《程式語言》

一、在程式語言中，請舉例並說明何謂混淆的文法（ambiguous grammar）？（20 分）

命題意旨	本題旨在測驗考生對於混淆文法一詞之掌握度，除須理解定義，還需能具體以案例說明混淆之處。以考古題而言，相似於 99 及 103 年檢事官試題。
答題關鍵	本題解題之關鍵在以具體之案例指出混淆文法，分 2 部分： 1.混淆文法之名詞定義。 2.實際寫出具混淆文法之 BNF 文法，並代入文句證明其屬於混淆文法。
考點命中	《高點程式語言講義》第四回，金乃傑編撰，頁 30-37。

【擬答】

混淆文法（ambiguous grammar）指的是若一語言的某一語句，依其 BNF 文法規則來推導，可繪出兩棵以上的不同剖析樹。

舉例而言，若有以下 BNF：

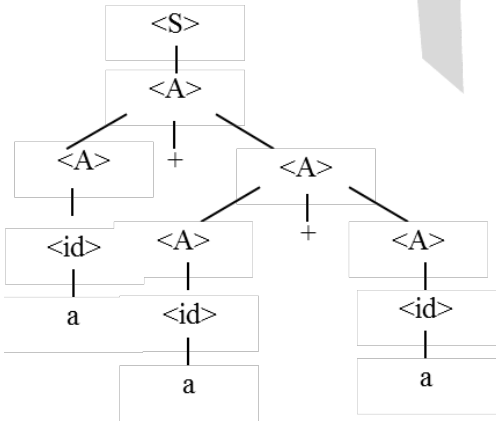
<S>→<A>

<A>→<A>+<A> | <id>

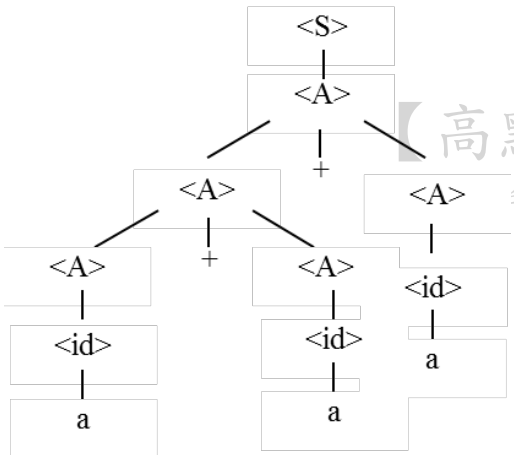
<id>→ a | b | c

則若語句為 a + a +a，可以產生兩棵剖析樹如下：

剖析樹一：



剖析樹二：



二、請寫出至少三種參數傳遞（parameter passing）方法，舉例並敘述其不同之結果？（25 分）

命題意旨	本題旨在測驗考生是否能理解何謂函數呼叫的參數傳遞，並實際舉出案例說明其運作方式。以考古題而言，相似於 102 高考、105 地特，但與以往不同的是本題為要求考生自行舉出程式碼案例，而非從題目中計算各種傳遞方法之執行答案。
答題關鍵	本題解題關鍵在能舉出在多種傳遞方法下會產出不同答案的程式碼，並依照此程式碼推導出各種傳遞之結果。
考點命中	《高點程式語言講義》第二回，金乃傑編撰，頁 4-5。

【擬答】

以下討論四種參數傳遞方法：傳值呼叫、傳參考呼叫、傳名呼叫及傳值與結果呼叫。

假設以仿 JavaScript 語法撰寫程式如下：

```
function swap(x, y){
    var temp = x;
    x = y;
    y = temp;
}

var v = 1;
var l[2] = {-1, 0};
swap(v, l[v]);
```

將四種傳遞方法之概念、執行完結果整理如下表：

傳遞方法	主要概念	變數 v	變數 l
傳值呼叫	將參數值複製到副程式中，副程式使用獨立的記憶體。	1	{-1, 0}
傳參考呼叫	主程式與副程式使用相同記憶體。	0	{-1, 1}
傳名呼叫	在副程式中先把形式參數的變數名稱取代為實際參數的變數名稱，再將主程式與副程式使用相同記憶體。	0	{0, 0}
傳值與結果呼叫	先執行傳值呼叫，當副程式結束時再將副程式中區域變數傳回主程式。	0	{-1. 1}

三、請敘述在程式語言中，靜態領域（static scoping）和動態領域（dynamic scoping）有何不同？請舉例並說明其優缺點各為何。（25 分）

命題意旨	本題旨在測驗考生對於函數中變數數值取值之兩種主要方法，靜態領域法、動態領域法之了解程度。除了名詞解釋，還需要舉出優缺點與程式碼案例。相較於以往考題，本題為較冷門題型，但仍屬於考生應準備範圍。
答題關鍵	本題解題關鍵在能說明靜態領域法與動態領域法之不同，宜用表格撰寫，提升作答之可讀性。
考點命中	《高點程式語言講義》第三回，金乃傑編撰，頁 93-94。

【擬答】

將靜態領域法與動態領域法之定義、優缺點與舉例比較如下表：

	靜態領域法（Static Scoping）	動態領域法（Dynamic Scoping）
定義	對於區段內沒有宣告的變數，往上層程式單元尋找定義。	對於區段內沒有宣告的變數，往呼叫方向反向尋找定義。
優點	容易從程式結構判斷變數數值：因靜態領域法遇到區段內沒有宣告的變數，會往程式結構上層尋找，因此很容易從結構判斷變數數值。	程式執行彈性較高：可根據程式呼叫對象動態決定其中所用到的變數值，使相同程式在不同環境中呼叫可以有不同結果。
缺點	程式執行彈性較低：變數的取值是根據程式撰寫時的結構決定，因此在不同情境呼叫都會執行相同結果，無法根據呼叫者決定要執行的內容。	難以從程式結構判斷變數數值：變數之數值必須從程式執行時環境來判斷，因此函數所宣告時所在的位置不足以預測變數值，使程式可讀性較差。

	靜態領域法（Static Scoping）	動態領域法（Dynamic Scoping）

舉例	以仿 JavaScript 語法撰寫程式如下：
	function a(){
	var x = 5, y = 10;
	function b(){
	var y = 6;
	print(x + ", "+y);
	}
	function c(){
	var x = 3;
	b();
	}
	c();
	}
	則以靜態領域法輸出為：
	5, 6
		以仿 JavaScript 語法撰寫程式如下：
		function a(){
		var x = 5, y = 10;
		function b(){
		var y = 6;
		print(x + ", "+y);
		}
		function c(){
		var x = 3;
		b();
		}
		c();
		}
		則以靜態領域法輸出為：
		3, 6

支援程式	C/C++、Java	LISP

四、在物件導向程式語言的設計中，覆寫（overriding）和多載（overloading）有何不同？請舉例並說明其不同之處。（20 分）

命題意旨	本題旨在測驗考生對於物件導向程式設計中覆寫與多載此二名詞之掌握度。由於此二名詞之英文相似，在程式撰寫時又都會用到同樣名稱但不同內容的方法，因此極容易混淆。本題相關考點參考 101 年高考試題，唯當年係以程式範例要求考生指出其中的覆寫與多載，但今年為讓學生自行整理比較。
答題關鍵	本題解題關鍵在能說明覆寫與多載之不同，宜用表格撰寫，提升作答之可讀性。
考點命中	《高點程式語言講義》第四回，金乃傑編撰，頁 61-62。

【擬答】

將覆寫(overriding)與多載(overloading)比較如下表：

	覆寫(overriding)	多載(overloading)
發生情況	在子類別中對父類別的方法重新定義，使子類別使用方法時，操作到子類別中定義新內容的方法。	在同一個類別中，使用相同名稱定義方法。藉由宣告不同型態或數量的參數來決定實際呼叫的方法。
目的	使子類別可以重新定義方法的執行內容，以更符合需求。	使相同名稱的方法具有不同的執行功能，亦可達到對方法設定預設值。
繫結時間	執行時根據實際的物件決定要呼叫的方法，屬於（晚期）動態繫結。	編譯時即可決定實際呼叫到的方法，屬於靜態（早期）繫結。
參數型態	相同	數量相同時必不同
參數數量	相同	型態相同時必不同
回傳型態	相同	可不同（但不能用來決定）
存取等級	大於等於原等級	可不同

舉例	class S extend P{
	void download(){
	//DEF
	}
	}
	
	class P{
	void download(){
	//ABC
	}
		class L{
		void print(){
		
		}
		void print(int n){
		
		}
		void print(int a, int b){
		
		}

	覆寫(overriding)	多載(overloading)

	}
	說明:子類別中的 download()方法 overriding 父類別的 download()方法，重新定義 download()的內容。
		void print(String s){
		
		}
		}
		說明：類別中 print()屬於多載方法，根據不同的參數型態與數量，可以呼叫到不同的 print()方法。

五、考慮以下 Prolog 程式：

```
mystery([],[]).
mystery([X|Y],[X,X|Z]):-mystery(Y,Z).
```

(一)以上函式（function）mystery 的功能為何？（5 分）

(二)以下目標的結果是什麼？必須顯示所有跟蹤步驟（tracing steps）。(5 分)

```
mystery([1,2,3],X).
```

命題意旨	本題旨在測驗考生對於 Prolog 語言之執行結果，並能說明程式之目的與執行過程。相似於 102 年高考之題目，但相較而言難度更高。
答題關鍵	本題解題分兩部分，先指出題目中程式之功能，再說明程式執行結果與追蹤步驟。以實務上來說，建議考生先計算程式執行結果，再根據結果推測程式之功能，有事半功倍之效。
考點命中	《高點程式語言講義》第五回，金乃傑編撰，頁 12-14。

【擬答】

(一)重複串列元素，該程式傳入一串列，並傳回每個元素重複兩次之串列。

(二)輸出結果為：X = [1, 1, 2, 2, 3, 3].

跟蹤步驟如下：

1. [trace] 1 ?- mystery([1,2,3],X).
2. Call: (7) mystery([1, 2, 3], _G2474) ? creep
3. Call: (8) mystery([2, 3], _G2554) ? creep
4. Call: (9) mystery([3], _G2560) ? creep
5. Call: (10) mystery([], _G2566) ? creep
6. Exit: (10) mystery([], []) ? creep
7. Exit: (9) mystery([3], [3, 3]) ? creep
8. Exit: (8) mystery([2, 3], [2, 2, 3, 3]) ? creep
9. Exit: (7) mystery([1, 2, 3], [1, 1, 2, 2, 3, 3]) ? creep

補充：

說明上述追蹤步驟：

mystery([1, 2, 3], X).，將 X 用變數_G2474 表示

mystery([1 | 2, 3], _G2474)，問題化為：mystery([2, 3], Z).

mystery([2, 3], Z).，將 Z 用變數_G2554 表示

mystery([2 | 3], _G2554)，問題化為：mystery([3], Z).

mystery([3], Z).，將 Z 用變數_G2560 表示

mystery([3], _G2560).，問題化為：mystery([], Z).

mystery([], Z).，將 Z 用變數_G2566 表示

根據規則，存在 mystery([], []).

返回 mystery([], _G2566).， _G2566 得解為[]，故 Z 為[]

返回 mystery([3], _G2560).，_G2560 得解為[3, 3]，故 Z 為[3, 3]

返回 mystery([2 | 3], _G2554).，_G2554 得解為[2, 2, 3, 3]，故 Z 為[2, 2, 3, 3]

返回 mystery([1 | 2, 3], _G2474).，_G2474 得解為[1, 1, 2, 2, 3, 3]，故 X 為[1, 1, 2, 2, 3, 3]