《程式語言》

試題評析

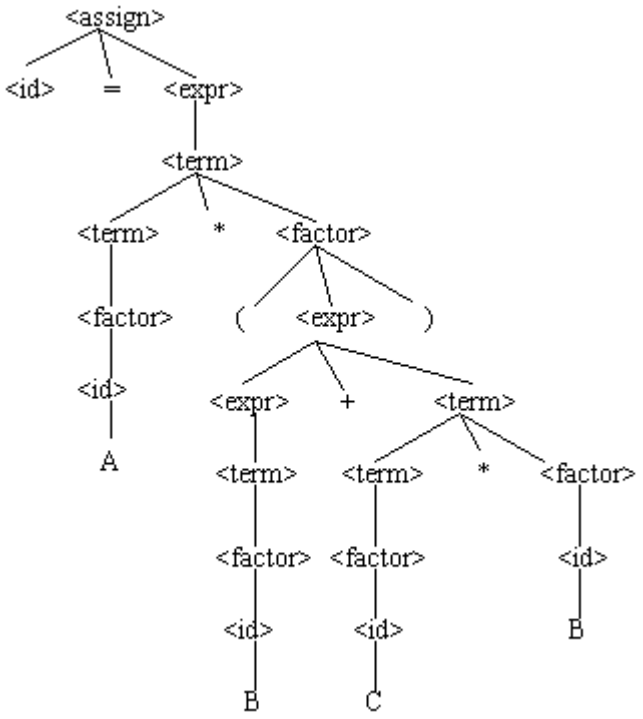
今年程式語言考題除了第三題之外其餘都是經典題型，題目均是重要章節的重點，分析如下：
第一題：標準文法的範圍，爲運算式的非模糊文法，畫出展開過程的剖析樹即可。
第二題：運算式的範圍，考運算式的邊際效用以及C語言程式語法。
第三題：考XML的應用工具，屬於名詞定義型的題目。
第四題：考物件導向以及C++中樣板函式的概念，也是基本題型。
第五題：考評估語言標準以及編譯流程。

綜合來說，今年題目除了第三題之外，均爲老師總複習以及課程中強調的考試趨勢，包括文法、程式撰寫、物件導向編譯器的相關重點。一般程度的學生大約可以拿60分，而理解上課內容的同學要拿75分以上不是難事。

一、使用下列的BNF語法，繪出A=A*（B+C*B）的剖析樹（parsing tree）。（16分）

<assign> -> <id> = <expr>
<id> -> A | B | C
<expr> -> <expr> + <term> | <term>
<term> -> <term> * <factor> | <factor>
<factor> -> (<expr>) | <id>

答：



二、考慮下面的C語言程式片段，並回答下列問題。（每小題10分，共20分）

```
int abc(int *k) {
    *k +=4;
    return 3 * (*k) - 1; }
void main(){
     int i = 10, j = 10, result1, result2;
     result1 = (i / 2) + abc(&i);
     result2 = abc(&j) + (j / 2); }
```

(一)如果在運算式中的運算元估算的順序是由左到右，則result1，則result2的值為何？

(二)如果在運算式中的運算元估算的順序是由右到左，則result1，則result2的值為何？

答：

(一)result1=5+41=46
result2=41+7=48

(二)result1=7+41=48
result2=41+5=46

三、XML是經常用在網路資料交換之語言，程式設計師經常使用DOM或SAX二種應用程式介面（Application Programming Interface）來存取及處理XML的資料，請說明這二種應用程式介面之全名、主要特性及比較此二種應用程式介面（在何種狀況下使用會較好）。（24分）

答：

(一)全名

DOM→Document Object Model
SAX→Simple API for XML

(二)主要特性

1.DOM：

XML DOM就是XML Document Object Model物件模型，屬於XML文件程式設計的介面文件，將XML文件視爲樹狀結構的節點，提供程式設計介面的屬性、方法和物件，透過XML DOM，程式設計者能夠瀏覽XML文件，新增、刪除和修改節點的資料。

2.SAX：

一組程式設計介面，將XML文件視爲一個文字流的資料，在讀取XML元素時觸發一系列的事件，只需撰寫事件處理程序，就可以取得XML元素的內容.SAX載入XML檔案時有如開啓一循序檔案(Sequential File)，將XML元素和內容視爲文字檔的字元讀入，在讀到XML元素的開始標籤，結束標籤和內容時將產生一系列的事件。

(三)比較：

1.SAX是一種唯讀，只能向前的Forward-Only資料處理方式，而DOM則是剖析整個文件，完整瀏覽和更新樹狀資料結構。

2.SAX在記憶體的使用比較有效率。

3.SAX執行速度較快。

4.DOM會先將整個文件讀入，並且轉換成tree的型態，如果說在xml文件中的節點會被重複讀取的話，這個方法當然有好處，因爲所有的資料已經被存在記憶體裡面了。但是，在第一次讀取的時候速度就會比較慢，因爲需要花時間來轉換成tree的格式。SAX則剛好相反，處理原則是使用event model，每抓到一個有定義的標籤，就引發對應的event來進行處理，這樣做當然是比較適合那些只被讀取一次的文件，而SAX並未儲存資料於記憶體中，所以第二次讀取的時候仍然要從頭處理一次。比較起來，SAX在第一次讀取的速度比較快，而DOM在之後的讀取比較快。

四、下列有關物件導向程式設計的試題。

（一）何謂晚期捆束(Late Binding)，與虛擬函數（virtual function）及多形（polymorphism）有何關聯？（10分）

(二)何謂函數樣板（Function Template）？有何作用或好處？（8分）

(三)以C++語言寫一個可以將二個值交換的函數樣板；函數名稱為swapValues，二個參數名稱分別為variable1及variable2。（8分）

答：

(一)先分別描述晚期捆束、虛擬函數、及多形的意義如下：

1.晚期捆束：函數呼叫所聯繫到特定函數的定義是延遲到程式執行時期才發生時稱爲late binding或dynamic binding動態連繫。

2.虛擬函數：虛擬函數是一種類別成員函式，它在基底類別中使用關鍵字"virtual"宣告（定義），並在衍生類別中重新定義虛擬函式，這將成員函式的操作決議（Resolution）推遲至執行時期再決定。

3.多形：多型是一個讓單一介面用在一般性動作的功能。特定的工作取決於狀況的實際性質。在物件導向的定義中指的是可以用同一種方法去操作不同種類的物件的現象。

此三者的關係爲：多型是指可用同種方式操作不同物件的現象，實際操作的物件是要到執行時期才決定，所以要實做多型的手段爲利用語言提供的虛擬函式，將特定的函式呼叫所繫結的定義延遲到執行時期才決定，這種繫結方式又稱爲晚期捆束(late binding)。

(二)函數樣板可以用同一函式描述處理各種不同型態的參數，編譯器會依照特定參數型態將函數樣板展開成特定資料型態的定義，程式設計師不需自己提供各種資料型態版本的函式定義,讓程式寫作簡化。

(三)

```
template<class T>
void swapValues(T &variable1, T &variable2)
{
    T temp = variable1;
   variable1 = variable2;
    variable2 = temp;
}
```

五、(一)評估程式語言的優劣，一般可以分為那四種？（4分）

(二)一般程式語言從撰寫到執行，可以分成三種方式，其中一種為編譯，即程式寫好後經過編譯程式（compiler）編譯成執行檔後再執行。請問另外二種是什麼？請說明這二種的作法為何？（10分）

答：

(一)1.可讀性(readability)：程式容易閱讀與容易了解的特性。

2.可寫性(writability)：容易建立程式的特性。

3.可靠性(reliability)：語言的設計可讓使用者不易犯下錯誤，即使犯錯也很容易找出來之特性。

4.成本(cost)：包括

(1)學習成本。

(2)撰寫程式之成本。

(3)編譯成本。

(4)執行成本。

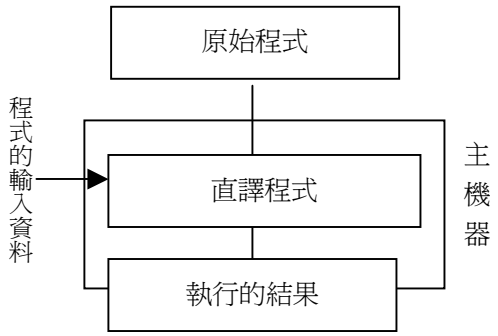
(5)編譯程式之建構成本。

(6)低可靠性之成本。

(7)維護程式之成本。

(二)1.直譯

直譯程式是按照高階語言所寫的程式執行時敘述的邏輯順序，逐指令轉爲機器語言指令並且執行之。每次執行程式時都必須要重新逐行翻譯並執行原始程式機器語言指令並且執行之。



2.混合式實作

混合式是先將原始程式經過編譯流程，產生和機器無關的中間碼(通常爲類似組合語言的格式)，然後執行時再利用直譯的方式去執行中間碼，好處是可以跨平台執行程式，且執行的速度比純直譯的方法執行快。