實作 Binary Search Tree (BST)

一. 簡介:

Binary search tree 是一種常見用來儲存資料的結構，本身是一棵樹，樹上的每個節點都用來儲存資料，每個 node 只有兩個 child，且對每一個 subtree 的 root 來說，以 right child 為 root 的 subtree 裡面所有 node 都小於 root，以 left child 為 root 的 subtree 裡面所有 node 都大於 root。

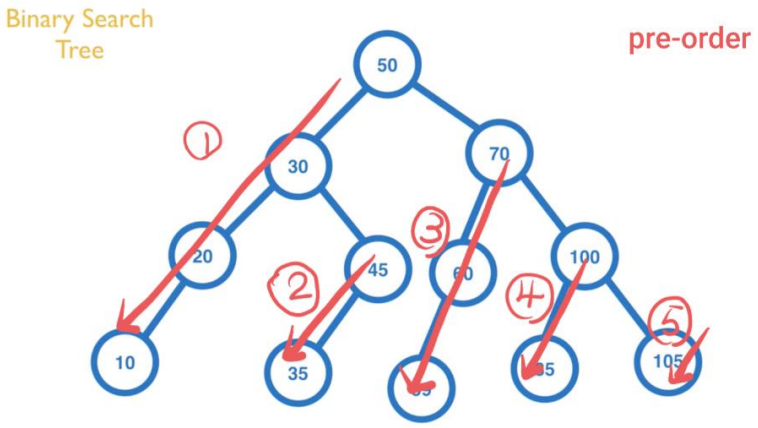
二. 程式碼說明/主要的函式用途:

(1) 定義一個名為 TreeNode 的 class，用來代表 BST 上的每一個 node，TreeNode 的 data member 包括用來存資料的 val，和指向左右 children 的 pointer（ left、right ），member function 則為 constructor，用來把 val 初始為 0，left 和 right 初始為 NULL。

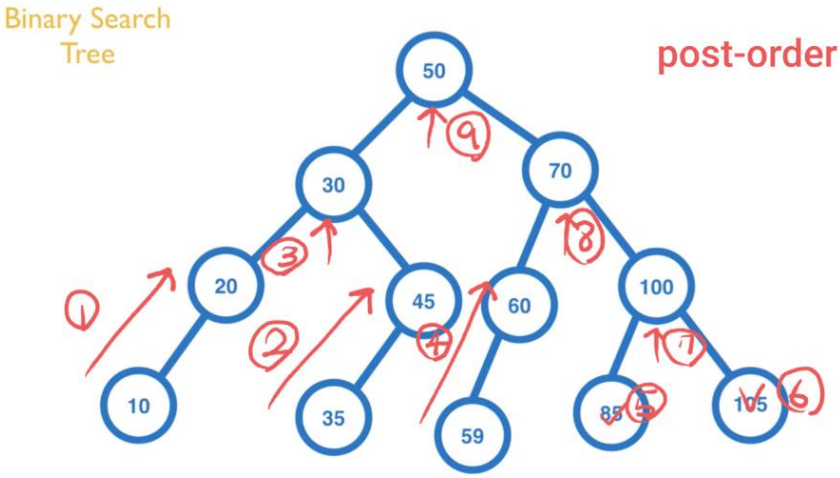
(2) 定義一個名為 BST 的 class，代表要實作的 Binary Search Tree，data member 為指向整棵樹 root 的 pointer，而 member function 有 constructor，初始 root 值為 NULL，getter（ getTree ），用來回傳 root 的地址，還有對 BST 的一系列操

作，包括（ 以下所有操作，若為空樹，則程式會報錯 ）：

1. Depth ()：回傳 BST 的樹高度。
2. preorderTraversal ()：輸出由上往下，由左至右（中左右）的順序。



3. inorderTraversal ()：輸出由小到大的順序，也就是從 root 開始，先把左邊的輸出，在輸出 root，最後再輸出右邊的（左中右）。
4. postorderTraversal ()：輸出由下往上，由左至右（左右中）的順序。



5. levelorderTraversal ( ): 以水平的方式由上往下依序輸出所有 node。
6. insert_to_leaf ( ): 新增 node 到 BST 時，若值比 root 大，往右下 insert，若比 root 小，往左下 insert，然後以同樣的邏輯遞迴往下檢查，直到 insert 到 leaf 的位置。（ 若插入已存在的 node，則程式會報錯 ）
7. maxKey ( ): 輸出整棵 BST 中，node 最大的值。
8. minKey ( ): 輸出整棵 BST 中，node 最小的值。
9. successor ( ): 給定 BST 中一個 node，輸出所有比 node 大的值中最小的一個，若給定最大值，則程式報錯。
10. predecessor ( ): 給定 BST 中一個 node，輸出所有比 node 小的值中最大的一個，若給定最小值，則程式報錯。
11. delNode ( ): 給定要刪除的 node，程式執行完後 BST 內就不存在此 node。(若刪除不在 BST 中的 node，則程式會報錯)

(3) main 主函式內部:

先建造一棵 BST (tree)，之後用 do while 一直重複執行一系列操作，直到使用者輸入 0，則結束程式。

三.程式輸入/輸出:

輸入:

(1) 每一輪使用者輸入要對 BST 執行的操作，直到使用者輸入 0 則結束操作，程式終止。

輸出:

(1) 一開始程式碼會告知使用者輸入何種指令對應到的操作，在每一輪使用者輸入完指令後，程式會輸出相對應的指令結果。

四.程式執行範例

```
歡迎使用Binary Search Tree!
operation:
1: Tree Height
2: Preorder
3: Inorder
4: Postorder
5: levelorder
6: Insert a node
7: Max
8: Min
9: Successor
10: Predecessor
11: Delete a node
請輸入要執行的指令: 1
樹的最大高度為: 0
請輸入要執行的指令: 2
This is an empty tree
請輸入要執行的指令: 3
This is an empty tree
請輸入要執行的指令: 4
This is an empty tree
請輸入要執行的指令: 5
This is an empty tree
請輸入要執行的指令: 7
This is an empty tree
```

```
請輸入要執行的指令: 8
This is an empty tree
請輸入要執行的指令: 9
This is an empty tree
請輸入要執行的指令: 10
This is an empty tree
請輸入要執行的指令: 11
This is an empty tree
請輸入要執行的指令: 0
執行結束
Process returned 0 (0x0)   execution time : 19.915 s
Press any key to continue.
```

```
歡迎使用Binary Search Tree!
operation:
1: Tree Height
2: Preorder
3: Inorder
4: Postorder
5: levelorder
6: Insert a node
7: Max
8: Min
9: Successor
10: Predecessor
11: Delete a node
請輸入要執行的指令: 6
請輸入要插入的值: 14
插入完成
請輸入要執行的指令: 6
請輸入要插入的值: 20
插入完成
請輸入要執行的指令: 6
請輸入要插入的值: 10
插入完成
請輸入要執行的指令: 6
請輸入要插入的值: 55
插入完成
請輸入要執行的指令: 6
請輸入要插入的值: 30
插入完成
請輸入要執行的指令: 6
請輸入要插入的值: 24
插入完成
請輸入要執行的指令: 6
請輸入要插入的值: 1
插入完成
請輸入要執行的指令: 6
請輸入要插入的值: 4
插入完成
請輸入要執行的指令: 1
樹的最大高度為: 5
請輸入要執行的指令: 5
the levelorder of the BST:
level 1 : 14
level 2 : 10 20
level 3 : 1 55
level 4 : 4 30
level 5 : 24
請輸入要執行的指令: 2
the preorder of the BST: 14 10 1 4 20 55 30 24
請輸入要執行的指令: 3
the inorder of the BST: 1 4 10 14 20 24 30 55
請輸入要執行的指令: 4
the postorder of the BST: 4 1 10 24 30 55 20 14
請輸入要執行的指令: 7
BST最大值是: 55
請輸入要執行的指令: 8
BST最小值是: 1
```

```
BST最小值是: 1
請輸入要執行的指令: 9
請輸入要查詢的successor值: 55
55 is the Max of the Tree, the successor of 55 doesn't exist
請輸入要執行的指令: 9
請輸入要查詢的successor值: 30
the successor of the 30 is 55
請輸入要執行的指令: 10
請輸入要查詢的predecessor值: 1
1 is the Min of the Tree, the predecessor of 1 doesn't exist
請輸入要執行的指令: 10
請輸入要查詢的predecessor值: 14
the predecessor of the 14 is 10
請輸入要執行的指令: 11
請輸入要刪除的值: 24
刪除完成
請輸入要執行的指令: 5
the levelorder of the BST:
level 1 : 14
level 2 : 10 20
level 3 : 1 55
level 4 : 4 30
請輸入要執行的指令: 11
請輸入要刪除的值: 10
刪除完成
請輸入要執行的指令: 5
the levelorder of the BST:
level 1 : 14
level 2 : 1 20
level 3 : 4 55
level 4 : 30
請輸入要執行的指令: 0
執行結束
Process returned 0 (0x0)   execution time : 113.329 s
Press any key to continue.
```