《程式語言》

試題評析	今年檢事官程式語言偏重軟體開發知識，例如：第三、第五題，同學需要具備實際開發經驗才能回答較完整，至於一、二、四題則是一般性題目，要注意的是必須回答題目再加上適當範例，預估一般分數落在 40～60 分間。

一、(一)試說明邏輯型程式語言（logic programming language）之特性？（10 分）
(二)試舉例說明何以邏輯型程式語言較適合發展專家系統（rule-based Systems）之雛形（prototype）。（10 分）

【擬答】
一、邏輯型程式語言特性：
(一)以某種符號邏輯(symbolic logic)的形式來描述程式：可描述命題(proposition),命題間的關係(relationship)和新的命題如何由已有的命題推導(inference)出來。
(二)以邏輯推論過程(logical inference process)去產生結果：程式不用描述結果如何被計算出來，而是只要描述事實以及推論結果的相關規則即可，語言會自動以現有事實和推論規則推導出結果。
(三)宣告型語意(declarative semantics)：以一種簡單方式去決定每個敘述的意義，即一個命題(proposition)的意義可以完全由該敘述本身決定而毋須參考上下文。
二、專家系統通常由以下三部份構成
(一)知識庫(knowledge base)：搜集人類的知識，將之有系統地表達或模組化，使電腦可以進行推論，解決問題。
(二)推論器(inference mechanism)：推論器的是藉由演算法或決策策略來進行與知識庫內各項專門知識的推論, 依據使用者的問題來推得正確的答案.
(三)介面(interface)：介面的主要功能是提供相關資料的輸入與輸出，而邏輯型程式語言，例如：Prolog 解決問題的方式和專家系統類似，以符號邏輯(symbolic logic)表達知識庫，而語言本身內含推論引擎(inference engine)可以依照知識庫中的事實及推論規則去推導結論故適合用來發展專家系統。

二、(一)試說明資料抽象化（data abstraction）在物件導向程式語言（object-oriented programming languages）中的角色及重要性。（10 分）
(二)試舉例說明資料抽象型態（abstract data types）參數化（parameterized）之功能。（10 分）

【擬答】
一、(一)資料抽象化所扮演的角色：資料抽象化是物件導向程式語言的三大特性之一，另外兩個特性爲，繼承(inheritance)以及動態繫結(dynamic binding)。
(二)資料抽象化的重要性：
1.達成程式模組化，使得資料可以容易被重複利用，因爲資料抽象化將資料以及相關的動作均包含在抽象資料型態中，使用者以抽象資料型態所提供的動作來操作資料。
2.可靠性高，由於抽象資料型態將資料的實際表達方式隱藏在資料形態內，使用者只能依靠抽象資料型態本身所提供的特定方式存取資料，故較安全，並且改變抽象資料型態內部表達方式不會影響到使用者。
二、抽象資料型態參數化的功能，是讓抽象資料型態可以只要描述內部實際儲存資料的基本形式，而不用具體描述所儲存資料的資料型態，實際儲存的資料型態可以參數的方式傳遞到此抽象資料型態中。例如：以 C++的 template 爲例，堆疊實際大小以及所存元素的型態，可以以參數方式來指定：

```
template<class T, int SIZE>
class Stack {
    void push(T data);
    T pop();
    T dataArray[SIZE];
};

void main() {
    Stack<int, 5> s1; //產生一個可以存整數，容量爲 5 的堆疊
   Stack<double, 10> s2; //產生一個可以存浮點數，容量爲 10 的堆疊
}
```

三、(一)試說明程式撰寫與應用系統開發如何確保軟體安全（software security)。(10 分)
(二)試舉例說明如何作系統測試及系統效能評估（benchmarks)。(10 分)

【擬答】

一、應用系統開發部分和程式撰寫確保軟體安全的方式：在軟體開發流程中即把安全性(security)納入，通常採用的流程如下：

(一)需求制訂階段(Requirement and use cases)：在此階段中將已知以及未來可能遇到的的安全性需求納入考量，並且製作實際發生安全狀況的例子，來探討遇到安全性問題時軟體的反應。

(二)軟體架構設計階段(Design and architecture)：在此階段制定統一的安全性處理機制，例如：統一的例外處理機制，並且將程式中的假設明確標出。

(三)制定安全性測試計劃(Security test plan):將可能發生的安全性問題，以及測試方法訂出。

(四)程式撰寫階段(Coding):此階段中專注於實做上可能造成的程式問題，例如緩衝區溢位，撰寫者必須考慮所有程式執行流程的可能性，並且利用程式分析工具，去分析程式可能潛在的問題。

(五)程式測試階段(Testing)：此階段依照事先定義的測試計劃，產生適當的測試案例(test case)去測試軟體遇到攻擊時是否能夠正常執行。

(六)軟體維護階段(Maintenance)：此階段會建立基本回歸測試案例(Regression cases)，確保未來軟體的更動不會影響到已有的功能。

二、(一)系統測試：將整個系統當做一個黑箱(black box)，對系統內部實做細節不做任何假設，然後藉由各種輸入資料與系統執行後所得的結果做比較，來測試此系統是否有滿足需求，其中又分爲功能性測試與效能測試，分別用來測試系統是否符合功能正確，以及效能是否符合需求。

(二)系統效能評估：通常是藉由效能評估工具(Profiling tool)來分析系統的效能。例如：若是分析統計軟體的效能, 則可以將大量統計資料餵入系統中執行，依照效能評估軟體來分析其執行速度以及記憶體使用量。若是分析顯示卡硬體效能, 則可以將大量複雜圖形資料送入顯示卡中，利用顯示卡的效能評估工具來測試顯示的速度。

四、試說明競爭（competition）及合作（cooperation）、同步（synchronization）之差異，並以任一並行（concurrent）程式語言舉例說明之。(20 分)

【擬答】

一、合作同步(cooperation synchronization):要求任務 A 必須等待另一任務 B 完成某項特定工作後，任務 A 才能繼續執行。
以生產者消費者問題爲例(Ada 語言)：只有一個整數共用資料，Consumer 必須等待 Producer 產生完資料才能消費。

```
task BufferTask;
entry produce(item:in integer);
entry consume(item: out integer);
end BufferTask;
task body BufferTask is
      buffer : integer;
      buffer_full : boolean := false;
loop
 select when not buffer_full =>
           accept produce(item: in integer) do
                buffer := item;
                buffer_full := true;
end produce;
      or when buffer_full =>
           accept consume(item: out integer) do
                item := buffer;
                buffer_full := false;
           end consume;
      end select;
 end loop;
 end BufferTask;

task Producer;
task Consumer;
task body Producer is
      new_value:integer;
      begin
```

```
            loop
                BufferTask.produce(new_value);
            end loop;
    end Producer;
    task body Consumer is
        sotred_value: integer;
        begin
            loop
                BufferTask.consume(stored_value);
            end loop;
        end Consumer;
```

二、競爭同步(competition synchronization)：允許兩個或兩個以上的任務競爭使用共享資料，沒有規定的使用順序，但必須是互斥的使用共享資料來達到資料同步的要求。以生產者消費者問題爲例(Ada 語言)：有一組環狀佇列的共用資料，可以讓多個 Consumer 和 Producer 去互斥存取此 Buffer。

```
    task body BufferTask is
        buffer : array(1..100) of integer;
        filled : integer range 0..100 := 1;
        next_in, next_out : integer range 1..100 := 1;
        loop
            select when filled < 100 =>
                accept produce(item: in integer) do
                    buffer(next_in) := item;
                end produce;
                next_in = (next_in mod 100) + 1;
                filled := filled + 1;
          or when filled > 0 =>
                accept consume(item : out integer) do
                    item := buffer(next_out);
                end consume;
                next_out := (next_out mod 100) + 1;
               filled := filled - 1;
            end select;
        end loop;
    end BufferTask;
```

五、在嵌入式系統（embedded systems）中發展程式要較非嵌入式系統多考慮 CPU 及記憶體（memory）等之資源使用問題，試說明如何達成上述議題，並可節省電能。(20 分)

【擬答】

嵌入式系統通常建立在特定的硬體平台上，所以除了一般程式撰寫上的技巧外還可以利用特定硬體提供的功能，來加快執行的速度或是記憶體使用。說明如下：

一、使用像鏈節串列等的動態資料結構，而非靜態陣列之類的靜態資料結構，避免浪費未使用到的空間。

二、自行管理記憶體，在應用程式端設計記憶體管理機制，回收不用的記憶體重複使用，避面記憶體洩漏(memory leakage)，同時也可以減少和作業系統要求記憶體的次數以加快程式速度。

三、少使用全域變數和靜態變數節省記憶體用量。

四、多使用副程式來代替巨集(macro)，減少執行檔空間。

五、避免將大的副程式變成嵌入式副程式(inline subprogram)，減少執行檔空間。

六、利用更有效率的運算來代替原本的運算，例如：以位元移動代替乘/除 2 的次方。

七、避免在迴圈結構中產生物件，應該將產生物件敘述移到迴圈外。

八、參數傳遞多使用傳位址方式傳遞，避免函式的參數傳遞使用傳值方式。

九、可以把常存取的變數指定爲暫存器變數(register variable)，讓編譯器使用此變數時會先將變數值放到 CPU 的暫存器上，就不用每此取用此變數均需要到記憶體中取值。

十、在編譯時加入最佳化參數，使編譯器對原始程式碼進行與機器有關的最佳化(machine dependent optimization)和與機器無關的最佳化(machine independent optimization)產生更有效率的目的碼。

十一、可使用並行方式(concurrency)來設計程式，將處理 I/O 的部份獨立爲一執行緒, 專門處理其餘的運算已另一執行緒來處理，提高 CPU 使用率來達成形成所需時間較短。

十二、可以使用的快取記憶體部分調大，避免一些重複的 I/O。
十三、利用硬體提供的特殊指令來處理運算，例如：若此嵌入式系統有支援整數加速運算，則儘量使用整數型態在程式中運算。