

《程式語言》

一、請說明early binding（如C語言）及late binding（如Perl語言）之間最大的差異，並列出兩者主要的優點。（10分）

試題評析	此題中的late binding相似於100檢事官、97高考與95地特的考古題，為實作，屬於物件導向三大特性中多型的方法；而early binding雖然單就名詞為第一次出現在國考中，但因考古題中仍有不少比較，應不難推斷出為「編譯時期」的繫節。此題若有充分準備，應可獲得不錯的分數。
考點命中	1.《103高上程式語言》第四回，金乃傑編撰，頁45-53。 2.《103高上程式語言》第三回，金乃傑編撰，頁34。 3.《103高上程式語言》總複習，金乃傑編撰，考點19，頁51-52。

答：

在程式語言中，繫節（Binding）指的是將程式名稱與功能連結的動作，而Early Binding即為早期繫結，意指在「編譯階段（Compilation Phase）」進行繫節；Late Binding則是在「執行時期（Runtime Phase）」才能進行繫節，故最大差異為繫節時間。

因此如同C語言，Early Binding在編譯階段即將函數的名稱與執行內容連結，除了可以方便預測函數傳回的資料、執行的結果外，還可以讓編譯器對函數進行最佳化，例如：將迴圈優化或配置適當的暫存器供函數使用。

而如Perl等語言，Late Binding提供了「多型（Polymorphism）」，在執行時期才會依照物件真實的型態決定要呼叫的函數實體。例如：使用父類別容器裝入父類別物件與子類別物件，透過容器執行則父類別物件時，會呼叫父類別裡的方法；執行子類別物件時則會執行子類別中的方法（如下以Java展示，假設Triangle繼承自Shape）。

```
Shape s[] = new Shape[2];
```

```
s[0] = new Shape();
```

```
s[1] = new Triangle();
```

使用s[0].height()與s[1].height()呼叫到不同的方法。如此可以增加程式的彈性與可維護性，使相同程式碼依照執行時真實物件提供不同操作模式，也可以在需要時透過多型概念替換傳入物件，在增加新功能時可以不會影響原始程式碼執行。

以表格比較差異如下：

	Early Binding	Late Binding
繫節時間	編譯時期	執行時期
主要優點	1. 提供編譯器極大的最佳化空間，優化程式碼與暫存器配置。 2. 可進行完整的檢查，避免執行時期發生錯誤。 3. 執行時期因不用再尋找定義因此效率較高。	1. 提供「多型」 2. 增加程式彈性，可以依照實際需求進行呼叫，減少程式碼撰寫。 3. 增加可維護性，透過模組化抽換功能元件，更新部分功能時不影響原有系統架構。

二、參考下述BNF grammar

$S \rightarrow S + S \mid S - S \mid S * S \mid S / S \mid (S) \mid a$ 【註：S只使用一次，S重複必究！】

請回答總共有幾個不同的剖析樹（parse tree）可得到下列結果。本題不需畫出剖析樹，但請說明。（每小題5分，共20分）

(一) $a + a * a$

(二) $a + a * a / a$

(三) $a + a + a + a$

(四) $(a + (a + a)) + a$

試題評析	此題亦為考古題，相似於101高考與100關務4等，考驗同學畫出剖析樹的數量。惟與前述兩題不同的是此題不必畫出所有剖析樹，只要列出數量即可，就難度與時間花費度而言應更為簡單。若同學結合資料結構，以樹型判斷能畫出樹有幾棵的概念，小心謹慎作答，應可獲得滿分。
考點命中	1.《103高上程式語言》第六回，金乃傑編撰，頁12-14。 2.《103高上程式語言》總複習，金乃傑編撰，考點27，頁78-80。

答：

剖析樹的數量是依照樹的形狀來判斷。由於題目中BNF規則運算子符號為二元運算子，且沒有優先順序，因此可將2個節點視為1組，剖析樹數量如下：

- (一) 左斜、右斜 = 2棵
- (二) 左斜→左斜、左斜→右斜、左右對稱、右斜→右斜、右斜→左斜 = 5棵
- (三) 左斜→左斜、左斜→右斜、左右對稱、右斜→右斜、右斜→左斜 = 5棵
- (四) 左斜 = 1棵

解析：

此題可以從樹的形狀切入，第二三小題可以想成「四個樹葉節點的樹」有幾種形狀。第四小題因為使用括號限制能夠展開的方向，故只有一棵剖析樹。

三、參考右圖類C語言的程式，請依下列小題所述參數傳遞方式，寫出執程式B後之輸出結果。（每小題10分，共20分）

- (一) Y is passed by value.
- (二) Y is passed by reference.

```

Program B ( ) {

function A (X: integer) {
    X := X + 1;
    write (X, Y);
}

var Y : integer;

Y := 1;
A (Y);
write (Y);
}

```

試題評析	參數傳遞方式同為考古題，就題目結構使用Pascal語法，在函數中宣告函數，相似於93高考，但只要掌握各參數傳遞方法及領域（Scope）的概念即可輕鬆作答，謹慎的同學要拿到這20分應勢在必得。
考點命中	1.《高上程式語言》第二回，金乃傑編撰，頁7, 16-18。 2.《高上程式語言》第三回，金乃傑編撰，頁42。 3.《103高上程式語言》總複習，金乃傑編撰，考點10，頁23-24。

答：

假設write()印出資料內容後會換行。

(一) passed by value輸出結果如下：

2, 1

1

(二) passed by reference輸出結果如下：

2, 1
2

四、請用 Scheme 或 Lisp 等 Functional Programming Language 寫一個符合下述規範的遞迴函數 calculate。所寫的遞迴函數應該越簡潔越好。(20分)

(calculate A B C) : A 是某函數 (function), B 是一整數 (integer), C 是任一數值 (value), 由傳滿足 $N \geq B$ 且 $(A \ N) = C$ 的最小整數 N。例如

(calculate list 0 (3)) 應回傳 3.

(calculate (lambda (X) (> X 10)) 0 T) 應回傳 11.

(calculate (lambda (X) (* X X)) 0 100) 應回傳 10.

試題評析	本題要求用 Scheme 或 Lisp 寫出程式，若沒有對函數型程式語言運作邏輯有充分的了解，寫出符合題意的答案實屬困難。 但此題實為 93 高普考相似題，若能記得講義中 Scheme 介紹的常用函數如 define、cond、equiv?... 的使用方法，配合講義後之補充練習題的解題經驗，應能仍從題目說明的回傳找到規則，以遞迴概念處理回傳答案，一樣可以寫出接近標準答案的結果。
考點命中	1. 《高上程式語言》第五回，金乃傑編撰，頁 4-10。 2. 《103 高上程式語言》總複習，金乃傑編撰，考點 22，頁 60-62。

答：

以 Scheme 撰寫如下：

```
(define (calculate A B C)
  (cond
    ((equal? (A B) C) B)
    (else (calculate A (+ B 1) C))
  )
)
```

解析：

此程式以仿 C 語言撰寫如下：

```
calculate(A, B, C){
  if(A(B) == C) return B;
  else return calculate(A, B+1, C);
}
```

切入點：

依照題意要滿足 $A(X) = C$ ，如：

函數 A	結果 C	函數輸入 X
list(X)	(3)	3
lambda(X){ return (X > 10)?true:false }	true	最小 X 必須為 11
lambda(X){ return X*X }	100	10

因題目又提示輸入的 X 必須要小於等於 B，所以我們可以用 B 每次加 1，代到符合結果 C 為止。再將此想法先以仿 C 語言遞迴呈現，最後轉為 Scheme 語言。

可將上述 Scheme 程式至 <http://repl.it/languages/Scheme> 網站執行，依序帶入察看結果。

(calculate list 0 '(3))

(calculate (lambda (X) (> X 10)) 0 #T)

(calculate (lambda (X) (* X X)) 0 100)

五、請參考以下的 Java classes，解釋下列各小題的語法是否正確，如不正確請說明原因。(每小

題5分，共15分)

```
class X { public void x ( ) { ... } }
```

```
class Y extends X { public void y ( ) { ... } }
```

```
class Z extends Y { public void z ( ) { ... } }
```

```
(一)int count (Set<Y> s) { ... } ... count (new TreeSet<Z> ( ));
```

```
(二)int count (Set<? extends Y> s) { ... } ... count (new TreeSet<Z> ( ));
```

```
(三)lint count (Set<? super Z> s) {for (X a : s) a.x( ); ... };
```

試題評析	此題為Java的泛型（Generics），接近於C++的樣板（Template），為新出現的題型，若缺乏充分實務Java經驗，應很難對泛型的使用方法及其特性了解，因此較難作答。 但課本第四回中有不少地方有提到相關的概念，如P.3的資料結構介紹了List型態、Map型態，都有使用到泛型傳入；Java與UML圖的P.58-59亦實際操作LinkedList；P.85 C++的函數樣板課堂中亦有補充泛型之概念。 若能結合上述概念，再配合函數參數傳遞與多型運作的知識，充分準備的考生應有拿分機會。
考點命中	《高上程式語言》第四回，金乃傑編撰，頁3、85。

答：

(一)不正確。

在count中，形式參數型態為Set<Y>，但實際參數型態為TreeSet<Z>，故可將問題簡化為：

```
Set<Y> s = new TreeSet<Z>();
```

其中<Y>代表傳入型態必須要為Y，雖然Z繼承自Y，但Z仍與Y不同，故不符合條件。

(二)正確。

在count中，形式參數型態為Set<? extends Y>，但實際參數型態為TreeSet<Z>，故可將問題簡化為：

```
Set<? extends Y> s = new TreeSet<Z>();
```

其中<? extends Y>代表傳入的型態必須要是Y的子類別，而Z繼承自Y，Z是Y的子類別，故符合條件。

(三)不正確。

在Java的泛型（Generics）中，若沒有確切指定變數型態，則必須要用Object作為變數型態，故題目中函數將內a型態以X表示為錯誤的，應改為：for(Object a: s) ((X) a).x();

解析：

此題使用Set結構，與課堂中介紹的LinkedList有異曲同工之妙。由於TreeSet繼承自Set，用課文中相近語法比擬為：

```
TreeSet<X> s = new TreeSet<X>(); //課文中LinkedList相近語法
```

結合多型概念可改寫為：

```
Set<X> s = new TreeSet<X>(); //考題中形式參數與實際參數對應
```

另外題目類別中的函數若為建構函數，則名稱必須與類別名稱大小寫相同、且不須加上回傳型態。

六、下述Prolog程式可推論圖形中A點到B點的可行性，但是無法知道路徑。請改寫該程式使其可以用Prolog list來記錄A點到B點的路徑。若A點到不了B點，則回應fail（false）即可。（15分）

```
get_to (A, B) :- path (A, B).
```

```
get_to (A, B) :- path (A, C), get_to (C, B).
```

試題評析	本題為本試卷難度最高的一題，要以Prolog撰寫程式，並且還要操作串列型態，若沒有Prolog的實戰經驗，要寫出標準答案實屬困難。 但實際上解答所使用的元素亦在講義中介紹，如93高考選擇題中的append函數、[HIX]的用法，都可以在考古題中找到線索，就高點學生而言，若充分練習講義練習題，仍可得到分數。
考點命中	《高上程式語言》第五回，金乃傑編撰，頁14-19。

答：

以Prolog撰寫，將規則子句改寫如下：

```
get_to(A, B, P) :- path(A, B), append([A], [B], P).
```

```
get_to(A, B, [A|P]) :- path(A, C), get_to(C, B, P).
```

並加入以下規則子句：

```
append([], Y, Y).
```

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

若事實子句如下：

```
path(a, b).
```

```
path(b, c).
```

```
path(c, d).
```

則執行結果以下表展示：

詢問	回傳
?- get_to(a, c, P).	P = [a, b, c]
?- get_to(c, a, P)	false

解析：

題目中原始規則搭配解答中事實，只會回傳true和false。

但題目要求必須要回傳路徑，因此搭配93年高考的append方法，將經過的原子串起來。

第一個敘述改寫為：

```
get_to(A, B, P) :- path(A, B), append([A], [B], P).
```

目的是將經過的路徑存在P中，而P是由append將A與B連在一起產生的。

值得注意的是，由於A與B都是「原子」，不是串列。但append是將兩個串列相連，因此傳入append時，必須先用中括號將A與B包起來變成原子。如此append會將結果儲存在P中，可再由get_to中的P去讀取傳回。

第二個敘述改寫為：

```
get_to(A, B, [A|P]) :- path(A, C), get_to(C, B, P).
```

由於get_to已經被改寫為三個參數，因此後面呼叫時也要把P傳入。但值得注意的是「形式參數」的地方，由於路徑必須要包括開始的原子，所以將[A|P]傳入，如此Prolog會自動將A串在P串列的前面，達到將首元素A也加入串列的功能。

【版權所有，重製必究！】