 分）

【擬答】

(一) foo()呼叫 bar()
1.控制權交給 bar()前，要儲存至 bar()啓動紀錄的資訊有：
(1)回傳位址（Return Address）：儲存 foo()呼叫 bar()的程式碼執行位址。
(2)動態鏈（Dynamic Link）：指向 foo()的啓動紀錄。
(3)參數（Parameter）：複製 foo 的實際參數。
2.控制權還給 foo()前，要儲存至 bar()的啓動紀錄資訊有：
回傳值（Function Value）：bar()內程式計算的結果。

(二)靜態連結是指向該副程式的上層副程式；而動態連結指向呼叫該副程式的副程式區塊。例如：

```
f(){
        g(){
        }
        h(){
                g();
        }
        h();
}
```

在上述程式結構中，f()函數中宣告了 g()與 h()兩個函數以及一個執行 h()的敘述。而 h()函數中又呼叫 g()函數。因此在此例中，若 g()建立活動紀錄時，則：

●靜態連結（static link）：指向 f()，因爲 f()是 g()函數結構的上層函數（f()包住 g()）。

●動態連結（dynamic link）：指向 h()，因爲 h()是 g()的呼叫者（程式中 h()呼叫 g()）。

由上述程式範例可以發現，靜態連結必須要在允許巢狀（nested）副程式的程式語言中使用，這樣才可能發生 f()包住 g()的情況，例如 PASCAL 語言。

補充：

函數呼叫執行的步驟

呼叫時：

1.儲存呼叫者的狀態（活動紀錄中提供暫存目前狀態的空間，以方便程式回傳時繼續執行）

2.判斷每個參數的參數傳遞方法，並安排適當的存取權

3.被呼叫者配置區域變數的儲存空間（一般會以暫存器優先，但資料量太多或太大時，還是會放在活動紀錄中）

4.初始化區域變數值

5.安排副程式控制權轉移（從呼叫者轉移到被呼叫者）

6.安排呼叫者控制權返回位址與方法

7.安排被呼叫者存取可見的非區域變數

回傳時：

1.移動最後參數值到呼叫者實際參數（call-by-result 時）

2.釋放被呼叫者區域變數的儲存空間

3.撤銷被呼叫者存取可見非區域變數的存取權

4.控制權返回呼叫者