《資料結構》

試題評析	今年檢事官資料結構的考題相當簡單，主要都在測驗應試者的觀念，沒有複雜及刁鑽的問題。第一題是測驗應試者是否了解資料結構、演算法與程式之間的關聯性。第二題則是基本的運算式換的問題，大部份著重在觀念較多。第三題是幾個基本線性資料結構的觀念題，取分不難。第四題必須先了解題目的需求，只是要檢查是否存在到每一個頂點的路徑，故只需要使用 DFS 或 BFS 即可，不需要使用像 Dijkstra 一類複雜的演算法。今年應試者在資料結構一科中可拿到不錯的分數，分數應可取得 60 分以上，程度較佳者要拿到 80 分的成績應有可為。

一、請論述 Data Structures + Algorithms = Programs 之意義。 (二十五分)

【擬答】

當要撰寫一個程式時，在了解到問題的要求之後，通常必須先設計所需要的演算法，然後再依演算法所需要的運算，選擇最適當的資料結構，來減少運算所需要的時間，然後評估其時間效能是否符合要求；如果不符合需求，就必須嘗試修改演算法或更改使用的資料結構，看看是否可以將執行時間再降低。

在確認演算法與資料結構之後，可以再選擇適當的程式語言，以便將程式實際撰寫出來。程式撰寫時，先定義資料結構所要儲存的資料的儲存結構，例如陣列、鏈結串列的節點結構，或所需要的變數等等；再寫輸入資料的程序，將輸入資料以預先選好的資料結構形式儲存下來；然後撰寫處理的程序，處理程序通常就是根據規劃好的演算法步驟，一步步寫出處理的程式命令；最後撰寫輸出結果的程序，而將程式完成。

因此，撰寫程式前，所需要的準備工作，就是要先規劃好演算法，使解決問題的過程以明確的步驟呈現出來；而資料結構的規劃，則是牽涉到資料在電腦的記憶體中的儲存方式，如何能協助演算法在最短的時間內，完成其每個步驟的功能。

二、(一)在什麼狀況需要把 infix expression，例如 a + b 改成 postfix expression ab +？(十分)

(二)轉換時要用那一資料結構來做轉換。(五分)

(三)請敘述轉換的方法？(十分)

【擬答】

(一)當電腦要計算算術式的結果時，因為 infix expression 的計算方式較為複雜，所以通常會先將 infix expression 先轉換成為 postfix expression，再加以計算，這樣會比較好處理。此外，compiler 中也常將 infix expression 先轉換成為 postfix 的中間碼，然後根據 postfix expression 翻譯成最後的機器目的碼。另外還有一種情形，就是計算運算式的機器結構是 stack machine，此時也必須將 infix expression 先轉換成為 postfix 形式再計算。

(二)infix expression 轉換為 postfix 形式時，需要使用 stack 結構，因為 postfix expression 的計算是由前而後掃瞄 postfix expression，先遇到的 operator 會先計算。因此，infix expression 中 priority 較低的 operator，通常會先被置於 stack 中，在較後的階段才輸出到 postfix expression，以配合 postfix expression 的 evaluation rule。

(三)infix expression 轉換成 postfix expression 的方法如下：

1.由前而後逐一掃瞄 infix expression 的 element。

2.若 element 為 operand，則直接輸出。

3.若 element 為 operator，則必須與 stack top 的 operator 比較其計算的先後次序。

如果 element 須優先計算，則將 element push 到 stack 中；

如果 stack top 的 operator 須先計算時，則將 stack top 的 operator 先取出並且輸出，然後重新做步驟 3。

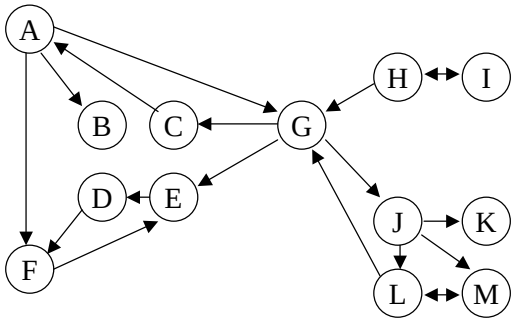
4.當掃瞄完 infix expression 後，若 stack 仍中還積存有 operator(s)，則必須將這些 operator(s) 逐一取出並且輸出。

三、請比較 array，list 及 queue 的使用時機，並舉例說明其優缺點。(二十五分)

【擬答】

	使用時機	優點的例子	缺點的例子
Array	需要對其中的資料項目，進行直接存取時，使用陣列會較爲適合。	二分搜尋法、雜湊表。	不適合經常 insert/delete 的資料
List	經常需要插入新資料或刪除舊資料的動態資料，比較適合使用 linked list 來儲存，可以減少所需要的資料搬動次數。	經常要 insert/delete 的一群資料	不適合做二分搜尋
Queue	適合於依照 FIFO 方式處理資料時，適合使用 queue。	FIFO scheduling	不適合做資料次序的反轉動作

四、給予一有向圖（directed graph）如下： (二十五分)



若存在一路徑可以從 x 到 y，且存在一路徑可以從 y 到 z，則必可找到一個路徑從 x 到 z。以上圖來看，有一個路徑可以從 A 到 G，並有一個路徑可以從 G 到 E，則，我們可以找到一個路徑從 A 到 E。但從上圖中，我們卻無法找到一個路徑從 A 到 I。請問，該以何種資料結構來幫助你找到所有的路徑？如何做？意即從圖中的任一點出發，是否存在路徑可以到達圖中之其他點。

【擬答】

可以使用 DFS(Depth-First Search)或 BFS(Breadth-First Search)來找路徑。以下是以 DFS 來處理。

```
push (起點);
while stack not empty do
 begin
     u <- pop();
     If   u is not visited then
       begin
           mark u is reachable;
           for each v adjacent from u do
                if v is not visited then Push(v);
       end;
 end;
```

資料結構：

(1)使用 adjacency list 來記錄圖形的 vertices 與 edges 的關係。

(2)使用 stack 來記錄可以到達的 vertices。